Merging news reports that describe events

Anthony Hunter and Rupert Summerton
Department of Computer Science
University College London
Gower Street, London WC1E 6BT, UK

June 9, 2005

Abstract

Many kinds of news report provide information about events. For example, business news reports in the area of mergers and acquisitions, provide information about events such as "company X making a bid for company Y", or "takeover of company Y by company X being rejected by the anti-trust authorities". Futhermore, news reports do not normally exist in isolation. There is an underlying narrative which concerns a number of entities related in some way over a period of time. In many domains, stories will follow a stereotypical sequence. For example, a particular takeover may involve a bid being made, a rejection by the target board, a rise in the bid value by the potential buyer, a recommendation of acceptance by the target board, acceptance by the shareholders, and finally successful completion of the takeover. In order to merge heterogeneous news reports that describe events, we need to identify and reason about the events being described prior to merging them. In this paper, we investigate this problem with a focus on structured news reports. Each structured news report (SNR) is an XML document, where the textentries are restricted to individual words or simple phrases, such as names and domain-specific terminology, and numbers and units. We assume SNRs do not require natural language processing. As each SNR is isomorphic to a term in logic, we use a logic-based approach to extract relevant information about the events being described in the reports to be merged. We then provide a new version of the event calculus to assimilate the information from the various reports, to obtain the most up-to-date and complete picture of the events being described. Finally, from this assimilated information, we generate an SNR as the output.

1 Introduction

Syntactically, a structured news report (SNR) is a data structure containing a number of grammatically simple phrases together with a tag (giving semantic information) for each phrase. Each phrase that is tagged is a textentry. The set of tags in an SNR is meant to parameterize a stereotypical situation, and so a particular SNR is an instance of that stereotypical situation. For example, news reports on corporate acquisitions can be represented as SNRs using tags including `buyer`, `seller`, `acquisition`, `value`, and `date`. Each phrase in an SNR is very simple, such as a proper noun, a date, or a number with unit of measure, or a word or phrase from a prescribed lexicon. For an application, the prescribed lexicon delineates the types of states, actions, and attributes, that could be conveyed by the SNRs.

We represent each SNR by an XML document. The definition for an SNR is very general. In practice, we would expect a document type definition (DTD) for a given domain. A DTD is a standard way of defining the legal building blocks of an XML document (for more information on DTDs, see www.w3.org). So for example, we would expect that for an implemented system that merges weather reports, there would be a corresponding DTD. One of the roles of a DTD, say for weather reports, would be to specify the minimum constellation of tags that would be expected of a weather report. We may also expect integrity constraints represented in classical logic to further restrict appropriate SNRs for a domain.

In order to merge SNRs, we need to take account of the contents of the SNRs. Different kinds of content need to be merged in different ways. To facilitate this, we can reason about the information in SNRs in logic. In our approach, each SNR can isomorphically be represented as a logical term: Each tagname is a function symbol, and each textentry is a constant symbol. Furthermore, subtrees of an SNR can be isomorphically represented as subterms in logic. In this way, the information in each SNR can be captured in a logical language. We can then define a range of predicates, in a Prolog knowledgebase, that capture useful relationships between SNRs, and so a set of SNRs can then be analysed or merged as Prolog queries to a Prolog knowledgebase. In this way, a query to merge some SNRs can be handled by recursive calls to Prolog to merge the subtrees in the SNRs. This gives a context-dependent logic-based approach to merging that is sensitive to the information in the SNRs and to the background knowledge in the Prolog knowledgebase.

In this paper, we focus on SNRs that provide information about events. Many kinds of news report provide information about events. For example, business news reports in the area of mergers and acquisitions provide information about events such as "company X makes a bid for company Y", or "takeover of company Y by company X is rejected by the anti-trust authorities". In order to merge heterogeneous news reports that describe events, we need to identify and reason about the events being described prior to merging them.

Normally, news reports do not exist in isolation. They are usually part of narratives which relate them to other articles that deal with the same story. For example, in the mergers and acquisitions domain, we may find a news report announcing a takeover bid followed by a news report of the bid being accepted by the board, followed by a report on the shareholders voting whether to accept the bid, and so on. All news reports belong to at least one narrative and each narrative involves one or more reports.

Example 1.1 *Consider the following two conflicting business reports. The left report states that the value of the bid is* `$35Billion` *and the capital of the target is* `$25Billion`*, and the right report states that the value of the bid is* `$47Billion` *and the capital of the target is* `$50Billion`*.*

```
⟨businessreport⟩                                  ⟨businessreport⟩
        ⟨source⟩ Reuters ⟨/source⟩                        ⟨source⟩ Reuters ⟨/source⟩
        ⟨action⟩ New Bid Made ⟨/action⟩                   ⟨action⟩ Board Accepts Offer ⟨/action⟩
        ⟨bidvalue⟩ $35Billion ⟨/bidvalue⟩                 ⟨bidvalue⟩ $47Billion ⟨/bidvalue⟩
        ⟨buyer⟩ Shell ⟨/buyer⟩                            ⟨buyer⟩ Shell ⟨/buyer⟩
        ⟨target⟩                                          ⟨target⟩
                ⟨company⟩ Texaco ⟨/company⟩                       ⟨company⟩ Texaco ⟨/company⟩
                ⟨capital⟩ $25Billion ⟨/capital⟩                   ⟨capital⟩ $50Billion ⟨/capital⟩
        ⟨/target⟩                                         ⟨/target⟩
⟨/businessreport⟩                                 ⟨/businessreport⟩
```

We can merge them to resolve these apparent conflicts. For this, we are drawing on domain knowledge concerning the relative ordering of actions. For example, the action `Board Accepts Offer` *would occur after* `New Bid Made`*. So in this case, the right report involves a more recent event involving the action of* `Board Accepts Offer`*, and so provides facts about a more up-to-date state.*

```
⟨businessreport⟩
        ⟨source⟩ Reuters ⟨/source⟩
        ⟨action⟩ Board Accepts Offer ⟨/action⟩
        ⟨bidvalue⟩ $47Billion ⟨/bidvalue⟩
        ⟨buyer⟩ Shell ⟨/buyer⟩
        ⟨target⟩
                ⟨company⟩ Texaco ⟨/company⟩
                ⟨capital⟩ $50Billion ⟨/capital⟩
        ⟨/target⟩
⟨/businessreport⟩
```

Example 1.2 *Consider the following two conflicting business reports. The left report states that the action* *is* `New Bid Made` *and the right report states that the action is* `Board Accepts Offer`.

```
⟨businessreport⟩                              ⟨businessreport⟩
        ⟨source⟩ Reuters ⟨/source⟩                    ⟨source⟩ DowJones ⟨/source⟩
        ⟨bidvalue⟩ $47Billion ⟨/bidvalue⟩             ⟨bidvalue⟩ $47Billion ⟨/bidvalue⟩
        ⟨action⟩ New Bid Made ⟨/action⟩               ⟨action⟩ Board Accepts Offer ⟨/action⟩
        ⟨buyer⟩ Shell ⟨/buyer⟩                        ⟨target⟩ Texaco ⟨/target⟩
        ⟨target⟩                                      ⟨buyer⟩
                ⟨company⟩ Texaco ⟨/company⟩                   ⟨company⟩ Shell ⟨/company⟩
                ⟨sector⟩ oil ⟨/sector⟩                        ⟨sector⟩ oil ⟨/sector⟩
        ⟨/target⟩                                     ⟨/buyer⟩
⟨/businessreport⟩                             ⟨/businessreport⟩
```

We can merge them so the more recent event is used. But we can also take the information from the previous report that is not invalidated by the newer report.

```
⟨businessreport⟩
        ⟨source⟩ Reuters and DowJones ⟨/source⟩
        ⟨action⟩ Board Accepts Offer ⟨/action⟩
        ⟨bidvalue⟩ $47Billion ⟨/bidvalue⟩
        ⟨buyer⟩
                ⟨company⟩ Shell ⟨/company⟩
                ⟨sector⟩ oil ⟨/sector⟩
        ⟨/buyer⟩
        ⟨target⟩
                ⟨company⟩ Texaco ⟨/company⟩
                ⟨sector⟩ oil ⟨/sector⟩
        ⟨/target⟩
⟨/businessreport⟩
```

In order to represent and reason with narratives, we need to model states and changes of state. To address this need, we use an event reasoning system based on a variant of the event calculus to represent and reason with such information. Event calculus was originally proposed by Kowalski and Sergot [KS86]. Since then a number of variants of event calculus have been proposed. For a review see [Sha99, MS99]. These variants of event calculus have been presented in either classical logic or in Prolog. In the following sections, we present a new version of event calculus based on meta-level classical logic that aims at reasoning about events in news reports. We also discuss the Prolog implementation of our version of the event calculus that we have used in a case study with simple business news reports.

We present our framework for taking a set of SNRs as input and produce a merged SNR as output in the following steps: (Section 2) We present the basic assumptions for our approach to event-based merging; (Section 3) We specify the format for the input; (Section 4) We cover the basic definitions for our object-level and meta-level languages; (Section 5) We present a format for axioms for extracting information about events from the set of input reports; (Section 6) We present transition models as a way of representing key inter-dependencies between events; (Section 7) We present our new variant of event calculus for reasoning about events in news reports; (Section 8) We present aggregation rules as the way of defining how to merge information to generate the output SNR; and (Section 9) We discuss our implementation in Prolog.

The net contribution of this paper is a specification for representing and reasoning with events arising in SNRs, together with some insights into how this can be implemented.

2 Assumptions about the domain for event-based merging

Now we clarify our main assumptions about the domain for event-based merging.

Action For any application domain, we assume a set of actions that can occur in the domain. We also assume a partial ordering over a set of actions. For example, "New Bid Made" is ordered before

"Board Accepts Offer", which in turn is ordered before "Shareholders Accept Offer". The partial ordering specifies the actions that may potentially follow an action. So for example, for the action "Bid Made", either the action "Board Accepts Offer" or the action "Board Rejects Offer" may follow it. For a larger example, see Figure 1.

Event An event is an instantaneous (i.e. durationless) action together with some information (represented by facts) about the action, such as the entities involved and attributes concerning the nature of the action. We have no general requirements for the information that needs to be represented about an event. The types of actions, entities, and attributes depend on the application domain. In the case of mergers and acquisitions, the entities include companies, shareholders, and boards, and the attributes include the share price and the bid value. In Example 1.1, the top left report is about the event that Shell has made a new bid for Texaco. This event would involve the action "New Bid Made", the entities would be Shell and Texaco, and an attribuite would be the price of $35billion. For a given domain, we use an ordering over actions to specify a partial ordering relation over a set of events. If E is an event involving action A, and E' is an event involving action A', and E and E' have the same subject, and A occurs before A' in the partial ordering over actions, then E occurs before E'. We will explain how we do this in the following sections.

State A state is a set of ground facts that hold for a time period. A state model is a set of states. For example according to the top left report in Example 1.1, there is a state in which Shell has made a $35billion bid for Texaco, and in which Texaco has a capitalization of $25Billion. This state can be represented by the facts

$$\begin{array}{l}\texttt{bidvalue}(\texttt{Shell}, \texttt{Texaco}, \$\texttt{35billion})\\ \texttt{capitalization}(\texttt{Texaco}, \$\texttt{25billion})\end{array}$$

We determine a fact holds in a state if (1) it is a result of the event that starts the state or (2) it held in the state before the event and it has not been terminated by the event. The second condition can be motivated by the following examples. If the headquarters location of company X is Z when X makes a bid for company Y, then when the board of Y accepts the offer, this new event does not terminate the fact that the headquarters location of company X is Z. However, if the bid value is V before the event the of "New Bid Made", then the bid value will no longer be V after this event. Again, we will explain how we do this in the following sections.

So the aim of event-based merging is to find the most up-to-date information in the input SNRs and use this to construct an output SNR that contains the most up-to-date information (facts). Our approach is to use the input SNRs to construct a state model. The most up-to-date information is the information in the last state (the most recent state) in the state model.

3 Input for event-based merging

We use XML to represent SNRs. Each structured news report is an XML document, but not vice versa, as defined below.

Definition 3.1 *If ϕ is a tagname (i.e. an element name), and ψ is textentry (i.e. a phrase represented by a string), then $\langle\phi\rangle\psi\langle/\phi\rangle$ is a SNR. If ϕ is an tagname and $\sigma_1, ..., \sigma_n$ are SNRs, then $\langle\phi\rangle\sigma_1...\sigma_n\langle/\phi\rangle$ is a SNR.*

We assume that the textentries in SNRs are heterogeneous in format. For example, the format of date values is unconstrained (12/12/1974; 31st Dec 96; 12 Nov 2001 etc.) as is the format of numbers and currency values (3 million; 3, 000, 000 GBP; $4, ¥500K etc.). Elsewhere, we have discussed how this heterogeneity can be handled in logic by various kinds of equivalence axioms [Hun02a, Hun02b, HS04].

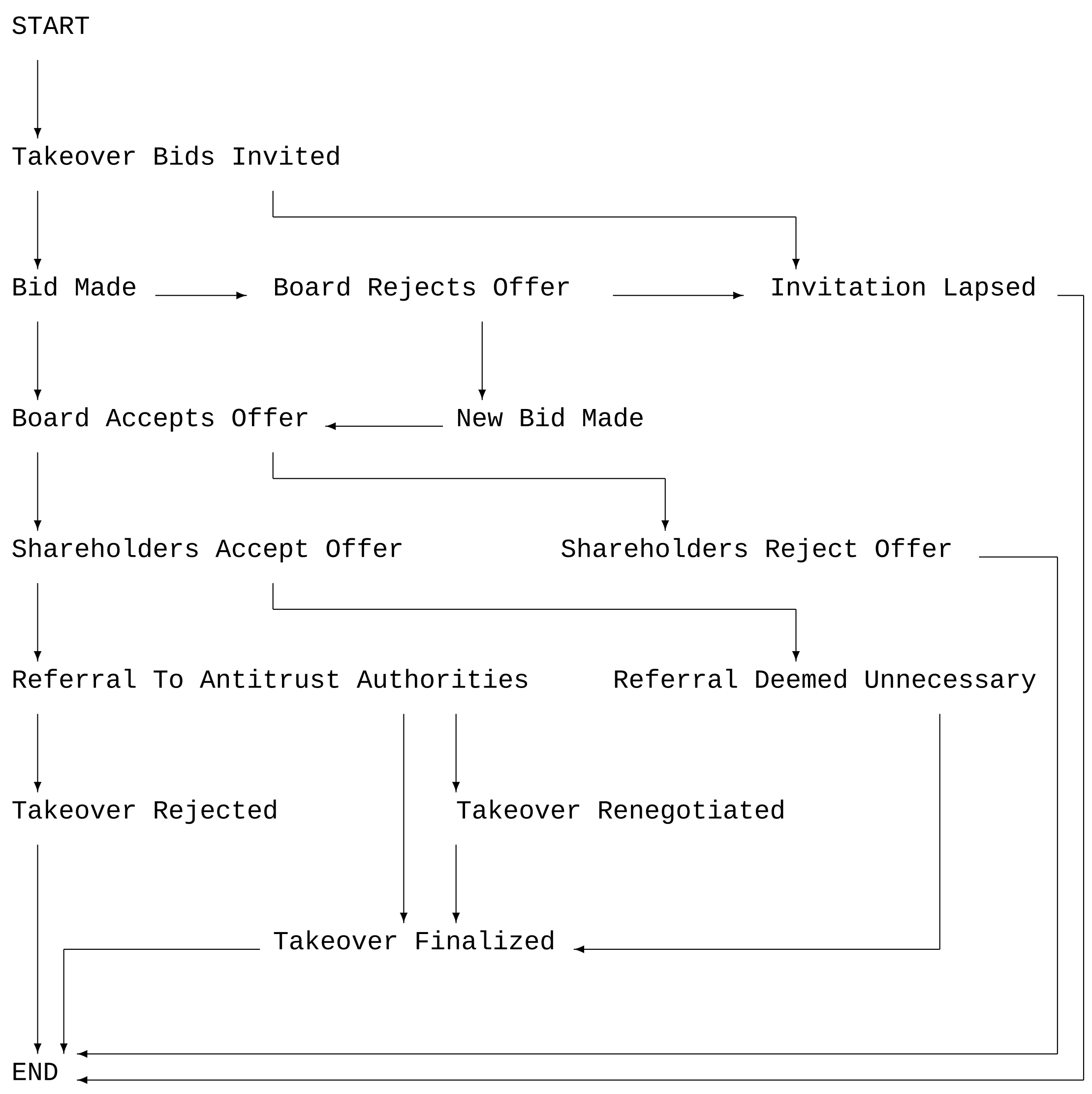

Figure 1: A graphical representation of a partial ordering of some actions for the mergers and acquisitions domain. For an action at a node, the outgoing arcs go to the possible actions that can follow it next.

Now we consider four main assumptions used for the SNRs that are the input to event-based merging.

1. **Each SNR describes exactly one event**. So each SNR has sufficient information to determine the subject of the event and the event type (i.e. action). The restriction of each SNR describing exactly one event type is reasonable for applications such as handling newsfeeds, where each news report would normally describe a single event. Furthermore, this assumption is useful if we generate SNRs using an information extraction system. Such systems extract information from free text, using natural language processing techniques, and use this extracted information to complete a template [CL96]. These templates are commonly restricted to single issues, such as individual events. Each completed template can be handled as an SNR.

2. **Each SNR does not contain any explicit temporal information** such as a timestamp for when the report was written or published or when the event occurred. The reason we have introduced this constraint is to show that we can often reason about events without recourse to explicit temporal information. Clearly if we also have explicit temporal information in our SNRs, then we can extend the proposal in this paper to take advantage of this information. Note that even if we did have explicit temporal information in our SNRs, it may be insufficiently discriminating on its own to resolve some problems in merging. For example, it is possible to get two SNRs with the same publication time, but one refers to a later event than other. If we are unable to identify and reason about the relative ordering of the events, we would be unable to adequately handle these examples.

3. **Each SNR is not necessarily a complete description of an event**. In other words, we do not assume that an SNR necessarily provides all the relevant information about an event. So if we have two SNRs describing the same event, then there may be information in one SNR that is not in the other. For example, consider two SNRs describing the event that company X has made a bid for company Y: One of the SNRs may include the name of the chief executive of X, and the other may include the location of the headquarters of X. See also Example 1.2.

4. **Each SNR is correct**. In other words, the information an SNR provides is regarded as correct at the time of the event occuring. So if we have two SNRs describing the same event, then there will be no inconsistency between them. For example, consider two SNRs describing the event that company X has made a bid for company Y: If one of the SNRs has the value as V_1 and the other SNR has the value as V_2, then V_1 and V_2 are interchangeable though not necessarily identical: Perhaps one is in US dollars and the other in Euros.

In future papers, we will investigate relaxing some of these assumptions. In particular, we will investigate the interplay between temporal and event information obtained from SNRs, and we will investigate techniques for handling incorrect and inconsistent event information in SNRs.

Clearly each SNR is isomorphic to a ground term where each tagname is represented by a function symbol and each textentry is represented by a constant symbol. Hence, we can represent each SNR by a ground logical atom in classical logic as follows.

Definition 3.2 *Let $\langle\phi\rangle\sigma_1, .., \sigma_n\langle/\phi\rangle$ be an SNR. An* **SNR term** *is a ground term of the form $\phi(\sigma'_1, .., \sigma'_n)$ where σ'_1 is a ground term that represents σ_1, ..., and σ'_n is a ground term that represents σ_n.*

Example 3.1 *Consider the SNR given at the top left of Example 1.1. This can be represented by the following SNR term.*

```
businessreport(source(Reuters), action(NewBidMade),
      bidvalue($35Billion), buyer(Shell), target(company(Texaco), capital($25Billion))
```

In the next section, we show how we reason with SNR terms by using them as input for logical reasoning.

4 Basic definitions for the object-level and meta-level

For this paper, we assume the basic notions of syntax and semantics of classical logic. We will present both an object-level language and a meta-level language based on classical logic.

We assume a set of object-level atoms that are composed in the usual way (the usual classical logic definition of atoms) from a set of object-level predicate symbols, a set of object-level variables, a set of object-level constant symbols, and a set of object-level function symbols. The only object-level formulae we will consider are object-level atoms. We use object-level atoms, which we will refer to as **facts**, to represent and reason with information about the application domain. These may or may not be ground. We will not use object-level atoms for any other task.

Example 4.1 *For the mergers and acquisitions domain, facts may include the following ground atoms.*

$$\begin{gathered}\texttt{forSale}(\texttt{BP})\\ \texttt{bought}(\texttt{BP},\texttt{AnvilOilCorp})\\ \texttt{lent}(\texttt{AlphaBank},\texttt{NewSoftCorp},\texttt{\$100Million})\end{gathered}$$

We also assume a set of meta-level terms that are composed in the usual way (using the usual classical logic definition of terms) from a a set of meta-level variables, a set of meta-level constant symbols, and a set of meta-level function symbols. We assume that the set of meta-level variables includes the object-level variables and the set of meta-level constant symbols includes the set of object-level constant symbols. We also assume that the meta-level functions symbols includes the set of object-level function symbols and the set of object-level predicate symbols. This means that the set of meta-level terms includes object-level atoms. In this way meta-level atoms can have object-level atoms as arguments. We assume a set of meta-level atoms that are composed in the usual way from a set of meta-level predicate symbols and the set of meta-level terms. In other words, if p is a meta-level predicate symbol, and t_1,.., t_n are meta-level terms, then $p(t_1, .., t_n)$ is a meta-level atom. The meta-level predicate symbols we consider in this paper for reasoning with events in SNRs are summarized in Table 1.

In this paper, we will use strings beginning with a lower case letter to denote object-level predicate symbols. For example, $\texttt{happy}(\texttt{Tony}, \texttt{on}(\texttt{holiday}))$ is an object-level atom. And we will use strings of upper case letters to denote meta-level predicate symbols. For example, $\texttt{HOLDS}(\texttt{happy}(\texttt{Tony}, \texttt{on}(\texttt{holiday})))$ is a meta-level atom.

From a set of meta-level atoms, we form meta-level formulae using the usual logical symbols of first-order classical logic (i.e. $\{\vee, \wedge, \leftarrow, \neg\}$) and the usual inductive definitions. As a notational convenience, we have used $\leftarrow$ instead of the usual $\rightarrow$ symbol and so we represent $\alpha \rightarrow \beta$ by $\beta \leftarrow \alpha$. We restrict our first-order formulae to those with all variables being quantified outermost by universal quantification. So all meta-level formulae are classical first-order formulae, but not vice-versa. We will restrict all implicational formulae by the form $\forall X_1, .., X_n\ \beta \leftarrow \alpha$ where all variables in β and α are free and they are all bound by the quantification outermost (i.e. by $\forall X_1, .., X_n$).

We will undertake reasoning with the meta-level formulae using the standard notion of generalized modus ponens. For this we require the subsidiary notion of grounding.

Definition 4.1 *A* **grounding** *is an equality predicate where the first argument is a variable and the second argument is a ground term. A* **grounding set** *is a set of groundings which can be substituted into an unground term to give a grounded term. Let α be a formula and let Φ be a grounding set.* $\mathsf{Ground}(\alpha, \Phi)$ *gives the results of substituting each variable X in α with term t where the grounding $X = t$ is in Φ.*

Example 4.2 *Let* $\texttt{a}(\texttt{b}(\texttt{X}), \texttt{c}(\texttt{Y}), \texttt{d}(\texttt{Z}))$ *be a formula where* $\texttt{X}$, $\texttt{Y}$ *and* $\texttt{Z}$ *are variables. Let the grounding set*

Φ *be* $\{\mathtt{Y} = \mathtt{john}, \mathtt{Z} = \mathtt{betty}\}$.

$$\mathsf{Ground}(\mathtt{a}(\mathtt{b}(\mathtt{X}), \mathtt{c}(\mathtt{Y}), \mathtt{d}(\mathtt{Z})), \Phi) = \mathtt{a}(\mathtt{b}(\mathtt{X}), \mathtt{c}(\mathtt{john}), \mathtt{d}(\mathtt{betty}))$$

If α is a formula and it contains at least one variable symbol, then α is an unground formula, otherwise α is a ground formula.

Definition 4.2 *The following rule of inference is* **generalized modus ponens**. *For literals α_i, α'_i, and β, where there is a grounding set Φ such that* $\mathsf{Ground}(\alpha'_i, \Phi) = \mathsf{Ground}(\alpha_i, \Phi)$ *for all i:*

$$\frac{\alpha'_1, .., \alpha'_n, \qquad \forall X_1, .., X_n\ \beta \leftarrow \alpha_1 \wedge .. \wedge \alpha_n}{\mathsf{Ground}(\beta, \Phi)}$$

There are a number of examples of using generalized modus ponens in the following sections.

Since the reasoning discussed in this paper is effectively limited to the meta-level, we have obviated a number of the common problems of reasoning with meta-languages [BK82, Fis96].

5 Inferences about events from input

In this paper, we assume that each SNR refers to only one event. So for the information in each report we use the same event number (an identifier for the event). In the examples in this paper, event numbers come from the set $\{\mathtt{e}_1, \mathtt{e}_2, \mathtt{e}_3, \mathtt{e}_4, ...\}$. Each event number is treated as a meta-level constant symbol. Since more than one SNR term may refer to the same event, we will see later how we can identify equivalences between event numbers and thereby infer that two or more SNRs refer to the same event. An SNR input set is the input to the event-based merging, and it is defined next.

Definition 5.1 *An* **input atom** *is binary meta-level atom with predicate symbol* `INPUT`. *A ground input atom is of the form* $\mathtt{INPUT}(\mathtt{E}, \mathtt{S})$ *where* `E` *is the event number for the SNR term* `S`. *An* **SNR input set**, *denoted Λ, is a set of input atoms.*

Example 5.1 *The top left SNR given in Example 1.1 is represented by the following input atom.*

$$\begin{aligned}\mathtt{INPUT}(\mathtt{e}_8, \mathtt{businessreport}(&\mathtt{source}(\mathtt{Reuters}), \mathtt{action}(\mathtt{NewBidMade}),\\ &\mathtt{bidvalue}(\$35\mathtt{Billion}), \mathtt{buyer}(\mathtt{Shell}),\\ &\mathtt{target}(\mathtt{company}(\mathtt{Texaco}), \mathtt{capital}(\$25\mathtt{Billion}))))\end{aligned}$$

Example 5.2 *We use the following input atoms for our running example.*

$$\mathtt{INPUT}(\mathtt{e}_1, \mathtt{report}(\mathtt{buyer}(\mathtt{Walmart}), \mathtt{target}(\mathtt{Asda}), \mathtt{act}(\mathtt{NewBidMade}), \mathtt{value}(\$9\mathtt{Billion})))$$

$$\mathtt{INPUT}(\mathtt{e}_2, \mathtt{report}(\mathtt{buyer}(\mathtt{Walmart}), \mathtt{target}(\mathtt{Asda}), \mathtt{act}(\mathtt{BoardAcceptsOffer}), \mathtt{body}(\mathtt{board}))))$$

From each input atom, we will extract information about the event being described by the SNR term. This raises the need to characterize events in terms of atoms that can be derived from each input atom. These derived atoms are called **event atoms**. There are two types, action atoms and info atoms, defined next.

For each application domain, we also assume a set of actions. For the mergers and acquisitions domain, we can use the actions given in Figure 1. Each action is treated as an object-level constant and therefore it is also a meta-level constant symbol.

Definition 5.2 *An* **action atom** *is binary meta-level atom with predicate symbol* ACTION*. A ground action atom is of the form* $\mathtt{ACTION(E,A)}$ *where* E *is the event number for an event with the action* A*.*

Definition 5.3 *An* **info atom** *is binary meta-level atom with the predicate symbol* INFO*. A ground info atom is of the form* $\mathtt{INFO(E,F)}$ *where* E *is the event number for an event with the associated fact* F *where* F *is an object-level atom.*

Example 5.3 *The following event atoms capture the event number* e1 *when Walmart has made a bid for Asda as given in Example 5.2.*

$$\begin{array}{l}\mathtt{ACTION(e_1, NewBidMade)}\\ \mathtt{INFO(e_1, bid(Walmart, Asda))}\\ \mathtt{INFO(e_1, bidvalue(\$9Billion))}\end{array}$$

Example 5.4 *The following event atoms capture the event number* e2 *when Asda's board has accepted Walmart's bid as given in Example 5.2.*

$$\begin{array}{l}\mathtt{ACTION(e_2, BoardAcceptsOffer)}\\ \mathtt{INFO(e_2, bid(Walmart, Asda))}\\ \mathtt{INFO(e_2, body(board))}\end{array}$$

Example 5.5 *The following event atoms capture the event number* e5 *when Walmart completes the takeover of Asda.*

$$\begin{array}{l}\mathtt{ACTION(e_5, TakeoverFinalized)}\\ \mathtt{INFO(e_5, takeover(Walmart, Asda, \$9Billion))}\end{array}$$

To extract information about events from the input atoms, we use first-order formulae called access rules. Each access rule has a condition that is an unground input atom and a consequent that is an unground event atom. So given a ground input atom and an access rule, we apply generalized modus ponens to generate ground event atoms.

Definition 5.4 *An* **access rule** *is a meta-level first-order formula of the form* $\forall X_1, ..., X_k; \beta \leftarrow \alpha$ *where* α *is an unground input atom and* β *is an unground event atom.*

Example 5.6 *The following are access rules.*

$$\begin{array}{l}\forall \mathtt{E, B, C, A; ACTION(E, A)} \leftarrow\\ \qquad\qquad \mathtt{INPUT(E, report(buyer(B), target(C), act(A)))}\\[1em] \forall \mathtt{E, B, C, A; INFO(E, bid(B, C))} \leftarrow\\ \qquad\qquad \mathtt{INPUT(E, report(buyer(B), target(C), act(A)))}\\[1em] \forall \mathtt{E, B, C, A, V; INFO(E, bidvalue(V))} \leftarrow\\ \qquad\qquad \mathtt{INPUT(E, report(buyer(B), target(C), act(A), value(V)))}\\[1em] \forall \mathtt{E, B, C, A, D; INFO(E, body(D))} \leftarrow\\ \qquad\qquad \mathtt{INPUT(E, report(buyer(B), target(C), act(A), body(D)))}\end{array}$$

Definition 5.5 *Let* Δ *be a set of access rules and let* Λ *be an SNR input set. For this,* $\mathsf{Access}(\Delta, \Lambda)$ *is the set of event atoms obtained by exhaustively applying the input atoms* Λ *to the access rules in* Δ *using generalized modus ponens as defined below.*

$$\mathsf{Access}(\Delta, \Lambda) = \{\mathsf{Ground}(\beta, \Phi) \mid \forall X_1, .., X_k; \beta \leftarrow \alpha \in \Delta \textit{ and } \mathsf{Ground}(\alpha, \Phi) \in \Lambda\}$$

Example 5.7 *Assuming the input atoms in Example 5.2 are in* Λ*, and the access rules given in Example 5.6 are in* Δ*, then we have the event atoms presented in Examples 5.3 and 5.4 in* $\mathsf{Access}(\Delta, \Lambda)$.

Access rules can be viewed as unpacking the required information about events from the nested information in the SNRs. If the SNRs are highly nested and/or structurally diverse, then a slightly more complex set-up for access rules drawing on subsidiary axioms would be appropriate.

6 Transition axioms

Transition axioms provide an abstract definition of the possible events that can take place in a stereotypical narrative, along with the ordering of these events. A set of transition axioms is domain specific knowledge. It is defined in terms of three types of axiom: (1) Initiating axioms; (2) Terminating axioms; and (3) Event ordering axioms. In this section, we define each of these types of axiom and give examples.

States are begun and ended by events. The `INITIATES` predicate allows us to define which states are begun by which events.

Definition 6.1 *An* **initiating axiom** *is of the following form where* `F` *is a fact,* `E` *is an event number,* β *is a conjunction of event atoms and possibly other meta-level atoms (such as comparison relations).*

$$\forall \mathtt{E}, \mathtt{X}_1, .., \mathtt{X}_n; \mathtt{INITIATES}(\mathtt{E}, \mathtt{F}) \leftarrow \beta$$

We refer to a fact `F` *being in a state that has been initiated by event* `E`.

Example 6.1 *The state during which a bid is made by a buyer* `B` *for a target* `C`*, and in which the buyer is awaiting a response, is initiated by the action of a new bid being made.*

$$\forall \mathtt{E}, \mathtt{B}, \mathtt{C}; \mathtt{INITIATES}(\mathtt{E}, \mathtt{boardConsidersBid}(\mathtt{B}, \mathtt{C})) \leftarrow \mathtt{ACTION}(\mathtt{E}, \mathtt{NewBidMade}) \wedge \mathtt{INFO}(\mathtt{E}, \mathtt{bid}(\mathtt{B}, \mathtt{C}))$$

Example 6.2 *The state during which a bid value exists for a bid by a buyer* `B` *for a target* `T` *is initiated by an event involving the action of a new bid being made for a value* `V`.

$$\begin{aligned} &\forall \mathtt{E}, \mathtt{B}, \mathtt{C}, \mathtt{V}; \mathtt{INITIATES}(\mathtt{E}, \mathtt{bidValue}(\mathtt{B}, \mathtt{C}, \mathtt{V})) \leftarrow \\ &\qquad \mathtt{ACTION}(\mathtt{E}, \mathtt{NewBidMade}) \wedge \mathtt{INFO}(\mathtt{E}, \mathtt{bid}(\mathtt{B}, \mathtt{C})) \wedge \mathtt{INFO}(\mathtt{E}, \mathtt{bidvalue}(\mathtt{V})) \end{aligned}$$

Example 6.3 *The state during which a board for a target* `C` *has accepted a bid and in which the shareholders are considering the bid, is initiated by the action of the bid being accepted by the board.*

$$\begin{aligned} &\forall \mathtt{E}, \mathtt{B}, \mathtt{C}; \mathtt{INITIATES}(\mathtt{E}, \mathtt{shareholdersConsiderBid}(\mathtt{B}, \mathtt{C})) \leftarrow \\ &\qquad \mathtt{ACTION}(\mathtt{E}, \mathtt{BoardAcceptsOffer}) \wedge \mathtt{INFO}(\mathtt{E}, \mathtt{bid}(\mathtt{B}, \mathtt{C})) \end{aligned}$$

The `TERMINATES` predicate allows us to define which states are ended by which events.

Definition 6.2 *An* **terminating axiom** *is either of the following two forms where* `F` *is a fact,* `E` *is an event number,* β *is a conjunction of event atoms and possibly other meta-level atoms (such as comparison relations).*

$$\forall \mathtt{E}, \mathtt{X}_1, .., \mathtt{X}_n; \mathtt{TERMINATES}(\mathtt{E}, \mathtt{F}) \leftarrow \beta$$

$$\forall \mathtt{E}, \mathtt{X}_1, .., \mathtt{X}_n; \neg\mathtt{TERMINATES}(\mathtt{E}, \mathtt{F}) \leftarrow \beta$$

Example 6.4 *The state during which a company* $\mathtt{B}$ *is the position of being a bidder for a target* $\mathtt{C}$ *ends when the bid has been accepted.*

$$\begin{aligned}&\forall \mathtt{E}, \mathtt{B}, \mathtt{C}; \mathtt{TERMINATES}(\mathtt{E}, \mathtt{boardConsidersBid}(\mathtt{B}, \mathtt{C})) \leftarrow \\ &\qquad \mathtt{ACTION}(\mathtt{E}, \mathtt{BoardAcceptsOffer}) \wedge \mathtt{INFO}(\mathtt{E}, \mathtt{bid}(\mathtt{B}, \mathtt{C})) \wedge \mathtt{INFO}(\mathtt{E}, \mathtt{body}(\mathtt{board}))\end{aligned}$$

We use the ordering over actions, for example in Figure 1, to specify an ordering over events. We represent the ordering over events by the $\preceq$ pre-ordering relation. Intuitively, $\mathtt{E}_1 \preceq \mathtt{E}_2$ means that event $\mathtt{E}_1$ occurs before, or at the same time as, $\mathtt{E}_2$. The pre-ordering relation is defined for a domain using a set of event ordering axioms defined as follows.

Definition 6.3 *An* **event ordering axiom** *is of the following form where* $\mathtt{E}_1$ *and* $\mathtt{E}_2$ *are event numbers,* $\mathtt{A}_1$ *and* $\mathtt{A}_2$ *are actions, and* $\mathtt{F}_1^1$*,..,*$\mathtt{F}_1^n$*,*$\mathtt{F}_2^1$*,..,*$\mathtt{F}_2^m$ *are facts, and* $\mathtt{X}_1, .., \mathtt{X}_n$ *are the free variables in them.*

$$\begin{aligned}&\forall \mathtt{E}_1, \mathtt{E}_2 \mathtt{X}_1, .., \mathtt{X}_n; \mathtt{E}_1 \preceq \mathtt{E}_2 \leftarrow \\ &\qquad \mathtt{ACTION}(\mathtt{E}_1, \mathtt{A}_1) \wedge \mathtt{INFO}(\mathtt{E}_1, \mathtt{F}_1^1) \wedge .. \wedge \mathtt{INFO}(\mathtt{E}_1, \mathtt{F}_1^n) \\ &\qquad \wedge \mathtt{ACTION}(\mathtt{E}_2, \mathtt{A}_2) \wedge \mathtt{INFO}(\mathtt{E}_2, \mathtt{F}_2^1) \wedge .. \wedge \mathtt{INFO}(\mathtt{E}_2, \mathtt{F}_2^m)\end{aligned}$$

Also let $\mathtt{E}_1 \simeq \mathtt{E}_2$ *iff* $E_1 \preceq E_2$ *and* $E_2 \preceq E_1$.

In effect, the $\mathtt{INFO}$ atoms ensure that the events $\mathtt{E}_1$ and $\mathtt{E}_2$ are on the same subject, and that the $\mathtt{ACTION}$ atoms ensure the ordering is consistent with the ordering over actions. If for some event numbers $\mathtt{E}_1$ and $\mathtt{E}_2$, we have $\mathtt{E}_1 \simeq \mathtt{E}_2$, then $\mathtt{E}_1$ and $\mathtt{E}_2$ refer to the same event.

Example 6.5 *Event number* $\mathtt{E}_1$ *is before event number* $\mathtt{E}_2$ *if* $\mathtt{E}_1$ *and* $\mathtt{E}_2$ *involve the same buyer and target and* $\mathtt{E}_1$ *involves a bid being made and* $\mathtt{E}_2$ *involves a bid being accepted.*

$$\begin{aligned}&\forall \mathtt{E}_1, \mathtt{E}_2, \mathtt{B}, \mathtt{C}; \mathtt{E}_1 \preceq \mathtt{E}_2 \leftarrow \\ &\qquad \mathtt{ACTION}(\mathtt{E}_1, \mathtt{NewBidMade}) \wedge \mathtt{INFO}(\mathtt{E}_1, \mathtt{bid}(\mathtt{B}, \mathtt{C})) \\ &\qquad \wedge \mathtt{ACTION}(\mathtt{E}_2, \mathtt{BoardAcceptsOffer}) \wedge \mathtt{INFO}(\mathtt{E}_1, \mathtt{bid}(\mathtt{B}, \mathtt{C}))\end{aligned}$$

Example 6.6 *Continuing Examples 5.3 and 5.4, and Example 6.5, we get* $\mathtt{e}_1 \preceq \mathtt{e}_2$

Example 6.7 *Event number* $\mathtt{E}_1$ *and event number* $\mathtt{E}_2$ *are equal in the ordering if* $\mathtt{E}_1$ *and* $\mathtt{E}_2$ *involve the same action, buyer, target, and value, in a bid being made*

$$\begin{aligned}&\forall \mathtt{E}_1, \mathtt{E}_2, \mathtt{A}, \mathtt{B}, \mathtt{C}, \mathtt{V}; \mathtt{E}_1 \preceq \mathtt{E}_2 \leftarrow \\ &\qquad \mathtt{ACTION}(\mathtt{E}_1, \mathtt{A})) \wedge \mathtt{INFO}(\mathtt{E}_1, \mathtt{bid}(\mathtt{B}, \mathtt{C})) \wedge \mathtt{INFO}(\mathtt{E}_1, \mathtt{bidvalue}(\mathtt{V})) \\ &\qquad \wedge \mathtt{ACTION}(\mathtt{E}_2, \mathtt{A})) \wedge \mathtt{INFO}(\mathtt{E}_2, \mathtt{bid}(\mathtt{B}, \mathtt{C})) \wedge \mathtt{INFO}(\mathtt{E}_2, \mathtt{bidvalue}(\mathtt{V}))\end{aligned}$$

Example 6.8 *Given the following set of event atoms together with Examples 5.3 and 6.7, we get* $\mathtt{e}_1 \preceq \mathtt{e}_9$ *and* $\mathtt{e}_9 \preceq \mathtt{e}_1$ *and so* $\mathtt{e}_1 \simeq \mathtt{e}_9$ *holds. As a result, we can regard the event numbers* $\mathtt{e}_1$ *and* $\mathtt{e}_9$ *as referring to the same event.*

$$\begin{aligned}&\mathtt{ACTION}(\mathtt{e}_9, \mathtt{NewBidMade})) \\ &\mathtt{INFO}(\mathtt{e}_9, \mathtt{bid}(\mathtt{Walmart}, \mathtt{Asda})) \\ &\mathtt{INFO}(\mathtt{e}_9, \mathtt{bidvalue}(\$9\mathtt{Billion})) \\ &\mathtt{INFO}(\mathtt{e}_9, \mathtt{payment}(\mathtt{cash}))\end{aligned}$$

So for each pair of actions, $\mathtt{A}_1$ and $\mathtt{A}_2$ such that $\mathtt{A}_2$ is an action that can occur immediately after $\mathtt{A}_1$, we require one or more event ordering axioms. In this way, we can start with a domain analysis where we identify a set of actions and a partial ordering over those actions, and then specify event ordering axioms consistent with this poset of actions.

We can now define a transition model in terms of the initiating axioms, terminating axioms, and event ordering axioms.

Definition 6.4 *A* **transition model** *is a set of formulae of the following types: (1) Initiating axioms (Definition 6.1); (2) Terminating axioms (Definition 6.2); and (3) Event ordering axioms (Definition 6.3).*

Once a set of transition axioms is in place we can use it to reason with a set of event atoms using our version of the event calculus defined in the next section.

7 Event calculus

In order to be able to derive the state model for an SNR input set, using a set of event atoms and a transition model, we need some way to define the relations between events and states. For this, we adopt the event calculus proposed by Kowalski and Sergot [KS86]. In this calculus, meta-level predicates are used to identify the states (or relationships) which hold during time periods by determining the events which initiate and terminate those states. In other words, a time period is a duration delineated by an event occuring at the start of the time period and an event occuring at the end of the time period. We do not need to use an explicit clock or calandar for this. The events are sufficient for the calibration. To illustrate, in information about the US government, events about presidents can be used to delineate useful time periods. For example, the first term of Bill Clinton in office is a time period that has been started by the event of him being elected for a first term and ended by him being elected for a second term.

In our version of the event calculus, the HOLDSAFTER predicate is used to identify facts holding after events occuring. In the following definition, the first axiom provides the base case for initiating facts holding in states. The second axiom propagates facts forward into the next state if they are not terminated by the event starting that next state.

Definition 7.1 *The* **holdsafter axioms** *are defined as follows.*

(1) $\forall \mathtt{E}, \mathtt{F};\ \mathtt{HOLDSAFTER}(\mathtt{E}, \mathtt{F}) \leftarrow \mathtt{INITIATES}(\mathtt{E}, \mathtt{F})$

(2) $\forall \mathtt{E}, \mathtt{E}', \mathtt{F};\ \mathtt{HOLDSAFTER}(\mathtt{E}', \mathtt{F}) \leftarrow \mathtt{HOLDSAFTER}(\mathtt{E}, \mathtt{F}) \land \mathtt{E} \preceq \mathtt{E}' \land \neg\mathtt{TERMINATES}(\mathtt{E}', \mathtt{F})$

Example 7.1 *Consider the event atoms in Examples 5.3 and 5.4 and the initiating axioms in Examples 6.1 and 6.2 and the terminating axioms in Example 6.4. Using the event calculus, we can infer the following.*

$$\begin{array}{l}\mathtt{HOLDSAFTER}(\mathtt{e}_1, \mathtt{boardConsidersBid}(\mathtt{Walmart}, \mathtt{Asda}))\\ \mathtt{HOLDSAFTER}(\mathtt{e}_2, \mathtt{shareholdersConsiderBid}(\mathtt{Walmart}, \mathtt{Asda}))\\ \mathtt{HOLDSAFTER}(\mathtt{e}_2, \mathtt{bidValue}(\mathtt{Walmart}, \mathtt{Asda}, \$9\mathtt{Billion}))\end{array}$$

We can now pull together the transition model and the event calculus to give the following definition of an event reasoning system.

Definition 7.2 *An* **event reasoning system** *is the union of a transition model (Definition 6.4) and the holdsafter axioms (Definition 7.1).*

Predicate	Informal definition
INPUT(E,S)	Event E is described by the SNR term S.
ACTION(E,A)	Event E involves action A.
INFO(E,F)	Event E involves fact F.
INITIATES(E,F)	Event E initiates fact F holding.
TERMINATES(E,F)	Event E terminates fact F holding.
$E_1 \preceq E_2$	Event E_1 occurs before or at the same time as event E_2.
HOLDSAFTER(E,F)	Fact F holds after event E.

Table 1: Summary of meta-predicates required for reasoning with events in structured news reports.

So an event reasoning system is defined for a particular application. It is domain-specific. The holdsafter axioms are common to any event reasoning system, but the transition model is defined to handle the kinds of events arising in the application. We summarize all the meta-level predicates introduced in this paper for an event reasoning system in Table 1.

Example 7.2 *Assume we have exactly the following event atoms.*

$\mathtt{ACTION}(\mathtt{e}_1, \mathtt{NewBidMade})$ $\mathtt{ACTION}(\mathtt{e}_2, \mathtt{BoardAcceptsOffer})$
$\mathtt{INFO}(\mathtt{e}_1, \mathtt{bid}(\mathtt{Walmart}, \mathtt{Asda}))$ $\mathtt{INFO}(\mathtt{e}_2, \mathtt{bid}(\mathtt{Walmart}, \mathtt{Asda}))$
$\mathtt{INFO}(\mathtt{e}_1, \mathtt{value}(\$\mathtt{9Billion}))$ $\mathtt{INFO}(\mathtt{e}_2, \mathtt{body}(\mathtt{board}))$

$\mathtt{ACTION}(\mathtt{e}_3, \mathtt{ShareholdersAcceptOffer})$ $\mathtt{ACTION}(\mathtt{e}_4, \mathtt{ReferralDeemedUnnecessary})$
$\mathtt{INFO}(\mathtt{e}_3, \mathtt{bid}(\mathtt{Walmart}, \mathtt{Asda}))$ $\mathtt{INFO}(\mathtt{e}_4, \mathtt{bid}(\mathtt{Walmart}, \mathtt{Asda}))$
$\mathtt{INFO}(\mathtt{e}_3, \mathtt{location}(\mathtt{Walmart}, \mathtt{USA}))$ $\mathtt{INFO}(\mathtt{e}_4, \mathtt{body}(\mathtt{EUAuthorities}))$

Now assuming an appropriate event reasoning system, extending the axioms given in Section 6, we can obtain the following using generalized modus ponens. First, we give the result of applying axiom (1) of the holdsafter definition.

$\mathtt{HOLDSAFTER}(\mathtt{e}_1, \mathtt{boardConsidersBid}(\mathtt{Walmart}, \mathtt{Asda})))$
$\mathtt{HOLDSAFTER}(\mathtt{e}_1, \mathtt{bidValue}(\mathtt{Asda}, \mathtt{Walmart}, \$\mathtt{9Billion}))$

$\mathtt{HOLDSAFTER}(\mathtt{e}_2, \mathtt{shareholdersConsiderOffer}(\mathtt{Walmart}, \mathtt{Asda}))$

$\mathtt{HOLDSAFTER}(\mathtt{e}_3, \mathtt{authoritiesConsiderBid}(\mathtt{Walmart}, \mathtt{Asda}))$
$\mathtt{HOLDSAFTER}(\mathtt{e}_3, \mathtt{bidderLocation}(\mathtt{Walmart}, \mathtt{USA}))$

$\mathtt{HOLDSAFTER}(\mathtt{e}_4, \mathtt{bidCleared}(\mathtt{Walmart}, \mathtt{Asda}))$
$\mathtt{HOLDSAFTER}(\mathtt{e}_4, \mathtt{bidClearedBy}(\mathtt{Walmart}, \mathtt{Asda}, \mathtt{EUAuthorities}))$

Now we give the result of applying axiom (2) of the holdsafter definition. These are the first two atoms below.

$\mathtt{HOLDSAFTER}(\mathtt{e}_4, \mathtt{bidValue}(\mathtt{Walmart}, \mathtt{Asda}, \$\mathtt{9Billion})$
$\mathtt{HOLDSAFTER}(\mathtt{e}_4, \mathtt{bidderLocation}(\mathtt{Walmart}, \mathtt{USA})$
$\mathtt{HOLDSAFTER}(\mathtt{e}_4, \mathtt{bidCleared}(\mathtt{Walmart}, \mathtt{Asda})$
$\mathtt{HOLDSAFTER}(\mathtt{e}_4, \mathtt{bidClearedBy}(\mathtt{Walmart}, \mathtt{Asda}, \mathtt{EUAuthorities}))$

The above four atoms give the most up-to-date information, according to the event atoms used.

The version of event calculus we have presented in this section uses classical logic. This provides a clear and straightforward solution for reasoning with information about events in news reports as explained in the next section.

Since we have focussed on only one aspect of event-based merging (getting the most up-to-date information that holds after the most reecent event according to the assumptions given in Section 2), we have not harnessed a number of aspects of event calculus as originally proposed and subsequently developed. In particular, we have not used timepoints to reason about the relationships between events and explicit representation of time (as given by clocks and/or calendars), and so we have not included definitions for the holdsat predicate $\mathtt{HOLDSAT}(\mathtt{F}, \mathtt{T})$ where $\mathtt{F}$ is a fact and $\mathtt{T}$ is an explicit timepoint. We leave the use of explicit time to a future paper.

8 Output of event-based merging

The output of event-based merging is an SNR that aggregates information found in the SNR input set. The aggregation is specified by an aggregation rule. Each aggregation rule states the preconditions for merging, in terms of the information that is required, together with the composition and structure of the merged SNR term.

Definition 8.1 *An* **output atom**, *denoted* $\mathtt{OUTPUT}(\mathtt{E}, \mathtt{F})$, *is a binary meta-level predicate where* $\mathtt{F}$ *is a fact and* $\mathtt{E}$ *is an event number.*

Definition 8.2 *An* **aggregation rule** *is a meta-level formula of the following form where if there is a grounding set* Φ *s.t.* $\mathsf{Ground}(\beta_1, \Phi)$,...,$\mathsf{Ground}(\beta_m, \Phi)$ *are ground holdsafter atoms, then* $\mathsf{Ground}(\mathtt{OUTPUT}(\mathtt{E}, \mathtt{F}), \Phi)$ *is a ground output atom.*

$$\forall X_1, ..., X_k; \mathtt{OUTPUT}(\mathtt{E}, \mathtt{F}) \leftarrow \beta_1 \wedge ... \wedge \beta_m$$

Example 8.1 *The following is an aggregation rule.*

$$\begin{array}{l}
\forall \mathtt{E}, \mathtt{B}, \mathtt{C}, \mathtt{L}, \mathtt{V};\\
\mathtt{OUTPUT}(\mathtt{E}, \mathtt{report}(\mathtt{buyer}(\mathtt{B}), \mathtt{target}(\mathtt{C}), \mathtt{status}(\mathtt{bidCleared}), \mathtt{bidderLocation}(\mathtt{L}), \mathtt{value}(\mathtt{V})))\\
\qquad\qquad \leftarrow \mathtt{HOLDSAFTER}(\mathtt{E}, \mathtt{bidCleared}(\mathtt{B}, \mathtt{C}))\\
\qquad\qquad\qquad \wedge \mathtt{HOLDSAFTER}(\mathtt{E}, \mathtt{bidderLocation}(\mathtt{B}, \mathtt{L}))\\
\qquad\qquad\qquad \wedge \mathtt{HOLDSAFTER}(\mathtt{E}, \mathtt{bidValue}(\mathtt{B}, \mathtt{C}, \mathtt{V}))
\end{array}$$

We now define event-based merging that takes an aggregation rule and a set of input atoms together with an event reasoning system and a set of access rules and returns an SNR term according to the specification given by the aggregation rule.

Definition 8.3 *From a set of input atoms* Λ, **event-based merging** *returns a merged report, represented by an SNR term* $\mathtt{S}$, *as follows, where* $\forall X_1, .., X_k$ $\mathtt{Output}(\mathtt{E}, \mathtt{F}) \leftarrow \beta_1 \wedge .. \wedge \beta_m$ *is an aggregation rule,* Π *is an event reasoning system, and* Γ *is a set of access rules:*

$$\begin{array}{l}
\textit{If the following hold, for some } \Phi,\\
\qquad \mathsf{Access}(\Gamma, \Lambda) \cup \Pi \vdash \mathsf{Ground}(\beta_1, \Phi)\\
\qquad \textit{and}\\
\qquad\qquad \vdots\\
\qquad \textit{and}\\
\qquad \mathsf{Access}(\Gamma, \Lambda) \cup \Pi \vdash \mathsf{Ground}(\beta_m, \Phi)\\
\qquad \textit{and}\\
\qquad \mathsf{Ground}(\mathtt{OUTPUT}(\mathtt{E}, \mathtt{F}), \Phi) = \mathtt{OUTPUT}(\mathtt{E}, \mathtt{S})\\
\textit{then return } \mathtt{S}\\
\textit{otherwise return } \mathtt{NULL}.
\end{array}$$

Here, we assume that $\vdash$ *is the classical consequence relation.*

In the above definition, we see the F acts as a template for the merged report, and S is the merged report.

Example 8.2 *Continuing Example 7.1, with the aggregation rule in Example 8.1, we get the following event-based merging.*

$$\begin{aligned}&\texttt{OUTPUT}(\texttt{e}_4, \texttt{report}(\texttt{buyer}(\texttt{Walmart}), \texttt{target}(\texttt{Asda}), \texttt{status}(\texttt{bidCleared}),\\ &\qquad \texttt{bidderLocation}(\texttt{USA}), \texttt{value}(\$9\texttt{Billion})))\end{aligned}$$

This gives the following SNR, which provides an aggregation of the information presented in Example 7.2 which in turn was obtained from four input SNRs.

```
⟨report⟩
        ⟨buyer⟩ Walmart ⟨/buyer⟩
        ⟨target⟩ Asda ⟨/target⟩
        ⟨status⟩ bidCleared ⟨/status⟩
        ⟨bidderLocation⟩ USA ⟨/bidderLocation⟩
        ⟨value⟩ $9Billion ⟨/value⟩
⟨/report⟩
```

In the next section, we consider how event-based can be implemented in Prolog.

9 Implementing event-based merging

We have implemented event-based merging in Prolog. The logical form of the axioms in access rules, transition models, and for the holdsafter definition, are all based on a subset of classical logic that can be straightforwardly represented in Prolog: They are all universally quantified with quantifiers outermost, and they are all of the form of rules where the antecendent is a conjunction of zero or more literals and the consequent is a literal.

Since Prolog does not support classical negation, we have adapted the definitions given so far. The key difference between the definitions given so far and the Prolog version are in the implementation for the holdsafter axioms. For these, we have substituted classical negation with the Prolog negation (negation-as-failure). In the Prolog version, we do not have negated terminates predicates. But by using Prolog negation, we are applying the closed world assumption to the terminates clauses that have an unnegated terminates predicate.

Definition 9.1 *The Prolog version of the holdsafter axioms are defined as follows where* `not` *is the negation-as-failure operator.*

```
holdsafter(E, F) :− initiates(E, F)

holdsafter(E2, F) :− holdsafter(E1, F)), eventorder(E2, E1), not terminates(E2, F)
```

In addition to the holdsafter axioms, we require clauses in our Prolog version for the initiates, terminates, and event ordering axioms. We list some of these clauses in the Appendix. In addition, we have incorporated the following clauses into our Prolog version.

Definition 9.2 *The* `findlatest(E, L)` *predicate, which is defined below, finds the latest event* `E` *in the list of event numbers* `L`*. This requires the following subsidiary predicates* `last(E, L)` *and* `order(E1, E2)` *as follows. Also,* `\==` `(T1, T2)` *is a built-in predicate for Prolog that holds when terms* `T1` *and* `T2` *are not* *identical.*

```
findlast(E, [E|R]) :- last(E, R).
findlast(E, [F|R]) :- not last(F, R), findlast(E, R).
findlast(E, [E]).

last(E1, [E1|R]) :- last(E1, R).
last(E1, [E2|R]) :- \== (E1, E2), order(E1, E2), last(E1, R).
last(E, []).

order(E1, E2) :- eventorder(E1, E2).
order(E1, E2) :- eventorder(E1, E3), order(E3, E2).
```

Example 9.1 *Suppose we have four SNRs which are given as input, with event numbers* `e1`, `e2`, `e3`, *and* `e4`. *In the Prolog program in the Appendix, we have explicitly listed some event atoms obtain from these SNRs. Now, suppose we consider the following query, where* `E` *is a variable and* `[e1, e4, e2, e3]` *is an arbitrary list containing these event numbers.*

```
findlast(E, [e1, e4, e2, e3])
```

For this query, we obtain the grounding `e4` *for the variable* `E`. *Now we can ask the following query where* `R` *is a variable.*

```
output(e4, R)
```

For this query, we obtain the following two groundings for the variable `R` *(assuming just the clauses in the appendix). Either of these groundings can be taken as out merged SNR.*

```
report(buyer(walmart), target(asda), value(us9billion))

report(buyer(walmart, location(usa), capitalization(us23billion)),
                          target(asda), value(us9billion))
```

Now consider, the following query.

```
findlast(E, [e3, e1, e2])
```

For this query, we obtain the grounding `e3` *for the variable* `E`. *Now we can ask the following query where* `R` *is a variable.*

```
output(e3, R)
```

For this query, one of the groundings that we obtain for the variable `R` *is as follows.*

```
report(buyer(walmart), target(asda), status(antiTrustAgenciesConsiderBid))
```

In this way, we can have a merged report that summarises the state after each event, and hence shows a narrative.

Using our Prolog implementation, we undertook a case study with mergers and acquisitions reports. Starting with a partial ordering over event types (i.e. actions), we can define a transition model. Then, we encoded initiating axioms and terminating axioms for each action. In the case study, we did not encode further background knowledge. However, it is straightforward to add clauses to provide some simple equivalences between information in the input reports. For example, synonyms for event types, different names for the same entity, or equivalence between monetary values given in different currencies.

A key advantage of Prolog for implementing our formalization of event-based merging is that meta-level programming is straightforward in Prolog. So, for example, an interpreter for Prolog can be encoded within a Prolog program [SS94]. This means that event-based merging as given in Definition 9.1 can be encoded as meta-level interpreter. Furthermore, we can enhance the reasoning to allow, for example, finding the largest output merged, and so address the choice of groundings we see in Example 9.1.

10 Discussion

In our approach to merging SNRs, we draw on domain knowledge to help produce merged reports. Using domain knowledge together with logical deduction, we can reason about the information to be merged.

The novel contribution of this paper is to show how SNRs, or parts of SNRs, that describe events can be merged using an underlying event model based on the event calculus approach. We have chosen to use event calculus as opposed to a temporal logic because we want to harness the explicit information describing events that exists in many business reports rather then seek explicit timepoints or time interval information. Whilst many news reports do have time point and time interval information clearly represented, it is not always the case. So in the situation that the temporal information is absent, or in the case that two reports have the same timepoint, but conflicting information, we may wish to use the event information to resolve conflicts and identify the most recent information.

If we were to use explicit temporal information, there are numerous options for formalising temporal reasoning based on a temporal logic (see for example [GHR94]). But for formalising reasoning with events, there are only limited options. The most significant of these options is based on event calculus. Event calculus was originally proposed by Kowalski and Sergot [KS86]. Since then a number of variants of event calculus have been proposed. For a review see [Sha99, MS99]. These variants of event calculus have been presented in either classical logic and in Prolog.

In developments of the event calculus, there are a number of concepts formalised in richer ontologies for representing and reasoning with events based on properties of actions and of fluents (facts that are affected by particular actions). In particular, bringing explict time into the reasoning offers some interesting developments for event-based merging. This may include using time points (see for example [Sho88, KM97, Sha99, KMT00]) or time intervals (see for example [All84, AF94]).

Our logic-based approach significantly differs from other logic-based approaches for merging information (e.g. [BKMS92, Rev97, KP98, LS98, LS00, KLM02, Kon02, KP02]). In these approaches, aggregation is undertaken on a set of knowledgebases. Each knowledgebase is a set of logial formulae and each is regarded as a source of information. These approaches adopt syntactic or semantic approaches to aggregate information based on strategies such as maximising the number of sources that can be accommodated in the merged information or taking information that is agreed on by the majority of sources. In none of these approaches is any consideration of the underlying ontology of the information in the sources, and in particular none consider identifying and reasoning about events arising in the information to be merged. As a result this paper offers a more intelligent approach to merging in the sense that it takes account of how the validity of facts changes with the occurrence of events. It provides a framework for representing and reasoning with events that can occur in focused domains such as mergers and acquisitions in business, and for merging facts based on those events.

There is further work that needs to be done on event-based merging including (1) introduce the option of representing explicit timepoints and time intervals in an SNR for an event that has occurred; (2) introduce the option of an explicit timepoint in an SNR for when it was filed or published; (3) relax the condition that actions are partially-ordered, and to then consider iterative actions (for example, a bid value may be revised a number of times to give a new bid value); and (4) allow a set of reports to be on mutliple subjects, and to support identification of different narratives at different levels of granularity. To expand on this last point, a narrative is a state model concerning a particular subject. The definition of a subject depends on the application domain. In the case of mergers and acquisitions, a subject could be defined as a combination of buyer and target companies, and so in each state in the narrative, the facts concern the same buyer(s) and the same target(s). A narrative can contain a subnarrative which is a subset of the states with a more focussed subject. For example, a narrative on the subject of a takeovers of companies on the London Stock Exchange includes a subnarrative of takeovers of pharmaceutical companies on the London Stock Exchange. A logic-based solution is a promising way forward on these issues.

Acknowledgements

The authors are grateful to the anonymous referees for a number of suggestions for improvements to this paper.

References

[AF94] J Allen and G Ferguson. Actions and events in interval temporal logic. *Journal of Logic and Computation*, 4:531–579, 1994.

[All84] J Allen. Towards a general theory of action and time. *Artificial Intelligence*, 23:123–154, 1984.

[BK82] K Bowen and R Kowalski. Amalgamating language and metalanguage in logic programming. In K Clark and S Tarnlund, editors, *Logic Programming*, volume 16, pages 153–172. Academic Press, 1982.

[BKMS92] C Baral, S Kraus, J Minker, and V Subrahmanian. Combining knowledgebases consisting of first-order theories. *Computational Intelligence*, 8:45–71, 1992.

[CL96] J Cowie and W Lehnert. Information extraction. *Communications of the ACM*, 39:81–91, 1996.

[Fis96] M Fisher. Languages, meta-languages, and metatem: A discussion paper. *Journal of the Interest Group for Pure and Applied Logic*, 4(2):255–272, 1996.

[GHR94] D Gabbay, I Hodkinson, and M Reynolds. *Temporal Logic: Mathematical Foundations and Computational Aspects*. Oxford University Press, 1994.

[HS03] A Hunter and R Summerton. Propositional fusion rules. In *Symbolic and Quantitative Approaches to Reasoning with Uncertainty*, volume 2711 of *Lecture Notes in Computer Science*, pages 502–514. Springer, 2003.

[HS04] A Hunter and R Summerton. Fusion rules for context-dependent aggregation of structured news reports. *Journal of Applied Non-classical Logic*, 14(3):329–366, 2004.

[Hun02a] A Hunter. Logical fusion rules for merging structured news reports. *Data and Knowledge Engineering*, 42:23–56, 2002.

[Hun02b] A Hunter. Merging structured text using temporal knowledge. *Data and Knowledge Engineering*, 41:29–66, 2002.

[KM97] A Kakas and R Miller. A simple declarative language for describing narratives with actions. *Journal of Logic Programming*, 31:157–200, 1997.

[KMT00] A Kakas, R Miller, and F Toni. E-res - a system for reasoning about actions, events and observations. In *Proceedings of the 8th International Symposium on Nonmonotonic Reasoning (NMR2000)*. Computing Research Repository (CoRR), 2000.

[KLM02] S Konieczny, J Lang, and P Marquis. Distance-based merging: a general framework and some complexity results. In *Proceedings of the 8th International Conference on Principles of Knowledge Representation and Reasoning (KR'02)*, pages 97–108, 2002.

[Kon02] S Konieczny. On the difference between merging knowledge bases and combining them. In *Proceedings of the 8th International Conference on Principles of Knowledge Representation and Reasoning (KR'02)*, pages 135–144, 2002.

[KP98] S Konieczny and R Pino Perez. On the logic of merging. In *Proceedings of the Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR'98)*, pages 488–498. Morgan Kaufmann, 1998.

[KP02] S Konieczny and R Pino Pérez. Merging information under constraints: a qualitative framework. *Journal of Logic and Computation*, 12(5):773–808, 2002.

[KS86] R Kowalski and M Sergot. A logic-based calculus of events. *New Generation Computing*, 4:67–95, 1986.

[LS98] P Liberatore and M Schaerf. Arbitration (or how to merge knowledgebases). *IEEE Transactions on Knowledge and Data Engineering*, 10:76–90, 1998.

[LS00] P Liberatore and M Schaerf. Brels: A system for the integration of knowledge bases. In *Proceedings of the 7th International Conference on Principles of Knowledge Representation and Reasoning (KR'00)*, pages 145–152, 2000.

[MS99] R Miller and M Shanahan. The event calculus in classical logic: An alternative axiomatisations. *Linkoping Electronic Articles in Computer and Information Science*, 4(16), 1999.

[Rev97] P Revesz. On the semantics of arbitration. *International Journal of Algebra and Computation*, 7:133–160, 1997.

[Sha99] M Shanahan. The event calculus explained. In M.J.Wooldridge and M.Veloso, editors, *Artificial Intelligence Today*, volume 1600 of *Springer Lecture Notes in Artificial Intelligence*, pages 409–430. Springer, 1999.

[Sho88] Y Shoham. *Reasoning about Change*. MIT Press, 1988.

[SS94] L Sterling and E Shapiro. *The Art of Prolog: Advanced Programming Techniques*. MIT Press, 1994.

Appendix

Here we give some of the clauses in the Prolog version of our case study. Below, we give the initiates, terminates, and event ordering axioms, which are part of the event reasoning system, together with the holdsafter axioms. See Section 9 for discussion of this case study.

```
initiates(E, boardConsidersBid(B, C)) :- action(E, bidMade), info(E, bid(B, C)).

initiates(E, boardConsidersBid(B, C)) :- action(E, newBidMade), info(E, bid(B, C)).

initiates(E, shareholdersConsiderBid(B, C)) :- action(E, boardAcceptsOffer), info(E, bid(B, C)).

initiates(E, buyerConsidersNewBid(B, C)) :- action(E, boardRejectstsOffer), info(E, bid(B, C)).

initiates(E, antiTrustAgenciesConsiderBid(B, C)) :- action(E, shareholdersAcceptOffer), info(E, bid(B, C)).

initiates(E, bidValue(B, C, V)) :- action(E, newBidMade), info(E, bid(B, C)), info(E, value(V)).

initiates(E, bidValue(B, C, V)) :- action(E, newBidMade), info(E, bid(B, C)), info(E, value(V)).

initiates(E, capitalisation(X, Y)) :- action(E,) , info(E, capitalisation(X, Y)).

initiates(E, location(X, Y)) :- action(E,) , info(E, location(X, Y)).
terminates(E, boardConsidersBid(B, C)) :- action(E, boardAcceptsOffer), info(E, bid(B, C)).

terminates(E, shareholdersConsiderBid(B, C)) :- action(E, shareholdersAcceptOffer), info(E, bid(B, C)).

terminates(E, antiTrustAgenciesConsiderBid(B, C)) :- action(E, referralDeemedUnnecessary), info(E, bid(B, C)).
```

```
eventorder(E2, E1) :− action(E2, boardAcceptsOffer), info(E2, bid(B, C)),
                      action(E1, newBidMade), info(E1, bid(B, C)).

eventorder(E2, E1) :− action(E2, shareholdersAcceptOffer), info(E2, bid(B, C)),
                      action(E1, boardAcceptsOffer), info(E1, bid(B, C)).

eventorder(E2, E1) :− action(E2, referralDeemedUnnecessary), info(E2, bid(B, C)),
                      action(E1, shareholdersAcceptOffer), info(E1, bid(B, C)).
```

Below, we have also included some output atoms that give the format for some merged SNRs.

```
output(E, report(buyer(B), target(C), value(V))) :−
                        holdsafter(E, bidValue(B, C, V)).

output(E, report(buyer(B, location(L), capitalization(J)), target(C), value(V))) :−
                        holdsafter(E, bidValue(B, C, V)), holdsafter(E, capitalisation(B, J)),
                        holdsafter(E, location(B, L)).

output(E, report(buyer(B), target(C), status(antiTrustAgenciesConsiderBid)) :−
                        holdsafter(E, antiTrustAgenciesConsiderBid(B, C)).
```

Below, we give some event atoms obtained from four SNRs.

```
action(e1, newBidMade).
info(e1, bid(walmart, asda)).
info(e1, value(us9billion)).

action(e2, boardAcceptsOffer).
info(e2, bid(walmart, asda)).
info(e2, location(walmart, usa)).
info(e2, capitalisation(walmart, us23billion)).

action(e3, shareholdersAcceptOffer).
info(e3, bid(walmart, asda)).
info(e3, location(asda, uk)).
info(e3, capitalisation(asda, us6billion)).

action(e4, referralDeemedUnnecessary).
info(e4, bid(walmart, asda)).
info(e4, body(euAntiTrustAgency)).
```

It is straighforward to extend this Prolog program with access rules, and to provide a meta-interpreter that would apply the access rules to input atoms to generate event atoms such as those above.