Fusion rules for context-dependent aggregation of structured news reports

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

June 24, 2004

Abstract

A NewsFusion System is a logic-based system for merging heterogeneous structured news reports. Structured news reports are XML documents, where the textentries are restricted to individual words or simple phrases, such as names and domain-specific terminology, and numbers and units. We assume structured news reports do not require natural language processing. In previous papers, we have presented aspects of a logic-based framework for merging structured news reports based on fusion rules. Fusion rules are a form of scripting language that define how structured news reports should be merged. The antecedent of a fusion rule is a call to investigate the information in the structured news reports and the background knowledge, and the consequent of a fusion rule is a formula specifying an action to be undertaken to form a merged report. It is expected that a set of fusion rules is defined for any given application. In this paper, we present a framework for aggregation predicates that can be used in fusion rules for context-dependent aggregation of structured news reports. We also present a case study that uses aggregation predicates to merge weather reports that have been obtained from a selection of websites.

1 Introduction

Syntactically, a structured news report 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 a structured news report is meant to parameterize a stereotypical situation, and so a particular structured news report is an instance of that stereotypical situation. For example, news reports on corporate acquisitions can be represented as structured news reports using tags including `buyer`, `seller`, `acquisition`, `value`, and `date`. Each phrase in a structured news report 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 structured news reports.

In order to merge structured news reports, we need to take account of the contents of the structured news reports. Different kinds of content need to be merged in different ways as illustrated below.

Example 1.1 *Consider the following two conflicting weather reports which are for the same day and same city. There are many further examples we could consider, each with particular features that indicate how the merged report should be formed.*

```
⟨weatherreport⟩
        ⟨source⟩ TV1 ⟨/source⟩
        ⟨date⟩ 19/3/02 ⟨/date⟩
        ⟨city⟩ London ⟨/city⟩
        ⟨today⟩ showers ⟨/today⟩
        ⟨temp⟩
                ⟨max⟩ 9C ⟨/max⟩
                ⟨min⟩ 3C ⟨/min⟩
        ⟨/temp⟩
        ⟨tomorrow⟩ sun ⟨/tomorrow⟩
        ⟨dewpoint⟩ 2C ⟨/dewpoint⟩
⟨/weatherreport⟩
```

```
⟨weatherreport⟩
        ⟨source⟩ TV3 ⟨/source⟩
        ⟨date⟩ 19 March 2002 ⟨/date⟩
        ⟨city⟩ London ⟨/city⟩
        ⟨today⟩ inclement ⟨/today⟩
        ⟨temp⟩
                ⟨max⟩ 45F ⟨/max⟩
                ⟨min⟩ 36F ⟨/min⟩
        ⟨/temp⟩
        ⟨tomorrow⟩ rain ⟨/tomorrow⟩
⟨/weatherreport⟩
```

We can merge them so the `source` *is* `TV1 and TV3`*, and the weather for* `today` *is* `showers and inclement`*, and the weather for* `tomorrow` *is* `sun or rain`*.*

```
⟨weatherreport⟩
        ⟨source⟩ TV1 and TV3 ⟨/source⟩
        ⟨date⟩ 19.03.02 ⟨/date⟩
        ⟨city⟩ London ⟨/city⟩
        ⟨today⟩ showers and inclement ⟨/today⟩
        ⟨temp⟩
                ⟨max⟩ 7 to 9C ⟨/max⟩
                ⟨min⟩ 2 to 3C ⟨/min⟩
        ⟨/temp⟩
        ⟨tomorrow⟩ sun or rain ⟨/tomorrow⟩
        ⟨dewpoint⟩ 2C ⟨/dewpoint⟩
⟨/weatherreport⟩
```

An alternative way of merging these reports may be possible if we have a preference for one source over the other. Suppose we have a preference for `TV3` *in the case of conflict, then we may prefer the textentry* `rain` *for the tag* `tomorrow`*. We consider various further aggregation functions for handling conflicting information in Section 4.*

In our approach to merging structured news reports, we draw on domain knowledge to help produce merged reports. The approach is based on fusion rules defined in an XML file [1] (using FusionRuleML [HS03a]). These rules are of the form $\alpha \Rightarrow \beta$, where if α holds in the knowledgebase, then β is an instruction that needs to be undertaken in the process of building a merged news report.

To merge a set of structured news reports, we start with the background knowledge and the information in the news reports to be merged, and apply the fusion rules to this information. For a set of structured news reports and a set of fusion rules, we attempt to ground each fusion rule with textentries from the structured news reports, and then check whether all the conditions of each ground fusion rule are implied by the background knowledge, and if they are, then the ground actions of the rule are added to the actionlist (a list of actions that specify how the merged report should be constructed).

A NewsFusion System is a system for merging structured news reports for a particular domain. It incorporates some modules for executing fusion rules and for executing the resulting actions. It also incorporates a set of fusion rules that has been defined for the application domain together with an appropriate background knowledgebase. The basic architecture for a NewsFusion System is given in Figure 1.

In other papers, we have (1) proposed that fusion rules can be specified in logic [Hun00a]: (2) shown how inconsistent information in structured news reports can be reasoned with using paraconsistent logics [Hun00b]; (3) presented a basic definition for fusion rules for merging structured news reports [Hun02a];

[1] Examples of fusion rules together with XML representation are available at www.cs.ucl.ac.uk/staff/a.hunter/frt

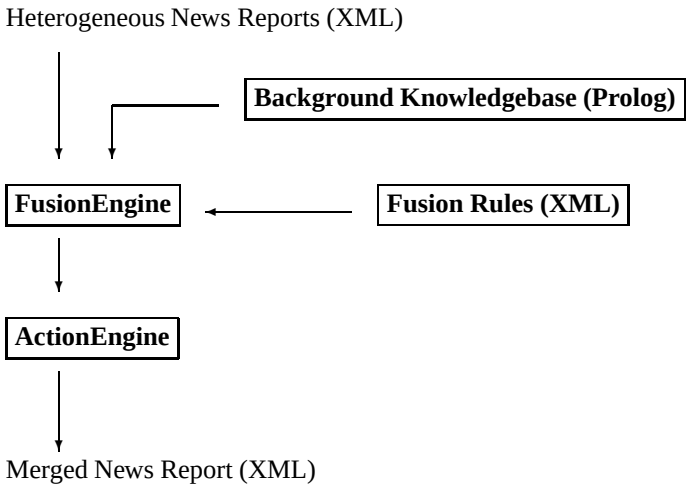

Figure 1: The basic architecture for a NewsFusion System. The two key modules are a FusionEngine and an ActionEngine implemented in Java. The fusion engine executes a set of fusion rules (an XML file containing the fusion rules marked up in FusionRuleML [HS03a]), by grounding the fusion rules with textentries from the structured news reports, and then checks whether all the conditions of each ground fusion rule are implied by the background knowledge and, if they are, then the ground actions of the rule are added to the actionlist (a list of actions that specify how the merged report should be constructed). The ActionEngine executes the actionlist to build a merged report

(4) presented a framework for using temporal logic in the background knowledge for fusion rules [Hun02b]; (5) explored properties of a restricted form of fusion rules [HS03b]; (6) developed a framework for measuring degree and significance of inconsistencies in information in order to choose how to act on inconsistency with actions including ignore, resolve and reject [Hun03, Hun04]; and (7) extended fusion rules for merging uncertain information [HL04a, HL04b].

A key aim of this paper, and indeed of the fusion rules approach, is to show how we can integrate a range of existing approaches in knowledge representation and reasoning for aggregating information. This allows us to harness (and evaluate) different theoretical approaches in a working system for context-dependent logic-based fusion. The novel contribution of this paper is: (1) to extend the definitions for fusion rules to support the use of aggregation predicates; (2) to provide a uniform presentation of aggregation predicates for encapsulating a range of logic-based approaches to aggregation; and (3) to present an extensive case study of their use in merging weather reports.

In this paper, after giving a short description of the components of a NewsFusion System, we present a number of different approaches to aggregating potentially inconsistent structured news reports that can be embodied using logical fusion rules. In particular, we define aggregation for syntactic generalization (disjunction); semantic generalization (i.e. selecting a term that is as general or more general than all of the input terms, but no more general than is necessary); aggregation using information from preferred sources; aggregation by voting, both non-weighted and weighted; and aggregation by centre of gravity. Some of these approaches, especially voting and preference over sources, admit of a variety of possibilities and several alternatives are defined. Furthermore, all of these functions have been implemented as sets of fusion rules and successfully tested using actual data from news reports on the web as part of the case study using weather reports.

2 Structured news reports

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

Definition 2.1 *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 structured news report. If ϕ is an tagname and $\sigma_1, ..., \sigma_n$ are structured news reports, then $\langle\phi\rangle\sigma_1...\sigma_n\langle/\phi\rangle$ is a structured news report.*

The definition for a structured news report 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.[2] 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 structured news reports for a domain.

Clearly each structured news report is isomorphic to a tree with the non-leaf nodes being the tagnames and the leaf nodes being the textentries.

Definition 2.2 *Let $\sigma_1, .., \sigma_n$ be tagnames occurring in a structured news report. For this, $\sigma_1/../\sigma_n$ is a* **branch** *of this structured news report iff σ_1 is the root, and for all i, if $1 \leq i < n$, then σ_i is the parent of σ_{i+1}.*

A skeleton is equivalent to a structured news report without text entries. It is the underlying structure without the content. We need this definition later.

Definition 2.3 *A* **skeleton** *is of the form $\phi(\psi_1, .., \psi_n)$ where ϕ is a tagname and $\psi_1, .., \psi_n$ are skeletons. If ψ_i is a tagname, then it is a skeleton. A skeleton $\phi(\psi_1, .., \psi_n)$ can be regarded as a tree.*

In a NewsFusion System, we assume a naming of the input reports with names from a set of report names. In this paper we use $\{\mathtt{in}_1, .., \mathtt{in}_n\}$ as the set of report names. When a set of reports is given as input, we need a process of **registration** to assign each report with a report name. If all reports are treated equally by a set of fusion rules, we can make an arbitrary assignment. But in some cases, we may want to treat them differently. For example, we may want to assign the weather report from the BBC website to $\mathtt{in}_1$, and the weather report from the CNN website to $\mathtt{in}_2$, etc. Registration is therefore a process that needs to be done before executing the fusion rules. Once a structured news report has been registered, we can refer to it by name.

3 Fusion rules

In this section, we present the framework for specifying and using a set of fusion rules. The definitions in this section extend the syntax for fusion rules given in [HS03b] by introducing logical variables.

[2] For more information on DTDs, see www.w3.org

3.1 Schema variables

When merging a set of reports, a set of fusion rules is instantiated with the information (textentries) from the reports. Schema variables specify what to instantiate and from which source. There are two types of schema variable, namely textentry variable and set variable.

Definition 3.1 *Let $\phi_1/../\phi_n$ be a branch, and let $\mathtt{in_i}$ be a report. A* **textentry variable** *is denoted $\mathtt{in_i}//\phi_1/.../\phi_n$. A* **set variable** *is denoted $//\phi_1/.../\phi_n$. A* **schema variable** *is either a set variable or a textentry variable.*

1. *For a report $\mathtt{in_i}$, $\mathtt{T}$ is the* **valid grounding for a textentry variable** $\mathtt{in_i}//\phi_1/.../\phi_k$ *iff (1) $\mathtt{T}$ is a textentry and $\mathtt{T}$ is the child of ϕ_k in $\phi_1/.../\phi_k$ in report $\mathtt{in_i}$ or (2) there is no branch $\phi_1/.../\phi_k$ in $\mathtt{in_i}$ and $\mathtt{T}$ is the null value denoted by the symbol* $\mathtt{null}$.
2. *For a set of input reports $\{\mathtt{in_1}, .., \mathtt{in_n}\}$, the list $[\mathtt{T_1}, .., \mathtt{T_n}]$ is the* **valid grounding for a set variable** $//\phi_1/.../\phi_k$ *iff $\mathtt{T_1}$ is the* **valid grounding for a textentry variable** $\mathtt{in_1}//\phi_1/.../\phi_k$ *and ... and $\mathtt{T_n}$ is the* **valid grounding for a textentry variable** $\mathtt{in_n}//\phi_1/.../\phi_k$

The $\mathtt{null}$ symbol can appear as the value of any textentry variable. It is intended to fulfill the same function as what is sometimes referred to as the *open* null value in database theory [ZP96]. Therefore, it denotes that "it is unknown whether a value exists", and so it should not be confused with values such as "zero", or "none". For example, if the marital status of a person is unknown, then the name of the spouse is assigned the open null value. The use of the open null value has implications for the aggregation functions we discuss in Section 4. We also introduce further types of null value in Section 4 that capture forms of *inexistent* null value. An inexistent null value, as proposed in database theory, is used when a value for an attribute does not exist [ZP96].

Example 3.1 *The valid grounding for the variable $\mathtt{in_1}//\mathtt{weatherreport}/\mathtt{today}$ is $\mathtt{showers}$ if we let $\mathtt{in_1}$ be the top left report in Example 1.1. Similarly, the valid grounding for $\mathtt{in_2}//\mathtt{weatherreport}/\mathtt{today}$ is $\mathtt{inclement}$ if we let $\mathtt{in_2}$ be the top right report in Example 1.1.*

Example 3.2 *The valid grounding for the set variable $//\mathtt{weatherreport}/\mathtt{today}$ is $[\mathtt{showers}, \mathtt{inclement}]$ if we let $\mathtt{in_1}$ and $\mathtt{in_2}$ be the two top structured news reports in Example 1.1.*

Example 3.3 *The valid grounding for the set variable $//\mathtt{weatherreport}/\mathtt{dewpoint}$ is $[\mathtt{showers}, \mathtt{null}]$ if we let $\mathtt{in_1}$ and $\mathtt{in_2}$ be the two top structured news reports in Example 1.1.*

Due to space pressures, we have omitted consideration of subtrees in this paper. As explained in [HS03b], these allow schema variables to take subtrees from the structured news reports. Since each structured news report is isomorphic to a ground logic term, these subtrees are treated as ground logical terms in fusion rules.

3.2 Syntax for fusion rules

In the following definition, we augment the usual definition for a classical logic language with notation for schema variables which are just placeholders to be instantiated with textentries before logical reasoning. In effect they provide the "input" for a fusion rule. Logical variables are the other kind of variables that we use in fusion rules. They are just the usual classical variables. As we explain below, after we ground the

schema variables, the literals in the antecedent of a fusion rule are conditions to be evaluated as queries in a Prolog knowledgebase, and the logical variables are ground by the Prolog knowledgebase if the respective queries succeed.

Definition 3.2 *We assume the following sets of symbols: (1) $\mathcal{C}$ is a set of constants; (2) $\mathcal{V}$ is a set of logical variables; (3) $\mathcal{S}$ is a set of schema variables; (4) $\mathcal{F}$ is a set of functions; and (5) $\mathcal{P}$ is a set of predicates.*

1. *The set of simple terms $\mathcal{T}_1$ is $\mathcal{C} \cup \{f(d_1, .., d_k) \mid f \in \mathcal{F}$ and $d_1, .., d_k \in \mathcal{C} \cup \mathcal{V} \cup \mathcal{S}\}$.*
2. *The set of schemafree terms $\mathcal{T}_2$ is $\mathcal{C} \cup \{f(d_1, .., d_k) \mid f \in \mathcal{F}$ and $d_1, .., d_k \in \mathcal{C} \cup \mathcal{V}\}$.*
3. *The set of ground terms $\mathcal{T}_3$ is $\mathcal{C} \cup \{f(c_1, .., c_k) \mid f \in \mathcal{F}$ and $c_1, .., c_k \in \mathcal{C}\}$.*
4. *The set of simple atoms $\mathcal{A}_1$ is $\{p(t_1, .., t_k) \mid p \in \mathcal{P}$ and $t_1, .., t_k \in \mathcal{T}_1\}$.*
5. *The set of schemafree atoms $\mathcal{A}_2$ is $\{p(t_1, .., t_k) \mid p \in \mathcal{P}$ and $t_1, .., t_k \in \mathcal{T}_2\}$.*
6. *The set of ground atoms $\mathcal{A}_3$ is $\{p(t_1, .., t_k) \mid p \in \mathcal{P}$ and $t_1, .., t_k \in \mathcal{T}_3\}$.*

We assume that all possible textentries are constant symbols in $\mathcal{C}$. We also assume that if $\phi_1/../\phi_n$ is a branch, then it is a constant symbol in $\mathcal{C}$. Similarly, if $\phi(\psi_1, .., \psi_n)$ is a skeleton, then it is a constant symbol in $\mathcal{C}$. We also assume that if α is an atom, then α is a literal, and $\neg\alpha$ is a literal. Let $\mathcal{L}_1$ be the literals formed from $\mathcal{A}_1$, $\mathcal{L}_2$ be the literals from $\mathcal{A}_2$, and $\mathcal{L}_3$ be the literals from $\mathcal{A}_3$.

We now consider the definition for fusion rules which may contain both logical variables and schema variables.

Definition 3.3 *A* **fusion rule** *is of the following form where $\alpha_1, .., \alpha_n \in \mathcal{L}_1$ and $\beta_1, .., \beta_m \in \mathcal{A}_1$, and the logical variables in $\beta_1, .., \beta_m$ are a subset of the logical variables in $\alpha_1, .., \alpha_n$.*

$$\alpha_1 \wedge .. \wedge \alpha_n \Rightarrow \beta_1 \wedge .. \wedge \beta_m$$

We call $\alpha_1, .., \alpha_n$ the condition literals and $\beta_1, .., \beta_m$ the action atoms.

We regard a fusion rule that incorporates schema variables, as a scheme for one or more classical formulae. These classical formulae are obtained by grounding schema variables as we explain below. We discuss condition literals in Section 3.3 and action atoms in Section 3.4.

Example 3.4 *Consider the following condition literal with the set of input reports $\{\mathtt{in}_1, \mathtt{in}_2, \mathtt{in}_3\}$*

$$\mathtt{SameCity}(//\mathtt{weatherreport}/\mathtt{city})$$

Now suppose that `London` *is the textentry on the* `weatherreport`/`city` *branch for reports $\mathtt{in}_1$ to $\mathtt{in}_3$. So after grounding the schema variable, we have the following ground condition literal.*

$$\mathtt{SameCity}([\mathtt{London}, \mathtt{London}, \mathtt{London}])$$

Definition 3.4 *A* **schemafree fusion rule** *is a fusion rule with every schema variable systematically replaced by a valid grounding according to Definition 3.1.*

Example 3.5 *The following is a propositional fusion rule that could be used for Example 1.1 where the set of news reports is given by the two top reports in Example 1.1.*

$$\texttt{Samecity}(//\texttt{weatherreport}/\texttt{city}) \Rightarrow \texttt{AddText}(\texttt{in}_1//\texttt{weatherreport}/\texttt{city}, \texttt{weatherreport}/\texttt{city})$$

The schemafree fusion rule obtained with the fusion rule is the following.

$$\texttt{Samecity}([\texttt{London}, \texttt{London}]) \Rightarrow \texttt{AddText}(\texttt{London}, \texttt{weatherreport}/\texttt{city})$$

So if $\alpha'_1 \wedge .. \wedge \alpha'_n \Rightarrow \beta'_1 \wedge .. \wedge \beta'_m$ is a fusion rule, then it is a schemafree fusion rule iff $\alpha'_1, .., \alpha'_n \in \mathcal{L}_2$ and $\beta'_1, .., \beta'_m \in \mathcal{A}_2$. A schemafree fusion rule is a formula of classical predicate logic where the logical variables are implicitly universally quantified, and the universal quantifiers are assumed to be outermost. In other words, if $\alpha'_1 \wedge .. \wedge \alpha'_n \Rightarrow \beta'_1 \wedge .. \wedge \beta'_m$ is a schemafree fusion rule, and $X_1, .., X_k$ are the free variables in $\alpha'_1, .., \alpha'_n, \beta'_1, .., \beta'_m$, then $\forall X_1, .., X_k \ \alpha'_1 \wedge .. \wedge \alpha'_n \Rightarrow \beta'_1 \wedge .. \wedge \beta'_m$ is a classical predicate formula.

3.3 Condition literals

The condition literals in fusion rules relate the contents of structured news reports to the background knowledge. There are many possible condition literals that we could define that relate one or more features from one or more structured news reports to the background knowledge. To illustrate, these literals may include the following kinds: $\texttt{SameDate}(\texttt{L})$ where the textentries in the list L refer to the same date; $\texttt{SameCity}(\texttt{L})$ where the textentries in L refer to the same city; $\texttt{SameDateAndCity}(\texttt{L}_1, \texttt{L}_2)$ where the textentries in the list $\texttt{L}_1$ refer to the same date; and the textentries in $\texttt{L}_2$ refer to the same city; $\texttt{Synonymous}(\texttt{L})$ where the textentries in L are synonyms; $\texttt{Coherent}(\texttt{L})$ where the textentries in L are mutually coherent; and $\texttt{WithinRange}(\texttt{L}, \texttt{S})$ where the values represented by the textentries in L are in the range specified by S.

Example 3.6 *Some examples of condition literals (when ground) may include the following.*

$$\texttt{SameDate}([\texttt{14April2003}, \texttt{14/04/03}, \texttt{14.4.03}, \texttt{14/04/03}])$$
$$\texttt{SameCity}([\texttt{Mumbai}, \texttt{Bombay}])$$
$$\texttt{Coherent}([\texttt{sun}, \texttt{sunny}, \texttt{sun}])$$
$$\texttt{Coherent}([\texttt{showers}, \texttt{inclement}])$$
$$\texttt{Synonymous}([\texttt{showers}, \texttt{rain}])$$
$$\texttt{WithinRange}([\texttt{11C}, \texttt{12C}, \texttt{11C}, \texttt{10C}, \texttt{12C}, \texttt{15C}, \texttt{14C}], \texttt{10}-\texttt{15C})$$

The condition literals are evaluated by querying the Prolog knowledgebase. If a condition literal incorporates logical variables, these variables are handled as queries by the Prolog knowledgebase. So if the queries succeed, a grounding for the variable is returned by the background knowledge. Furthermore, any grounding is used to systematically instantiate further occurrences of that variable in the other literals in the fusion rule. For this we require the following definition for grounding logical variables.

Definition 3.5 *A* **ground fusion rule** *is a fusion rule formed from a schemafree fusion rule with every logical variable systematically replaced by a valid grounding by the Prolog knowledgebase.*

So if $\alpha''_1 \wedge .. \wedge \alpha''_n \Rightarrow \beta''_1 \wedge .. \wedge \beta''_m$ is a fusion rule, then it is a ground fusion rule iff $\alpha''_1, .., \alpha''_n \in \mathcal{L}_3$ and $\beta''_1, .., \beta''_m \in \mathcal{A}_3$.

Example 3.7 *In the following condition,* X *is a logical variable, and* Interval *is a predicate that captures the interval of textentries, and so* X *will be ground by the prolog knowledgebase with a value that according*

to the background knowledge is the interval corresponding to the textentries in the input reports. Below the textentries are on the $\texttt{weatherreport/temp/max}$ *branch of the* $\texttt{in}_1$ *and* $\texttt{in}_2$ *reports (given as the top two reports in Example 1.1).*

$$\texttt{Interval}(\texttt{in}_1\texttt{//weatherreport/temp/max}, \texttt{in}_2\texttt{//weatherreport/temp/max}, \texttt{X})$$

After grounding the schema variables, we have the following.

$$\texttt{Interval}(\texttt{9C}, \texttt{45F}, \texttt{X})$$

Then the grounding for $\texttt{X}$ *returned by the Prolog knowledgebase is* $\texttt{7}-\texttt{9C}$ *if we assume appropriate clauses in the Prolog knowledgebase.*

In our Java prototype for executing fusion rules, conditions are evaluated from left to right. So for a schemafree fusion rule $\alpha'_1 \wedge .. \wedge \alpha'_n \Rightarrow \beta'_1 \wedge .. \wedge \beta'_m$, first the condition literal α'_1 is processed as a Prolog query, then the the condition literal α'_2 is processed as a Prolog query, and so on. This means the query variables are evaluated and the result instantiated systematically from left to right. In the current implementation, negation is treated as negation-as-failure in Prolog. If the Prolog queries α'_1,..,α'_n all succeed (i.e. the Prolog knowledgebase implies α'_1,..,α'_n), then we say that the fusion rule has been **fired**, otherwise we say that it has **failed**.

We assume the background knowledge is defined by a knowledge engineer building a NewsFusion System for an application. We have used Prolog because it is currently the most practical way of using logic in knowledgebased applications. As found in the weather reports case study, it is straightforward to encode the necessary background knowledge. Prolog has been used in a wide variety of knowledgebased applications, and so it is likely we can use this approach in a variety of domains. Some of the Prolog clauses defined will be applicable in a variety of domains (e.g. the aggregation predicates), and some will need to be defined for each application (e.g. ontological information providing relationships between the textentries used in a domain).

Since the background knowledge is handled in Prolog, there is the possibility of multiple instantiations for query variables. So far we have constrained the knowledgebase to provide unique instantiations, but an alternative strategy is to take the first answer only. As will be apparent in Section 3.5, we do not support backtracking of fusion rules.

3.4 Action atoms

Action atoms specify the structure and content for a merged report. In a ground fusion rule $\alpha'' \Rightarrow \beta''$, if the ground literals in the antecedent α'' succeed as queries to the Prolog knowledgebase, then the merged report should meet the specification represented by the ground atoms in β''. We look at this in more detail in the next section. Here we consider the syntax for action literals. In Definition 3.6, we define a basic set of action atoms.

Definition 3.6 *The* **action atoms** *are literals that include the following specifying how the merged report should be constructed.*

1. $\texttt{Initialize}(\phi(\psi_1, .., \psi_n))$ *where* $\phi(\psi_1, .., \psi_n)$ *is a skeleton constant. The intended action is to start the construction of the merged structured news report with* $\phi(\psi_1, .., \psi_n)$ *defining the basic structure. So the root of the merged report is* ϕ*.*
2. $\texttt{AddText}(\texttt{T}, \phi_1/../\phi_n)$ *where* $\texttt{T}$ *is a textentry, and* $\phi_1/../\phi_n$ *is a branch constant. The intended action is to add the textentry* $\texttt{T}$ *as the child to the tagname* ϕ_n *in the merged report on the branch* $\phi_1/../\phi_n$*.*

3. AddNode(N, $\phi_1/../\phi_n$) *where* N *is a tagname for a node, and* $\phi_1/../\phi_n$ *is a branch constant. The intended action is to add* N *as the child to the tagname* ϕ_n *in the merged report on the branch* $\phi_1/../\phi_n$.

The action atoms are specifications that are intended to be made to hold by producing a merged report that satisfies the specification. For further analysis of the logic of action atoms, see [Hun02a].

Example 3.8 *Consider the action atom in the consequent of Example 3.5. After grounding the schema variable, the action atom is now the following:*

AddText(London, weatherreport/city)

This specifies that the textentry should be London *in the merged report for the tagname* city *on the branch* weatherreport/city.

The set of action atoms that we have defined in this paper is only part of the range of possible action atoms. We have implemented others including ExtendTree(T, $\phi_1/../\phi_n$) where T is a ground term representing a tree (which is isomorphic to part of a structured news report), and $\phi_1/../\phi_n$ is a branch constant, and the intended action is to extend the merged report (which is also a tree) with T so that the tagname for the root of T is ϕ_n on the branch $\phi_1/../\phi_n$.

3.5 Rule execution

In order to use a set of fusion rules, we need to be able to execute them with background knowledge and a set of reports. Essentially each rule in turn is ground with the information extracted from the input set of structured news reports. Then for each ground fusion rule $\alpha'' \Rightarrow \beta''$, if each of the conditions in α'' succeeds as a query to the Prolog knowledgebase, then all of the actions in β'' are added to the actionlist, and so the merged report should meet the specification β''. In the Java implementation of the modules for NewsFusion Systems, this is the basis of the key algorithm in the FusionEngine. The ground actions in the actionlist specify the structure and content for a merged report. The minimum we expect of the ground actions in an actionlist is that a merged report is produced that meets the specification. In the Java implementation, this is the basis of the key algorithm in the ActionEngine (see Figure 1).

Example 3.9 *The pair of reports given in the top of Example 1.1, together with an appropriate set of fusion rules and an appropriate background knowledge would be executed by a NewsFusion System to give the following actionlist which specifies the merged report given in the bottom of Example 1.1.*

Initialize(weatherreport(source, date, country, today, temp(max, min), tomorrow))
AddText(TV1 and TV3, weatherreport/source)
AddText(19.03.02, weatherreport/date)
AddText(London, weatherreport/city)
AddText(showers and inclement, weatherreport/today)
AddText(7 to 9C, weatherreport/temp/max)
AddText(2 to 3C, weatherreport/temp/min)
AddText(sun or rain, weatherreport/tomorrow)
AddNode(dewpoint, weatherreport)
AddText(2C, weatherreport/dewpoint)

The fusion rules in a NewsFusion System need to be engineered so that an Initialize atom is obtained for any set of reports that are to be covered by the system, and no Initialize atom is to be obtained for any set of reports that are not to be covered by the NewsFusion System.

FusionEngine(Reports,FRules,ORules)

while FRules $\neq \emptyset$ and FActions $= \emptyset$

let $\alpha_1 \wedge \ldots \wedge \alpha_n \Rightarrow \beta_1 \wedge \ldots \wedge \beta_m \in$ FRules

let FRules $:=$ FRules $\setminus \{\alpha_1 \wedge \ldots \wedge \alpha_n \Rightarrow \beta_1 \wedge \ldots \wedge \beta_m\}$

let $\alpha'_1 \wedge \ldots \wedge \alpha'_n \Rightarrow \beta'_1 \wedge \ldots \wedge \beta'_m :=$ Ground(Reports, $\alpha_1 \wedge \ldots \wedge \alpha_n \Rightarrow \beta_1 \wedge \ldots \wedge \beta_m$)

while $1 \leq i \leq n$ and $\Delta \vdash_{prolog} \alpha'_i[x_i/t_i]$

let $\alpha'_1 \wedge \ldots \wedge \alpha'_n \Rightarrow \beta'_1 \wedge \ldots \wedge \beta'_m := (\alpha'_1 \wedge \ldots \wedge \alpha'_n \Rightarrow \beta'_1 \wedge \ldots \wedge \beta'_m)[x_i/t_i]$

if $i < n$, **then let** $i := i + 1$

if $i = n$, **then let** FActions $:= \{\beta'_1, \ldots, \beta'_m\}$

if FActions $\neq \emptyset$

then while ORules $\neq \emptyset$

let $\alpha_1 \wedge \ldots \wedge \alpha_n \Rightarrow \beta_1 \wedge \ldots \wedge \beta_m \in$ ORules

let ORules $:=$ ORules $\setminus \{\alpha_1 \wedge \ldots \wedge \alpha_n \Rightarrow \beta_1 \wedge \ldots \wedge \beta_m\}$

let $\alpha'_1 \wedge \ldots \wedge \alpha'_n \Rightarrow \beta'_1 \wedge \ldots \wedge \beta'_m :=$ Ground(Reports, $\alpha_1 \wedge \ldots \wedge \alpha_n \Rightarrow \beta_1 \wedge \ldots \wedge \beta_m$)

while $1 \leq i \leq n$ and $\Delta \vdash_{prolog} \alpha'_i[x_i/t_i]$

let $\alpha'_1 \wedge \ldots \wedge \alpha'_n \Rightarrow \beta'_1 \wedge \ldots \wedge \beta'_m := (\alpha'_1 \wedge \ldots \wedge \alpha'_n \Rightarrow \beta'_1 \wedge \ldots \wedge \beta'_m)[x_i/t_i]$

if $i < n$, **then let** $i := i + 1$

if $i = n$, **then let** OActions $:=$ OActions $\cup \{\beta'_1, \ldots, \beta'_m\}$

let Actions $:=$ FActions $\cup$ OActions

return Actions

Figure 2: The algorithm for the FusionEngine. Let FRules be the set of foundational rules and let ORules be the set of optional rules. Let FActions be the set of foundational actions, OActions be the set of optional actions, and let Actions be the set of actions. Let Reports be a set of structured news reports given as input. Let Ground(Reports, $\alpha_1 \wedge \ldots \wedge \alpha_n \Rightarrow \beta_1 \wedge \ldots \wedge \beta_m$) ground the schema variables in $\alpha_1 \wedge \ldots \wedge \alpha_n \Rightarrow \beta_1 \wedge \ldots \wedge \beta_m$ using the reports in Reports. Let $(\alpha'_1 \wedge \ldots \wedge \alpha'_n \Rightarrow \beta'_1 \wedge \ldots \wedge \beta'_m)[x_i/t_i]$ denote the grounding of each occurrence of the logical variable x_i by the term t_i in the formula $\alpha'_1 \wedge \ldots \wedge \alpha'_n \Rightarrow \beta'_1 \wedge \ldots \wedge \beta'_m$. Let Δ be the Prolog knowledgebase.

Fusion rules that contain an `Initialize` atom clearly have a special status; if they all fail, no other actions should be executed because there cannot be a merged report without a skeleton to which textentries and nodes are to be attached. But other rules may also enjoy a similar status. This is because for most domains of interest, certain report elements enjoy a privileged position in that if conditions involving those elements are not met, the reports in question should not be merged. In the case of weather reports, for example, if the reports are not for the same date, location, and from accepted sources, no merged report should be constructed. Conditions involving key elements such as these are, therefore, combined into rules, (together with the `Initialize` atom) whose status is then flagged as being *foundational* in the FusionRuleML: if no fundational rule is fired, then no merged report is constructed. If there is more than one foundational rule in a set of fusion rules, then the first foundational rule to be fired, causes the subsequent foundational rules to be disabled.

The case is different for all other report elements (whether those elements are expected to appear in all reports, or only expected to occur in some reports): The fusion rules that handle these other report elements are flagged as *optional*, meaning that if such a rule fails, the fusion engine will still attempt to execute further rules, and a merged report will still be produced. The fact that some rules are deemed optional does, however, have consequences for the rule engineer. For these report elements, not only do we need a fusion rule telling us what to do if a certain condition concerning one of those elements holds, we also need one or more fusion rules telling us what to do if that condition *fails*. In the simplest case, rules that deal with report elements that are not foundational occur in pairs (though see subsection 4.3 below for an exception). For example, if the dewpoints forecast in the reports in Example 1.1 all fall within the accepted range, then an interval consisting of the minimum and maximum values, together with the unit suffix, is constructed. If, on the other hand, not all forecast dewpoints fall within that range, then we need a second

rule that might, for example, discard outlying values, and construct a similar interval from the remaining values.

Optional rules may, of course, be grouped together in more complicated ways, as reflected in the logical nature of the tests embodied in their conditions. We might, for example, have a group of rules with conditions of the following form: (1) If p is true and q is true, then execute action A_1; (2) If p is true and q is false, then execute action A_2; (3) If p is false and r is true, then execute action A_3; and (4) If p is false and r is false, then execute action A_4. Clearly, the number of rules that need to be grouped together in this way depends upon the complexity of the knowledge in the subject area in question. In fact, there is a close interdependence between the rules and the background knowledge in the knowledgebase, as can be shown by considering two potential problems that must be avoided when formulating such groups of rules.

The first problem to avoid is that we do not want two (or more) rules to fire whose actions involve doing something to the *same node or element* in the merged report. What is crucial about any such grouping of rules whose actions concern the same merged report element is that, for any given list of input textentries, *at most one rule in that group should fire*. In other words, it is not enough that the conditions of such a group should be such that there is no set of circumstances in which (according to the knowledgebase) they should *all* be true. Rather, there should be no set of circumstances in which the conditions of more than one rule should be true. However, whether or not this condition is violated by a group of rules cannot be determined simply by inspecting the logical form of their conditions. For example, two rules with conditions of the form (1) p and q, and (2) p and r, may seem to violate this condition in that both sets of conditions could be true, given a suitable list of input textentries. However, it may be that, given the background knowledge in the knowledgebase, whenever q is true r is false, and vice versa, so that in fact there is no possibility of both rules firing.

The second problem to avoid is for the conditions in a group of rules not to cover all possible truth-functional combinations of inputs, so that for a given list of input textentries *no* rule would fire when the merged report ought in fact to contain some information. Consider, for example two rules with conditions of the form (1) p and q, and (2) $not\ p$. In this case the first problem is avoided, since if the first rule's conditions are true the second rule's must be false, and vice versa. But if p is true and q is false neither rule will fire and the merged report may well be lacking information it ought to contain. However, whether another rule should be added (with conditions p and $not\ q$) depends, once again, on the background knowledge. If the background knowledge is such that if p is true q must also be true then the rules are satisfactory as they stand.

A final point worth making is that it is not the case that in such a grouping of rules whose actions all concern the same output report element, and for any input set of textentries, the conditions of *at least* one rule should be *true*. Rather, the conditions of a group of rules that belong together are formulated so that they can all *fail*. Intuitively, this is desirable behaviour. If none of the input reports has a textentry for the report element with which the conditions of the rules in the group are concerned (the set variable consists of a list of `nulls`), we want no action to be taken, and so all the rules in the group should fail. For this reason negation-as-failure is in general not employed in formulating conditions for groups of rules (though of course it is frequently used with predicates in the Prolog knowledgebase), purpose-built predicates being defined instead. As an instance of this consider Example 3.10: if the set variable that is the argument to `EquivalentTerms` in rule 12 is ground with a list of textentries that are all `null`, then the predicate will fail. This means, of course, that were `NOT EquivalentTerms` to be used with the same set variable as an argument in rule 13, it would succeed, leading to the risk that if other conditions in the rule were also to succeed, a `null` textentry would be added to the merged report. As this is deemed undesirable, a new predicate, `NotEquivalentTerms` is defined that also fails if the set variable is ground with a list of textentries that are all `null`.

Example 3.10 *A pair of rules for merging weather reports dealing with the entry for today's weather. The rule code indicates that these have been registered as the twelfth and thirteenth rules in the rulebase. The status of both is optional. Rule 12 says that if all the textentries for today's weather are from the same*

equivalence class, the preferred term from that class is selected and that text is added to the `today` *tag in the merged report. If the first condition of rule 12 fails, then, provided that the values of the textentry variables for the report element* `today` *are not all* `null`*, the first condition of rule 13 will succeed. The second condition simply constructs the disjunction of each textentry for today's weather from the set of input reports (duplicates and nulls being removed), and the action adds the disjunction as the textentry for the branch* `weatherreport/today`*.*

$$\texttt{Rulecode is 12}\ (\texttt{Status} = \texttt{optional})$$

$$\texttt{EquivalentTerms}(\texttt{//weatherreport/today})$$
$$\texttt{AND PreferredTerm}(\texttt{//weatherreport/today}, \texttt{X})$$
$$\texttt{IMPLIES AddText}(\texttt{X}, \texttt{weatherreport/today})$$

$$\texttt{Rulecode is 13}\ (\texttt{Status} = \texttt{optional})$$

$$\texttt{NotEquivalentTerms}(\texttt{//weatherreport/today})$$
$$\texttt{AND Disjunction}(\texttt{//weatherreport/today}, \texttt{X})$$
$$\texttt{IMPLIES AddText}(\texttt{X}, \texttt{weatherreport/today})$$

4 Aggregation predicates

A common problem in merging reports arises when they do not agree on a particular textentry. For example, for `weatherreport/precipitation`, we may have the first report with textentry `rain`, the second with `rain`, and the third with `sun`, which we can represent by the list $[\texttt{rain}, \texttt{rain}, \texttt{sun}]$. In order to select a textentry for the merged report, we need to aggregate the list. To do this, we use a special kind of condition literal called an aggregation predicate. The arguments of an aggregation predicate include the textentries to be aggregated, any other relevant contextual information in the form of textentries, and a logical variable. The Prolog knowledgebase evaluation of the aggregation predicate finds a grounding of this logical variable which constitutes the aggregation. In case there is no acceptable aggregation of the list of textentries, then the grounding is a type of null value.

There are a number of different approaches to aggregation predicates that we may consider, but each of them can be captured as a condition literal. Note, the key advantage of using aggregation predicates within fusion rules is that the choice of aggregation can be dependent on the context of the input reports, so that different report elements can be merged in different ways. Note, also, that whichever approach is selected, care must be taken in dealing with `null` textentries because, as mentioned in Section 3 above, these are not values but rather indicate that it is not known whether a value exists. When merging conflicting textentries by voting, for example, we must not tally up the `null` entries and consider them votes for a particular value. In most cases, we can deal with the `null` entries by simply stripping them from the list of textentries before applying the aggregation predicates. However, matters are not quite so simple in the case of aggregation by using preferences over sources, or in the case of weighted voting, as we explain below.

Definition 4.1 *Let Ω be a list of textentries of the form $[\psi_1, .., \psi_n]$. Since this is a tuple, there may be multiple occurrences of one or more textentries in the tuple. Let the set of textentries in Ω be given by* $\mathsf{Set}(\Omega)$.

Example 4.1 *Let* $\Omega = [\texttt{sun}, \texttt{sun}, \texttt{rain}, \texttt{rain}, \texttt{rain}]$. *So* $\mathsf{Set}(\Omega) = \{\texttt{sun}, \texttt{rain}\}$.

In the following subsections, we consider a range of aggregation predicates, including syntactic generalization (disjunction), semantic generalization (subsumption), preference by sources, voting, weighted voting, and centre of gravity.

4.1 Aggregation by syntactic generalization

The simplest form of aggregation that we consider is to take the syntactic generalization of the input textentries that is obtained by taking the disjunction of the input textentries.

Definition 4.2 *Let Ω be a tuple of textentries of the form and let* `X` *be a logical variable. Also let* $\mathsf{Set}(\Omega) = \{\psi_1, .., \psi_m\}$. *The aggregation predicate* $\texttt{Disjunction}(\Omega, \texttt{X})$ *is defined so* `X` *is ground to* ψ_1 `or ...` `or` ψ_m.

Example 4.2 *For* $\texttt{Disjunction}([\texttt{rain}, \texttt{snow}, \texttt{rain}, \texttt{sun}, \texttt{sun}], \texttt{X})$, `X` *is ground to* `rain or snow or sun`.

Syntactic generalization captures a common approach to aggregation that can be viewed as a credulous perspective: Any of the options may be regarded as correct and so their disjunction may be regarded as correct. The name "syntactic generalization" is appropriate because no account is taken of the meaning of the textentries. Though of course we may use the `Disjunction` predicate with other predicates. For example, it could be used in conjunction with `NotEquivalentTerms` which, as introduced above, checks the textentries do not belong to the same equivalence class (as defined by the knowledgebase). The second of the rules (`Rulecode13`) in Example 3.10 is an illustration of this approach to conflicting information.

4.2 Aggregation by semantic generalization

Aggregation by semantic generalization selects a textentry that is sufficiently general to generalize on all the input textentries, but no more general than necessary. See for example Example 4.3. The textentry selected for the merged report will, then, represent the least upper bound of those terms that are general enough to include all of the (non-null) textentries in the input reports. Here generality is semantic generality and it is defined in terms of a semantic network encoded in Prolog. If the semantic network is implemented in the knowledgebase as a tree, then the least upper bound will be the node deepest in the tree that is a member of the upset of each of the nodes to be generalized. If there is no such node, because not all of the textentries belong to the same semantic network, then the textentry in the merged report is assigned the null value `NoLeastUpperBound`.

The kind of relation that must hold between two textentries if one is to be more general than the other is sometimes referred to as a generic relationship or an inclusion relationship. This is the relationship between a class and its subclasses; the class is more general than its subclasses because every member of the subclasses is a member of the original class, but not vice versa. So for example, vertebrate is more general than mammal, amphibian, bird, fish, and reptile, because all mammals (etc.) are vertebrates, but not all vertebrates are mammals. Figure 3 gives a semantic network for a weather report knowledgebase.

In this section, we use a restricted form of semantic network called a semantic tree which we formalize in Definition 4.3. Figure 3 is also an example of a semantic tree.

Definition 4.3 *A* **semantic tree** *is a tree Σ of the form (N, A) where N is a set of nodes and A is a set of arcs such that $(x, y) \in A$ means x is the parent of y. A node x is an ancestor of a node z iff x is the parent of z or x is the parent of a node y and y is an ancestor of z. For a node $z \in N$, let $\mathsf{Upset}(\Sigma, z) = \{z\} \cup \{x \mid x\ is\ an\ ancestor\ of\ z\}$. For a set $M \subseteq N$, a youngest member is an element $x \in M$ such that for all $y \in M$ if $x \neq y$ then x is not an ancestor of y.*

For a semantic tree $\Sigma = (N, A)$, and for any $z \in N$, there is a unique youngest member in $\mathsf{Upset}(\Sigma, z)$. Also for a semantic tree $\Sigma = (N, A)$, and for any $x, y \in N$, the set $\mathsf{Upset}(\Sigma, x) \cap \mathsf{Upset}(\Sigma, y) \neq \emptyset$, since at least the root of the tree is in $\mathsf{Upset}(\Sigma, x) \cap \mathsf{Upset}(\Sigma, y)$.

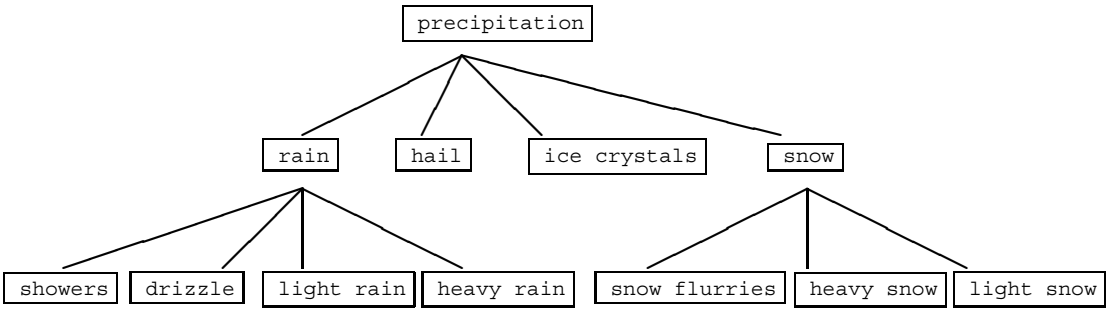

Figure 3: The tree represents a very incomplete version of a semantic network for precipitation that could be used as part of a knowledgebase for merging weather reports. The node for `rain` being a child of the node for `precipitation` should be taken to indicate that `rain` is a kind of, or type of, `precipitation`.

Now we can define the aggregation predicate $\texttt{SemanticGeneralization}(\Omega, \Sigma, \texttt{X})$ which gives the least upper bound on the textentries in Ω according to the semantic tree Σ. In other words, `X` will be ground by the Prolog knowledgebase with a value that according to the background knowledge is more general than or equal to the values in the input reports.

Definition 4.4 *Let Ω be a tuple of textentries, let $\mathsf{Set}(\Omega) = \{\psi_1, .., \psi_n\}$, let Σ be a semantic tree and let* `X` *be a logical variable. The aggregation predicate* $\texttt{SemanticGeneralization}(\Omega, \Sigma, \texttt{X})$ *is defined so that* `X` *is ground to ϕ as follows:*

If $\mathsf{Upset}(\Sigma, \psi_1) \cap \ldots \cap \mathsf{Upset}(\Sigma, \psi_n) \neq \emptyset$
Then ϕ is the youngest member in $\mathsf{Upset}(\Sigma, \psi_1) \cap \ldots \cap \mathsf{Upset}(\Sigma, \psi_n)$.
Otherwise ϕ is `NoLeastUpperBound`

The constant symbol `NoLeastUpperBound` *is a type of null value.*

Example 4.3 *In the following condition,* `X` *is a logical variable,* $\texttt{S}_1$ *is a constant symbol that is the name of the semantic tree given in Figure 3, and* `SemanticGeneralization` *gives the least upper bound of the textentries on the* `weatherreport/precipitation` *branch of the structured news reports.*

$$\texttt{SemanticGeneralization}(\texttt{//weatherreport/precipitation}, \texttt{S}_1, \texttt{X})$$

If `//weatherreport/precipitation` *is ground as* $[\texttt{rain}, \texttt{showers}, \texttt{light rain}, \texttt{drizzle}]$*, then the query sent to the Prolog knowledgebase is the following:*

$$\texttt{SemanticGeneralization}([\texttt{rain}, \texttt{showers}, \texttt{light rain}, \texttt{drizzle}], \texttt{S}_1, \texttt{X})$$

The grounding for `X` *returned by the Prolog knowledgebase is* `rain`.

Semantic generalization can be viewed as finding subsumption relationships in an ontology. Description logics can be used to represent and reason with ontologies to find subsumption relations between concepts [BCM+03]. Furthermore, implementations such as the FACT system could potentially be used to compute subsumption relations as subqueries from the background knowledge [Hor98].

4.3 Aggregation by preferences over sources

When a set of reports conflict, one might simply select the textentry from one of the reports on the grounds that its source is preferred to that of any of the others. Assuming that our preferences over sources form a linear pre-ordering (contained in the knowledgebase), the procedure is first to determine which subset of the input reports have a textentry for the report element in question; then determine which is the preferred source from amongst this subset of reports; and, finally, select the textentry from the report with that source. With this approach to resolving conflicts, particular care must be taken with the treatment of null values; if a textentry variable for a particular report is grounded by `null` then not only must that `null` be eliminated from the list of textentries, the corresponding source textentry must also be eliminated from the list of sources. A consequence of this is that the textentry selected for the merged report will not necessarily originate from the input report with the most preferred source; and, if the reports in the input set have more than one element that conflicts, then even if all of these conflicts are adjudicated using the same preference ordering, the textentries selected for the merged report may in fact originate from a number of different input reports.

Definition 4.5 *Let Π be a list of sources represented by textentries and let $\geq$ be a linear pre-ordering over sources such that $\pi \geq \pi'$ means π is preferred to π'.*

$$\mathsf{MostPreferredSources}(\geq, \Pi) = \{\pi \in \mathsf{Set}(\Pi) \mid \textit{ for all } \pi' \in \mathsf{Set}(\Pi)\ (\pi \geq \pi')\}$$

Definition 4.6 *Let Ω be a list of textentries to be aggregated and let Π be a list of textentries denoting sources s.t. $|\Omega| = |\Pi|$ and the source of ψ_i is π_i. Let $\geq$ be a constant symbol denoting the name of a pre-ordering relation. The aggregation predicate* $\mathtt{UsePreferredSource}(\Omega, \Pi, \geq, \mathtt{X})$ *is defined so that if there is a ψ_j such that* $\mathsf{MostPreferredSources}(\geq, \Pi) = \{\pi_j\}$*, then* `X` *is ground to ψ_j, otherwise* `X` *is ground to* `NoPreferredSource`*. The constant symbol* `NoPreferredSource` *is a type of null value.*

Example 4.4 *Rule 13 (below) is a fusion rule dealing with conflicting entries for today's weather by using preferences over sources. It says that if not all the textentries for today's weather are from the same equivalence class, then, from amongst those reports that have a textentry for* `today`*, choose the report with the preferred source, and select the textentry for* `today` *from that report and add it to the* `today` *tag in the merged report.*

$$\mathtt{Rulecode\ is\ 13}\ (\mathtt{Status} = \mathtt{optional})$$

$$\mathtt{NotEquivalentTerms}(//\mathtt{weatherreport}/\mathtt{today})$$
$$\mathtt{AND\ UsePreferredSource}(//\mathtt{weatherreport}/\mathtt{source}, //\mathtt{weatherreport}/\mathtt{today}, \geq, \mathtt{X})$$
$$\mathtt{IMPLIES\ AddText}(\mathtt{X}, \mathtt{weatherreport}/\mathtt{today})$$

Let the pre-ordering over the sources, denoted by $\geq$, from most preferred to least preferred be the sequence: `The Weather Channel`, `BBCi`, `BBC LDN`, `CNN`*. Then, if* $//\mathtt{weatherreport}/\mathtt{source}$ *is ground as*

$$[\mathtt{BBC\ LDN}, \mathtt{CNN}, \mathtt{BBCi}, \mathtt{The\ Weather\ Channel}]$$

and $//\mathtt{weatherreport}/\mathtt{today}$ *is ground as*

$$[\mathtt{rain}, \mathtt{null}, \mathtt{sun}, \mathtt{null}]$$

the query sent to the Prolog knowledgebase is the following:

$$\mathtt{UsePreferredSource}([\mathtt{BBC\ LDN}, \mathtt{CNN}, \mathtt{BBCi}, \mathtt{The\ Weather\ Channel}], [\mathtt{rain}, \mathtt{null}, \mathtt{sun}, \mathtt{null}], \geq, \mathtt{X})$$

The grounding for `X` *returned by the Prolog knowledgebase is* `sun` *because, of the two reports that have a textentry for* `today`*,* `BBCi` *is preferred to* `BBC LDN`*.*

Of course there may be cases where two or more sources are equally preferred. To deal with this, we may prefer the following more relaxed aggregation predicate.

Definition 4.7 *Let Ω be a list of textentries to be aggregated and let Π be a list of textentries denoting sources s.t. $|\Omega| = |\Pi|$ and the source of ψ_i is π_i. Let $\geq$ be a constant symbol denoting the name of a pre-ordering relation. The aggregation predicate* $\texttt{UsePreferredSources}(\Omega, \Pi, \geq, \texttt{X})$ *is defined so that if* $\mathsf{MostPreferredSources}(\geq, \Pi) = \{\pi_{j_1}, .., \pi_{j_n}\}$*, then* X *is ground to* ψ_{j_1} or ... or ψ_{j_n}*, otherwise* X *is ground to* NoPreferredSource*. The constant symbol* NoPreferredSource *is a type of null value.*

Example 4.5 *Let the ordering $\geq$ over the sources be: (1st)* The Weather Channel*, (2nd)* BBCi*, (2nd)* BBC LDN*, (4th)* CNN *(i.e.,* BBCi *and* BBC LDN *are equally preferred). If* //weatherreport/source *is ground as the list* [BBC LDN, CNN, BBCi, The Weather Channel]*, and* //weatherreport/today *is ground as* [rain, rainy, sun, null]*, the query sent to the Prolog knowledgebase is the following:*

$$\texttt{UsePreferredSources}([\texttt{BBC LDN}, \texttt{CNN}, \texttt{BBCi}, \texttt{The Weather Channel}], [\texttt{rain}, \texttt{rainy}, \texttt{sun}, \texttt{null}], \geq, \texttt{X})$$

The grounding for X *returned by the Prolog knowledgebase is the list* rain or sun *because, of the three reports that have a textentry for* today*, there is an equal preference for* BBCi *and* BBC LDN*, and both are preferred to* CNN*.*

A more fine-grained approach to resolving conflicts via preferences over sources is to have different preference orderings depending on the subject or context; in the case of weather reports, BBCi may be preferred to The Weather Channel for windspeed, say, whilst The Weather Channel may be preferred to BBCi for relativehumidity. Such preferences may be arbitrary in this particular case, but in merging news reports about war, for example, one might easily prefer one source for casualty figures about one side while preferring another source for casualty figures about the other side. Obviously, predicates for resolving conflicts in this way would require an additional argument for subject or context. We define this aggregation predicate, using PreferenceForSubject and UsePreferredSource predicates as follows.

The predicate PreferenceForSubject is a type of relation in the Prolog knowledgebase. For example, $\texttt{PreferenceForSubject}(\texttt{relativehumidity}, \geq_1)$ relates relativehumidity to the ordering $\geq_1$ and $\texttt{PreferenceForSubject}(\texttt{windspeed}, \geq_2)$ relates windspeed to the ordering $\geq_2$, where $\geq_1$ and $\geq_2$ are names for appropriate pre-ordering relations, that defined in the Prolog knowledgebase.

Definition 4.8 *Let* S *and* R *be schema variables, let* X *and* Y *be logical variables, and let* C *be a constant symbol denoting context. The aggregation predicate* $\texttt{UsePreferredSourceForSubject}(\texttt{S}, \texttt{R}, \texttt{C}, \texttt{X})$ *is defined by the the grounding for* X *obtained by the following query*

$$\texttt{PreferenceForSubject}(\texttt{C}, \texttt{Y}) \textit{ and } \texttt{UsePreferredSource}(\texttt{S}, \texttt{R}, \texttt{Y}, \texttt{X})$$

Example 4.6 *Consider the aggegation predicate* UsePreferredSourceForSubject *with the schema variables* weatherreport/windspeed *and* weatherreport/source *using the preferred source for the context* windspeed*.*

$$\texttt{UsePreferredSourceForSubject}(\texttt{//weatherreport/source}, \texttt{//weatherreport/windspeed}, \texttt{windspeed}, \texttt{X})$$

Let the preference ordering over the sources for the context windspeed *be: (1st)* The Weather Channel*, (2nd)* BBCi*, (3rd)* BBC LDN*, (4th)* CNN*. If* //weatherreport/source *is ground as*

$$[\texttt{BBC LDN}, \texttt{CNN}, \texttt{BBCi}, \texttt{The Weather Channel}]$$

and //weatherreport/windspeed *is ground as*

$$[\texttt{17mph}, \texttt{35mph}, \texttt{15mph}, \texttt{null}]$$

the query sent to the Prolog knowledgebase is the following:

$$\texttt{UsePreferredSourceForSubject}([\texttt{BBC LDN}, \texttt{CNN}, \texttt{BBCi}, \texttt{The Weather Channel}], [\texttt{17mph}, \texttt{35mph}, \texttt{15mph}, \texttt{null}], \texttt{windspeed}, \texttt{X})$$

The grounding for X *returned by the Prolog knowledgebase is* 15mph *because, of the three reports that have a textentry for* windspeed*, the most preferred source for that context is* BBCi*, and the textentry for* windspeed *in that report is* 15mph*.*

Preferences (such as preferences over sources) are a well-established approach to conflict resolution in knowledge representation and reasoning (e.g. non-monotonic and defeasible reasoning). Furthermore, it seems preference information is readily available for structured news reports.

4.4 Aggregation by simple voting

In this approach we will consider a form of voting. Each report that can ground a textentry variable with a value other than null "votes" for the merged textentry, and based on this idea some aggregation predicates, defined below, have been implemented in Prolog.

Definition 4.9 *Let* Ω *be a list of textentries. Let* Score *be a scoring function such that* $\mathsf{Score}(\Omega, \psi)$ *is the number of occurrences of* ψ *in* Ω*. Let* $\tau \in [0, 1]$ *be a threshold.*

$$\mathsf{MaxVote}(\Omega) = \{\psi \in \mathsf{Set}(\Omega) \mid \textit{ for all } \psi' \in \mathsf{Set}(\Omega)\ \mathsf{Score}(\Omega, \psi) \geq \mathsf{Score}(\Omega, \psi')\}$$
$$\mathsf{ThresholdVote}(\Omega, \tau) = \{\psi \in \mathsf{MaxVote}(\Omega) \mid (\mathsf{Score}(\psi)/|\Omega|) \geq \tau\}$$

Example 4.7 *Let* $\Omega = [\texttt{rain}, \texttt{snow}, \texttt{rain}, \texttt{showers}, \texttt{rain}, \texttt{snow}, \texttt{snow}, \texttt{cloudy}]$.

$$\mathsf{MaxVote}(\Omega) = \{\texttt{rain}, \texttt{snow}\}$$
$$\mathsf{ThresholdVote}(\Omega, 0.5) = \emptyset$$
$$\mathsf{ThresholdVote}(\Omega, 0.4) = \emptyset$$
$$\mathsf{ThresholdVote}(\Omega, 0.3) = \{\texttt{rain}, \texttt{snow}\}$$
$$\mathsf{ThresholdVote}(\Omega, 0.2) = \{\texttt{rain}, \texttt{snow}\}$$

Definition 4.10 *Let* Ω *be a list of textentries. Let* τ *be a constant symbol denoting a threshold. The simple voting predicates are defined as follows, where* X *is a logical variable to be ground by the Prolog knowledgebase:*

1. *For* $\texttt{Majority}(\Omega, \texttt{X})$*, if* $\mathsf{MaxVote}(\Omega) = \{\psi_i\}$ *and* $\mathsf{Score}(\Omega, \psi_i) > |\Omega|/2$*, then* X *is ground to the textentry* ψ_i*, otherwise* X *is ground to the textentry* NoMajority*.*
2. *For* $\texttt{FirstPastThePost}(\Omega, \texttt{X})$*, if* $\mathsf{MaxVote}(\Omega) = \{\psi_i\}$*, then* X *is ground to the textentry* ψ_i*, otherwise* X *is ground to the textentry* NoFirstPastThePost*.*
3. *For* $\texttt{PopularSharing}(\Omega, \texttt{X})$*,* X *is ground to the textentry* ψ'_1 or...or ψ'_k *where* $1 \leq k \leq n$ *and* $\mathsf{MaxVote}(\Omega) = \{\psi'_1, ..., \psi'_k\}$*.*
4. *For* $\texttt{ThresholdVoting}(\Omega, \tau, \texttt{X})$*,* X *is ground to the textentry* ψ'_1 or...or ψ'_k *where* $1 \leq k \leq n$*,* $0 \leq \tau \leq 1$ *and* $\mathsf{ThresholdVote}(\Omega, \tau) = \{\psi'_1, ..., \psi'_k\}$*, otherwise* X *is ground to the textentry* NoTextentryhasT%ofthevotes*, where* T *is* $\tau \times 100$*. Note that a disjunction of textentries can be used to ground* X *only if* $\tau < 0.5$*.*

Condition literal with logical variable	Instantiation of logical variable
Majority([rain, rain, snow], X)	rain
Majority([rain, rain, snow, showers, sleet], X)	NoMajority
Majority([rain, rain, snow, snow], X)	NoMajority
FirstPastThePost([rain, rain, snow], X)	rain
FirstPastThePost([rain, rain, snow, showers, sleet], X)	rain
FirstPastThePost([rain, rain, snow, snow], X)	NoFirstPastThePost
PopularSharing([rain, rain, snow], X)	rain
PopularSharing([rain, rain, snow, showers, sleet], X)	rain
PopularSharing([rain, rain, snow, snow], X)	rain or snow

Table 1: Examples of aggregation predicates (a type of condition literal) with the instantiation of the logical variables X.

The constant symbols NoMajority, NoFirstPastThePost, *and* NoTextentryhasT%ofthevotes *are types of null value.*

Our Majority predicate may be described as capturing absolute majority voting and our FirstPastThePast may be described as capturing relative majority voting.

Note that an additional degree of flexibility can be introduced into the merging process by adopting different thresholds for different contexts (such as different report elements).

Example 4.8 *In the following condition,* X *is a logical variable, and the* ThresholdVoting *predicate selects the most frequently occurring textentry from the* weatherreport/today *branch, provided that the frequency is greater than 0.75.*

$$\texttt{ThresholdVoting}(\texttt{//weatherreport/today}, 0.75, \texttt{X})$$

If //weatherreport/today *is ground as* [rain, rain, rain, snow, rain, rain, rain]*, the query sent to the Prolog knowledgebase is the following:*

$$\texttt{ThresholdVoting}([\texttt{rain}, \texttt{rain}, \texttt{rain}, \texttt{snow}, \texttt{rain}, \texttt{rain}, \texttt{rain}], 0.75, \texttt{X})$$

The grounding for X *returned by the Prolog knowledgebase is* rain *because it is the most frequently occurring textentry and that frequency is 0.85. If the condition remains the same, but* //weatherreport/today *is ground as* [rain, rain, rain, snow, sun, snow, showers, snow]*, the grounding for* X *returned by the Prolog knowledgebase is* NoTermhas75%ofthevotes *because no term occurs with a frequency equal to, or greater than, that specified. If, however, the specified frequency is dropped to 0.3, the same grounding for* //weatherreport/today *results in* X *being ground as* rain or snow *because both textentries occur with a frequency of 0.375.*

So far with voting, we have merged textentries with no consideration of background knowledge. For examples, see Table 1. However, we can introduce background knowledge. A simple development is just to consider preferred synonyms. In this way, we can take a list of textentries Ω, find the preferred synonyms for each item in Ω, the result of which is denoted Ω', and then use the existing simple voting predicates given in Definition 4.10. We formalize this in the next section where we use $\texttt{PreferredTerms}(\Omega, \Omega')$ as a relation in the Prolog knowledgebase.

Definition 4.11 *Let* Ω *be a list of textentries. The transferred voting predicates are defined as follows, where* X *and* Y *are logical variables to be ground by the Prolog knowledgebase.*

1. *For* $\texttt{Majority2}(\Omega, \texttt{X})$*, the grounding for* $\texttt{X}$ *is obtained by the following query*

$$\texttt{PreferredTerms}(\Omega, \texttt{Y}) \wedge \texttt{Majority}(\texttt{Y}, \texttt{X})$$

2. *For* $\texttt{FirstPastThePost2}(\Omega, \texttt{X})$*, the grounding for* $\texttt{X}$ *is obtained by the following query*

$$\texttt{PreferredTerms}(\Omega, \texttt{Y}) \wedge \texttt{FirstPastThePost}(\texttt{Y}, \texttt{X})$$

3. *For* $\texttt{PopularSharing2}(\Omega, \texttt{X})$*, the grounding for* $\texttt{X}$ *is obtained by the following query*

$$\texttt{PreferredTerms}(\Omega, \texttt{Y}) \wedge \texttt{PopularSharing}(\texttt{Y}, \texttt{X})$$

4. *For* $\texttt{ThresholdVoting2}(\Omega, \tau, \texttt{X})$*, the grounding for* $\texttt{X}$ *is obtained by the following query*

$$\texttt{PreferredTerms}(\Omega, \texttt{Y}) \wedge \texttt{ThresholdVoting}(\texttt{Y}, \tau, \texttt{X})$$

Example 4.9 *Suppose the following can be inferred from the Prolog knowledgebase.*

$$\texttt{PreferredTerms}([\texttt{rain}, \texttt{showers}, \texttt{rainy}, \texttt{snow}], [\texttt{rain}, \texttt{rain}, \texttt{rain}, \texttt{snow}])$$

Let Ω *be* $[\texttt{rain}, \texttt{showers}, \texttt{rainy}, \texttt{snow}]$*. So for* $\texttt{PreferredTerms}(\Omega, \texttt{X})$*,* $[\texttt{rain}, \texttt{rain}, \texttt{rain}, \texttt{snow}]$ *is the grounding for* $\texttt{X}$*. Hence for the condition literal* $\texttt{Majority}([\texttt{rain}, \texttt{rain}, \texttt{rain}, \texttt{snow}], \texttt{X})$*,* $\texttt{X}$ *is ground to* $\texttt{rain}$*. Therefore, for the condition literal* $\texttt{Majority2}([\texttt{rain}, \texttt{showers}, \texttt{rainy}, \texttt{snow}], \texttt{X})$*,* $\texttt{X}$ *is ground to* $\texttt{rain}$*.*

The condition literals given in Definitions 4.10 and 4.11 can be used in fusion rules with the corresponding definitions in the Prolog knowledgebase. We can show these aggregation predicates are equivalent to some of the aggregation predicates defined in voting theory (social choice theory) (see for example [Kel88]). Variations on these definitions may allow formalization of aggregation predicates that adhere to the Alchourron-Gardenfors-Makinson axioms [AGM85].

4.5 Aggregation by weighted voting

A further enhancement of voting can include the weighting of sources to allow weighted voting [Lin96]. All of the voting functions we have defined can be reformulated so as to accomodate this refinement. This is done by attaching a numerical value between 0 and 1 to each source and the source/weighting pairings are entered into the knowledgebase. The numerical value is then used as the number of votes cast by each report in favour of its textentry. In the cases of majority and threshold weighted voting, care must be taken to add up all of the weighted votes to determine how many votes are in play, so that we can in turn determine how many votes would constitute a majority or how many votes would be needed to meet a specified threshold. Furthermore, as was the case with aggregation using preferences over sources, care must also be taken with the treatment of $\texttt{null}$s. If a textentry for a report is $\texttt{null}$, the corresponding source textentry from the list of sources must also be removed before the votes are calculated using the weights.

Definition 4.12 *Let* Ω *be a list of textentries, let* Π *be a list of textentries denoting sources, let* π_i *be the source of* ψ_i*, let* λ *be a weighting function from sources to [0,1], and let* $\tau \in [0, 1]$ *be a threshold.*

$$\mathsf{WeightedScore}(\Pi, \lambda, \psi) = \sum_{\pi \in \Pi \textit{such that } \pi \textit{is a source of } \psi} \lambda(\pi)$$

$$\mathsf{WeightedScoreSum}(\Omega, \Pi, \lambda) = \sum_{\psi \in \mathsf{Set}(\Omega)} \mathsf{WeightedScore}(\Pi, \lambda, \psi)$$

$$\mathsf{WeightedMaxVote}(\Omega, \Pi, \lambda) = \{\psi \in \mathsf{Set}(\Omega) \mid \textit{for all } \psi' \in \mathsf{Set}(\Omega)\ \mathsf{WeightedScore}(\Pi, \lambda, \psi) \geq \mathsf{WeightedScore}(\Pi, \lambda, \psi')\}$$

$$\mathsf{WeightedThresholdVote}(\Omega, \Pi, \lambda, \tau) = \{\psi \in \mathsf{WeightedMaxVote}(\Omega, \Pi, \lambda) \mid (\mathsf{WeightedScore}(\Pi, \lambda, \psi)/\mathsf{WeightedScoreSum}(\Omega, \Pi, \lambda)) \geq \tau\}$$

Example 4.10 *Let* $\Omega = [\texttt{sun}, \texttt{rain}, \texttt{rain}, \texttt{sun}]$ *and let* $\Pi = [\texttt{BBC LDN}, \texttt{BBCi}, \texttt{CNN}, \texttt{The Weather Channel}]$. *Also let* $\lambda(\texttt{BBC LDN}) = 0.3$, $\lambda(\texttt{BBCi}) = 0.5$, $\lambda(\texttt{CNN}) = 0.4$, $\lambda(\texttt{The Weather Channel}) = 0.9$, *and let* $\tau = 0.33$.

$$\mathsf{WeightedMaxVote}(\Omega, \Pi, \lambda) = \{\texttt{sun}\}$$
$$\mathsf{WeightedThresholdVote}(\Omega, \Pi, \lambda, \tau) = \{\texttt{sun}\}$$

Definition 4.13 *Let* Ω *be a list of textentries to be aggregated and let* Π *be a list of textentries denoting sources such that* ψ_i *in* Ω *is from source* π_i *in* Π. *Let* λ *be a constant symbol denoting the name of a function from sources to* $[0, 1]$. *This gives the following schemafree atoms, as conditions, where* X *is a logical variable to be ground by the Prolog knowledgebase:*

1. *For* $\texttt{WeightedMajority}(\Omega, \Pi, \lambda, \texttt{X})$, *if* $\mathsf{WeightedMaxVote}(\Omega, \Pi, \lambda) = \{\psi_i\}$ *and* $\mathsf{WeightedScore}(\Pi, \lambda, \psi_i) > |\Omega|/2$, *then* X *is ground to the textentry* ψ_i, *otherwise* X *is ground to the textentry* NoWeightedMajority.
2. *For* $\texttt{WeightedFirstPastThePost}(\Omega, \Pi, \lambda, \texttt{X})$, *if* $\mathsf{WeightedMaxVote}(\Omega, \Pi, \lambda) = \{\psi_i\}$, *then* X *is ground to the textentry* ψ_i, *otherwise* X *is ground to the textentry* NoWeightedFirstPastThePost.
3. *For* $\texttt{WeightedPopularSharing}(\Omega, \Pi, \lambda, \texttt{X})$, X *is ground to the textentry* ψ'_1 or...or ψ'_k *where* $1 \leq k \leq |\Omega|$ *and* $\mathsf{WeightedMaxVote}(\Omega, \Pi, \lambda) = \{\psi'_1, ..., \psi'_k\}$.
4. *For* $\texttt{WeightedThresholdVoting}(\Omega, \Pi, \lambda, \tau, \texttt{X})$, X *is ground to the textentry* ψ'_1 or...or ψ'_k *where* $1 \leq k \leq |\Omega|$, $0 \leq \tau \leq 1$ *and* $\mathsf{WeightedThresholdVote}(\Omega, \Pi, \lambda, \tau) = \{\psi'_1, ..., \psi'_k\}$, *otherwise* X *is ground to the textentry* NoTextentryhasT%oftheWeightedvotes, *where* T *is* $\tau \times 100$. *Note that a disjunction of textentries can be used to ground* X *only if* $\tau < 0.5$.

The constant symbols NoTextentryhasT%oftheWeightedvotes, NoWeightedFirstPastThePost, *and* NoWeightedMajority *are types of null value.*

This approach to weighted voting differs from a standard understanding of the term. As it is often understood, weighted voting is an arrangement in which voters, or $players$, control unequal numbers of votes and decisions are made by forming coalitions where the total of votes at the disposal of the coalition is equal to or greater than an agreed threshold known as the $quota$. Probably the best-known example is the electoral college employed in U.S. general elections, where different states command different numbers of votes according to the size of their populations (California controlling the largest number at 54 down to Wyoming with only 3), 270 votes being the quota required of a candidate to be elected president. The standard notation for such a system is $[q : w_1, ..., w_n]$, where q is the quota, n is the number of players, and w_i is the number of votes controlled by the ith voter.

The above standard approach to weighted voting is, however, unsuitable as an aggregation function for merging news reports. This is because of two kinds of uncertainty concerning the voters or players in a NewsFusion System. Unlike, say, a U.S. general election, where it is known in advance how many electoral college votes are in play, in the case of the NewsFusion System it is not known either how many reports are to be merged, or from which sources those reports will come. This means that we do not know in advance of actually merging a set of reports either how many voters or players there will be, nor do we know how many votes those players will command. And this in turn means that it is not possible to determine a quota that could either be used as a fact in the background knowledge, or incorporated into the rules set. This is why a different approach to weighted voting was harnessed instead.

Example 4.11 *In the following condition,* X *is a logical variable.*

$$\texttt{WeightedFirstPastThePost}(\texttt{//weatherreport/source}, \texttt{//weatherreport/today}, \texttt{Weight}, \texttt{X})$$

Date	Report textentry (BBCi, BBC LDN, CNN, Weather Channel)	Aggregation Function			
		Popular Sharing2	Preferred Source	Least Upper Bound	Syntactic Generalization
3/12/02	showers, showers, cloudy, cloudy	showers or cloud	cloudy	no least upper bound	showers or cloudy
4/12/02	rain, rain, showers, showers	rain or showers	showers	rain	rain or showers
5/12/02	cloudy, cloudy, sunny, mostly cloudy	cloud	mostly cloudy	no least upper bound	cloudy, sunny or mostly cloudy
6/12/02	cloudy, cloudy, cloudy, cloudy	cloud	cloud	cloud	cloud
9/12/02	sunny, sunny, sunny, partly cloudy	sun	partly cloudy	no least upper bound	sunny or partly cloudy
10/12/02	cloudy, cloudy, snow, rain/snow	cloud	rain/snow	no least upper bound	cloudy, snow or rain/snow
11/12/02	sunny, sunny, cloudy, mostly cloudy	sun or cloud	mostly cloudy	no least upper bound	sunny, cloudy or mostly cloudy
24/3/03	sunny, sunny, p/cloudy, fair	sun	fair	no least upper bound	sunny, p/cloudy or fair
26/3/03	cloudy, cloudy, mostly cloudy, sunny	cloud	sunny	no least upper bound	cloudy, mostly cloudy or sunny
27/3/03	sunny, sunny, cloudy, mostly sunny	sun	mostly sunny	no least upper bound	sunny, cloudy or mostly sunny

Table 2: The results of four approaches to aggregating the textentry for `today`. The textentries are real data drawn from the four sources on the indicated dates. In the case of the preference aggregation function, the linear preference ordering was purely for the sake of illustration, and so is completely arbitrary. That ordering is (with most preferred first): The Weather Channel, BBCi, BBC LDN, CNN. The reports for December 4th best illustrate the differences between the approaches: voting by popular sharing produces a tie, so the textentry `rain or showers` is selected. Because the Weather Channel was nominated as the preferred source, resolving the conflict in this way results in the textentry `showers` being chosen. And because (arguably) rain is a more general concept than showers, `rain` is the textentry selected by using the least upper bound. The fourth approach, syntactic generalization, in this case produces the same result, `rain or showers`, as the first approach. By contrast, the reports for December 6th do not conflict, hence none of the four approaches to resolving conflicts is required, the preferred term (`cloud`) for the equivalence class to which all the textentries belong being used instead.

Let the weightings assigned to the sources be: $\texttt{Weight}(\texttt{BBCi}) = 0.5$, $\texttt{Weight}(\texttt{CNN}) = 0.2$, $\texttt{Weight}(\texttt{Sky}) = 0.2$. *If* `//weatherreport/source` *is ground as* $[\texttt{BBCi}, \texttt{CNN}, \texttt{Sky}]$, *and* `//weatherreport/today` *is ground as* $[\texttt{rain}, \texttt{sun}, \texttt{sun}]$, *the query sent to the Prolog knowledgebase is the following:*

$$\texttt{WeightedFirstPastThePost}([\texttt{BBCi}, \texttt{CNN}, \texttt{Sky}], [\texttt{rain}, \texttt{sun}, \texttt{sun}], \texttt{X})$$

The grounding for `X` *returned by the Prolog knowledgebase is* `rain` *because, even though two of the three reports have the textentry* `sun`*, the single report that has the textentry* `rain` *(*`BBCi`*) commands more weight than the other two reports together.*

The weighted voting given in Definition 4.13 is based on a simple and well-established extension of the voting strategies discussed in the previous section. It can also be viewed as a special case of integration of weighted knowledgebases [Lin96].

4.6 Aggregation by centre of gravity

Consider a situation where one report says `rain`, one says `snow`, and one says `sun`. Further suppose you are about to leave your home and you want to know how to dress appropriately. Intuitively, the reports saying `rain` and `snow` seem to support each other insofar as they both suggest that you should take some waterproof clothing. The reason that `rain` and `snow` seem to support each other is that the semantic distance between them is much less than between either of them with `sun`. In the following, we formalize this as aggregation by centre of gravity. First, we define some subsidiary functions in Definition 4.14 and Definition 4.15, and then define the centre of gravity aggregation predicate in Definition 4.16.

Definition 4.14 *A function D from a set Λ to $[0,1]$ is a distance function iff (1) for all $x, y \in \Lambda$, $D(x,y) = D(y,x)$; (2) $D(x,x) = 0$ and (3) $D(x,z) \leq D(x,y) + D(y,z)$.*

Definition 4.15 *Let Ω be a set of textentries, and let $X \subseteq \mathsf{Set}(\Omega)$. Let D be a distance function for $\mathsf{Set}(\Omega)$.*

$$\mathsf{MajoritySet}(X) = \{Y \subset X \mid ((|\mathsf{Set}(\Omega)|/2) + 1) \geq |Y| \geq (|\mathsf{Set}(\Omega)|/2)\}$$

$$\mathsf{Cost}(X, D) = \mathsf{Max}(\{D(x,y) \mid x, y \in X\})$$

$$\mathsf{MinDistanceSet}(X, D) = \{Y \in \mathsf{MajoritySet}(X) \mid \textit{ for all } Z \in \mathsf{MajoritySet}(X)\ \mathsf{Cost}(Z, D) \geq \mathsf{Cost}(Y, D)\}$$

Example 4.12 *Let* $\Omega = [\mathtt{rain}, \mathtt{rain}, \mathtt{snow}, \mathtt{snow}, \mathtt{sun}, \mathtt{sun}, \mathtt{sun}]$.

$$\mathtt{D}(\mathtt{rain}, \mathtt{snow}) = 0.1 \qquad \mathtt{D}(\mathtt{rain}, \mathtt{sun}) = 0.6 \qquad \mathtt{D}(\mathtt{snow}, \mathtt{sun}) = 0.7$$
$$\mathsf{Cost}(\{\mathtt{rain}, \mathtt{snow}\}, \mathtt{D}) = 0.1 \quad \mathsf{Cost}(\{\mathtt{rain}, \mathtt{sun}\}, \mathtt{D}) = 0.6 \quad \mathsf{Cost}(\{\mathtt{snow}, \mathtt{sun}\}, \mathtt{D}) = 0.7$$

$$\mathsf{MajoritySet}(\mathsf{Set}(\Omega)) = \{\{\mathtt{rain}, \mathtt{snow}\}, \{\mathtt{snow}, \mathtt{sun}\}, \{\mathtt{sun}, \mathtt{rain}\}\}$$

$$\mathsf{MinDistanceSet}(\mathsf{Set}(\Omega, \mathtt{D})) = \{\{\mathtt{rain}, \mathtt{snow}\}\}$$

Definition 4.16 *Let Ω be a tuple of textentries, let* `D` *be a constant symbol referring to a distance function, and let* `X` *be a logical variable. If* $\mathsf{MinDistanceSet}(\Omega, \mathtt{D})$ *is a singleton set, then the aggregation predicate* $\mathtt{CentreOfGravity}(\Omega, \mathtt{D}, \mathtt{X})$ *is defined so that* `X` *is ground to $\psi_1 \vee \ldots \vee \psi_n$ where* $\mathsf{MinDistanceSet}(\Omega, \mathtt{D}) = \{\psi_1 \vee \ldots \vee \psi_n\}$ *otherwise* `X` *is ground to* `NoCentreOfGravity`*. The constant symbol* `NoCentreOfGravity` *is a type of null value.*

Example 4.13 *For* $\mathtt{CentreOfGravity}([\mathtt{rain}, \mathtt{rain}, \mathtt{snow}, \mathtt{snow}, \mathtt{sun}, \mathtt{sun}, \mathtt{sun}], \mathtt{D}, \mathtt{X})$*, the logical variable* `X` *is ground to* $\mathtt{rain} \vee \mathtt{snow}$*, where* `D` *is defined in Example 4.12.*

Aggregation by centre of gravity can be viewed as an adaptation of majority voting. Aggregation by centre of gravity requires more information in the form of a distance measure, and it only appears to be intuitive for some types of textentry.

4.7 Relationships between aggregation predicates

The aggregation predicates described in this paper are just indicative of the range of further aggregation predicates that we can capture as condition literals. As discussed above, many of them can be viewed as special cases of well established aggregation techniques such as disjunction, subsumption, and voting.

Whilst we have defined a number of aggregation predicates in this paper, in some restricted cases, some combinations of aggregation predicates give the same result:

- For the predicates $\texttt{Majority}(\Omega, \texttt{X})$ and $\texttt{Majority2}(\Omega, \texttt{Y})$, if $\texttt{PreferredTerms}(\Omega, \Omega)$ holds, then X and Y have the same grounding. We get analogous results for FirstPastThePost and FirstPastThePost2, for PopularSharing and PopularSharing2, and for ThresholdVoting and ThresholdVoting2.
- For the predicates $\texttt{Majority}(\Omega, \texttt{X})$ and $\texttt{WeightedMajority}(\Omega, \Pi, \lambda, \texttt{Y})$, if $\lambda(\pi) = 1$ holds for all $\pi \in \Pi$, then X and Y have the same grounding. We get analogous results for FirstPastThePost and WeightedFirstPastThePost, for PopularSharing and WeightedPopularSharing, and for ThresholdVoting and WeightedThresholdVoting.
- For the predicates $\texttt{PopularSharing}(\Omega, \texttt{X})$ and $\texttt{ThresholdVoting}(\Omega, \tau, \texttt{Y})$, if $\tau = 0$, and Y is not a null value, then X and Y have the same grounding.
- For the predicates $\texttt{PopularSharing}(\Omega, \texttt{X})$ and $\texttt{Disjunction}(\Omega, \texttt{Y})$, if $\mathsf{Score}(\Omega, \psi) = 1$ for all $\psi \in \mathsf{Set}(\Omega)$, then X and Y have the same grounding.
- For the predicates $\texttt{Majority}(\Omega, \texttt{X})$ and $\texttt{UsePreferredSource}(\Omega, \Pi, \geq, \texttt{Y})$, if $\mathsf{MaxVote}(\Omega) = \{\psi\}$, and $\mathsf{Score}(\Omega, \psi) > |\Omega|/2$, and $\mathsf{MostPreferredSources}(\geq, \Pi) = \{\pi\}$, and the source of ψ is π, then X and Y have the same grounding.

In general, the aggregation predicates in this paper give divergent behaviour in a number of respects. One kind of difference is with respect to null values obtained. Some are never ground to a null value (e.g Disjunction and Popular sharing). For a list of textentries Ω, when $\texttt{FirstPastThePost}(\Omega, \texttt{X})$ has X ground to NoFirstPastThePost, then $\texttt{Majority}(\Omega, \texttt{Y})$ has Y ground to NoMajority. The same kind of ranking is observed with the predicates $\texttt{FirstPastThePost2}(\Omega, \texttt{Y})$ and $\texttt{Majority2}(\Omega, \texttt{X})$.

Semantic generalization, preference of sources, weighted voting, and centre of gravity, all require meta-level information, whereas syntactic generalization and voting approaches do not. For many applications, it seems that meta-level knowledge is an important requirement for intuitive aggregation. Furthermore, for many applications it seems feasible to acquire meta-level knowledge such as hyponyms/hypernyms for semantic generalization, preferences over sources, weights for sources for weighted voting, and distances beween textentries for centre of gravity evaluations.

In practice, a rule engineer who is compiling a set of fusion rules for an application can therefore assess whether easy, but less informative, aggregations like syntactic generalizations are better than harder, but more informative, aggregations like centre of gravity. Though fusion rules can be defined so that if only a null value can be obtained from the harder aggregations, then an easier aggregation will be adopted. This is another example showing the advantage of context-dependency in the fusion rules approach.

5 Case study with weather reports

The weather reports case study is based on merging weather reports obtained from websites including www.bbc.co.uk and www.cnn.com. These weather reports are already in the form of structured news reports though not in the form of XML. They were marked up in XML by hand, though it would be straightforward to automate the construction of each XML file from the HTML file. The DTD for the weather reports contains 20 elements (see Figure 4).

In addition to the DTD, the textentries for the structured news reports were delineated as follows: **Cities** Major UK cities; **Accepted Sources** Main UK TV and radio broadcasters, and national newspapers; **Today** Today's weather conditions in the form of a word or phrase from a selection of 66 possibilities (e.g. cloudy, scattered showers, mostly wet, prolonged rain, etc); **Date** Acceptable date formats for day (e.g. 1, 1st, etc.), month (e.g. 1, Jan, January, etc.), year (e.g. 1998, 98, etc.), and these can be separated by ",", ". ", "- ", "/ ", "\ " and " "; **Visibility** Quantitative (e.g. 1 Mile, 1 km, etc.); **Visibility** Qualitative from the selection of good, fair, very good, excellent, unlimited, moderate, and poor; **Pressure** Absolute value (e.g. 1021.0

⟨!ELEMENT weatherreport(source, date, city, today, temp,
windspeed, relativehumidity, daylighthours,
pressure, visibility, visibilitydistance,
airpollutionindex, sunindex, sunindexrating,
dewpoint)⟩
⟨!ELEMENT temp(max, min)⟩
⟨!ELEMENT daylighthours(sunrise, sunset)⟩
⟨!ELEMENT pressure(absolutevalue, directionofchange)⟩
⟨!ELEMENT α(#PCDATA)⟩

Figure 4: A condensed version of the DTD for the weather reports where α is instantiated with any of source, date, city, today, max, min, windspeed, relativehumidity, sunrise, sunset, absolutevalue, directionofchange, visibility, airpollutionindex, sunindex, and dewpoint.

mb, etc.); **Pressure** Direction of change from the selection of rising, falling, and steady; **Max and Min Temperatures** (e.g. -1C, 23.9F, etc.); **Windspeed** (e.g. 15mph, 23kph, etc.); **Humidity** (e.g. 80%, etc.); **Time** (e.g. 10.13, 22.13, etc.); **SunindexRating** from the selection of low, high, and medium; **Sunindex** and **Airpollutionindex** 1, 2, 3, etc.

To merge information in reports of this kind we required a minimum of 31 fusion rules and a Prolog knowledgebase with 425 clauses. Various combinations of fusion rules were used to compare the use of different aggregation predicates on the same data. The NewsFusion System developed had no problem merging the information from ten of these reports at any one time, and there is no reason not to suppose that it could merge substantially more (say, 20-30) in a timely fashion.

Example 5.1 *A pair of rules from the case study dealing with the textentry for today's weather. Rule 18 says that if all the textentries for today's weather are from the same equivalence class, the preferred term from that class is selected and that text is added to the* today *tag in the merged report. If the first condition of rule 18 fails, then the first condition of rule 19 will succeed. The second condition of rule 18 constructs a list, substituting for each textentry for today's weather from the set of input reports the preferred term from the equivalence class to which it belongs. The disjunction of the most frequently occurring textentries is then chosen as the textentry in the merged report. The condition literal in rule 19 therefore uses* PopularSharing *as in Definition 4.11*

Rulecode is 18 (Status = optional)

EquivalentTerms(//weatherreport/today)
AND PreferredTerm(//weatherreport/today, X)
IMPLIES AddText(X, weatherreport/today)

Rulecode is 19 (Status = optional)

NotEquivalentTerms(//weatherreport/today)
AND PreferredTerms(//weatherreport/today, X)
AND PopularSharing(X, Y)
IMPLIES AddText(Y, weatherreport/today)

Many predicates in the knowledgebase fall into one of two categories: those that deal with qualitative textentries, and those that deal with quantitative textentries. Textentries for today, visibility, sunindexrating, and pressure (direction of change), are qualitative (rain, sun, moderate, etc.). These are tested to see if

Preferred term	Equivalence class
cloud	cloud, cloudy, mostly cloud, mostly cloudy, overcast, heavy cloud.
sun	sun, sunshine, sunny, mostly sunny, fair.
sun and cloud	scattered cloud, scattered clouds, partly cloudy, patchy cloud, p/cloudy, sunny spells, sunny intervals, patchy sunshine.
haze	haze, hazy.
showers	showers, showery, scattered showers, patchy rain, intermittant rain, drizzle, heavy showers, blustery showers.
storms	storm, storms, stormy, thunder, lightning, thunder storms, thundery storms, thunder and lightning, severe storms.
ice	ice, icy, frost, frosty, ground frost, black ice, hoar frost.
snow	snow, snow showers, sleet, snow flurries, snowy, snowing, heavy snow, drifting snow.
rain	rain, rainy, wet, precipitation, inclement, mostly wet, mostly rain, heavy rain, prolonged rain, downpour, downpours, frequent downpours, rain/snow.

Table 3: For the today tagname (representing today's conditions), the above are used as equivalence classes of textentry. If the entries for today from all input reports are in the same equivalence class, the preferred term is used in the merged report. If the entries are not all similar terms, each is converted into its preferred form, and the most frequently occurring term is used.

they are members of the same equivalence class. (See Table 3 for equivalence classes for the textentries of today.) If they are, the preferred term for that class is used to construct the merged report. If they are not, then the goal EquivalentTerms(...) will fail, but another goal, NotEquivalentTerms(...), will succeed in a subsequent rule (assuming that not all the textentries are ground with null). In this case each textentry is substituted by the preferred term for the equivalence class to which it belongs, and the disjunction of the most frequently occurring terms is used. (See Example 5.1 above.)

Where textentries denote quantitative values (temperature, windspeed, humidity, visibilitydistance, sunindex, airpollutionindex, time, and pressure) the unit suffix, if present, is stripped from the textentry and any required unit conversion is carried out (e.g kph to mph, degrees F to C, etc.). The maximum and minimum values are then found and, if they are within the accepted range (as determined by the knowledge engineer), those values are returned as part of a textentry including the unit suffix. If, for example, the query is SimilarWindSpeeds([10mph, 12mph, 11mph, 11mph, 24kph], X), the grounding for X would be 10 − 14.9136mph. On the other hand, if the minimum and maximum values are not within the accepted range, this goal would fail, and another goal, NotSimilarWindSpeeds(...), would succeed. In this case, the converted values would be ordered and, provided they form a majority, the greatest number within the accepted range would be selected. A similar interval textentry would then be constructed from this subset of the original values.

The accepted ranges used are as follows: visibilitydistance (quantitative) accepted range is 5 miles; pressure (absolute value) accepted range is 5mb; max and min temperatures accepted range is 5 degrees; daylighthours accepted range is 5 minutes; sunindex accepted range is 2; airpollution accepted range is 2; and dewpoint accepted range is 2.

Another group of predicates deals with dates. By checking for common seperator characters ("/", "-", etc.), dates are segmented into day, month, and year. Corresponding segments can then be compared to see if they are members of an equivalence class (Jan, January, 1, etc.). At present, date strings must be in the order day-month-year, although it may be possible to use context (e.g location or source) as a reason for judging a string such as "3/1/03" to be in UK or US format.

From this case study, it is clear that the weather reports DTD allows a diverse range of potential conflicts to arise in sets of input reports. Yet many of these conflicts can be addressed systematically, correctly, and efficiently by a modest sized combination of a fusion rulebase and a Prolog knowledgebase within a NewsFusion System. Furthermore, it is straightforward to encode both the rulesbase and the knowledgebase. It is also straightforward to extend the fusion rules and the knowledgebase to handle any extensions or changes to the range of input reports.

6 Discussion

Whilst we bill fusion rules as a logic-based scripting language for specifying how structured reports can be merged, we can also see them as a metalanguage for specifying how structured reports can be merged since the information in the input and output reports are represented by logical terms in the fusion rules. This logical clarity, together with the "CONDITION implies ACTION" form of the fusion rules that we have shown can be readily implemented, provides an ideal framework for harnessing knowledge representation and reasoning techniques for specifying condition literals in a potentially rich background knowledgebase.

Our logic-based approach differs from other logic-based approaches for handling inconsistent information such as belief revision theory (e.g. [Gar88, DP98, KM91, LS98]) and knowledgebase merging (e.g. [KP98, BKMS92]). These proposals are too simplistic in certain respects for handling news reports. Each of them has one or more of the following weaknesses: (1) One-dimensional preference ordering over sources of information — for news reports we require finer-grained preference orderings; (2) Primacy of updates in belief revision — for news reports, the newest reports are not necessarily the best reports; and (3) Weak merging based on a meet operator — this causes unnecessary loss of information. Furthermore, none of these proposals incorporate actions on inconsistency or context-dependent rules specifying the information that is to be incorporated in the merged information, nor do they offer a route for specifying how merged reports should be composed.

Other logic-based approaches to fusion of knowledge include the KRAFT system and the use of Belnap's four-valued logic. The KRAFT system uses constraints to check whether information from heterogeneous sources can be merged [PHG$^+$99, HG00]. If knowledge satisfies the constraints, then the knowledge can be used. Failure to satisfy a constraint can be viewed as an inconsistency, but there are no actions on inconsistency. In contrast, Belnap's four-valued logic uses the values "true", "false", "unknown" and "inconsistent" to label logical combinations of information (see for example [LSS00]). However, this approach does not provide actions in case of inconsistency.

Merging information is also an important topic in database systems. A number of proposals have been made for approaches based in schema integration (e.g. [PM98]), the use of global schema (e.g. [GM99]), and conceptual modelling for information integration based on description logics [CGL$^+$98b, CGL$^+$98a, FS99, PSB$^+$99, BCVB01]. These differ from our approach in that they do not seek an automated approach that uses domain knowledge for identifying and acting on inconsistencies. Heterogeneous and federated database systems are relevant, but they do not identify and act on inconsistency in a context-sensitive way [SL90, Mot96, CM01], though there is increasing interest in bringing domain knowledge into the process (e.g. [Cho98, SO99]). Also relevant is revision programming, a logic-based framework for describing and enforcing database constratints [MT98].

Our approach also goes beyond other technologies for handling heterogeneous information. The approach of wrappers offers a practical way of defining how heterogeneous information can be merged (see for example [HGNY97, Coh98, SA99]) and NLP/NLG offer some techniques for identifying information in input reports and generating a natural language summary [BME99]. However, in these approaches there is little consideration of problems of conflicts arising between sources. Our approach therefore goes beyond these in terms of formalizing reasoning with inconsistent information and using this to analyse the nature of the news report and for formalizing how we can act on inconsistency.

We have already stressed that the fusion rules approach is intended to harness and integrate existing proposals in knowledge representation and reasoning. In Section 4, we have demonstrated how a range of proposals for aggregation can be used in fusion rules. Another key area for harnessing existing proposals is in addressing semantic heterogeneity. Problems of semantic heterogeneity include different textentries used for the same thing, the same textentry used for different things, different constellations of tagnames used for the same thing, and the same constellation of tagnames used for different things. We have indicated how potentially background knowledge encoded in Prolog can be used for some of these problems. However, it is possible that technologies, based on description logics [BCM+03], for building, integrating, and querying ontologies can be harnessed more directly.

Acknowledgements

The authors wish to thank Weiru Liu for helpful feedback on work reported in this paper, and to the anonymous referees for suggestions for improvements to the paper.

References

[AGM85] C Alchourron, P Gardenfors, and D Makinson. On the logic of theory change: partial meet contraction and revision functions. *Journal of Symbolic Logic*, 50:510–530, 1985.

[BCM+03] Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[BCVB01] S Bergamaschi, S Castano, M Vincini, and D Beneventano. Semantic integration of heterogeneous information sources. *Data and Knowledge Engineering*, 36:215–249, 2001.

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

[BME99] R Barzilay, K McKeown, and M Elhadad. Information fusion in the context of multi-document summarization. In *Proceedings of 37th Annual Meeting of the Association of Computational Linguistics*, 1999.

[CGL+98a] D Calvanese, G De Giacomo, M Lenzerini, D Nardi, and R Rosati. Description logic framework for information integration. In *Proceedings of the 6th Conference on the Principles of Knowledge Representation and Reasoning (KR'98)*, pages 2–13. Morgan Kaufmann, 1998.

[CGL+98b] D Calvanese, G De Giacomo, M Lenzerini, D Nardi, and R Rosati. Source integration in data warehousing. In *Proceedings of the 9th International Workshop on Database and Expert Systems (DEXA'98)*, pages 192–197. IEEE Computer Society Press, 1998.

[Cho98] L Cholvy. Reasoning with data provided by federated databases. *Journal of Intelligent Information Systems*, 10:49–80, 1998.

[CM01] L Cholvy and S Moral. Merging databases: Problems and examples. *International Journal of Intelligent Systems*, 10:1193–1221, 2001.

[Coh98] W Cohen. A web-based information system that reasons with structured collections of text. In *Proceedings of Autonomous Agents'98*, 1998.

[DP98] D Dubois and H Prade, editors. *Handbook of Defeasible Resoning and Uncertainty Management Systems*, volume 3. Kluwer, 1998.

[FS99] E Franconi and U Sattler. A data warehouse conceptual data model for multidimensional aggregation. In S Gatziu, M Jeusfeld, M Staudt, and Y Vassiliou, editors, *Proceedings of the Workshop in Design and Management of Data Warehouses*, 1999.

[Gar88] P Gardenfors. *Knowledge in Flux*. MIT Press, 1988.

[GM99] G Grahne and A Mendelzon. Tableau techniques for querying information sources through global schemas. In *Proceedings of the 7th International Conference on Database Theory (ICDT'99)*, Lecture Notes in Computer Science. Springer, 1999.

[HG00] K Hui and P Gray. Developing finite domain constraints – a data model approach. In *Proceedings of Computation Logic 2000 Conference*, pages 448–462. Springer, 2000.

[HGNY97] J Hammer, H Garcia-Molina, S Nestorov, and R Yerneni. Template-based wrappers in the TSIMMIS system. In *Proceedings of ACM SIGMOD'97*. ACM, 1997.

[HL04a] A Hunter and W Liu. Fusion rules for merging uncertain information. Technical report, UCL Department of Computer Science, 2004.

[HL04b] A Hunter and W Liu. Logical reasoning with multiple granularities of uncertainty in semi-structured information. In *Proceedings of the IPMU Conference*, 2004.

[Hor98] I Horrocks. The FaCT system. In *Automated Reasoning with Analytic Tableaux and Related Methods: International Conference Tableaux'98*, volume 1397 of *Lecture Notes in Artificial Intelligence*, pages 307–312. Springer, 1998.

[HS03a] A Hunter and R Summerton. FusionRuleML: Representing and executing fusion rules. Technical report, UCL Department of Computer Science, 2003.

[HS03b] 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.

[Hun00a] A Hunter. Merging potentially inconsistent items of structured text. *Data and Knowledge Engineering*, 34:305–332, 2000.

[Hun00b] A Hunter. Reasoning with inconsistency in structured text. *Knowledge Engineering Review*, 15:317–337, 2000.

[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.

[Hun03] A Hunter. Evaluating the significance of inconsistency. In *Proceedings of the International Joint Conference on AI (IJCAI'03)*, pages 468–473, 2003.

[Hun04] A Hunter. How to act on inconsistent news: Ignore, resolve, or reject. Technical report, UCL Department of Computer Science, 2004.

[Kel88] J Kelly. *Social Choice Theory: An Introduction*. Springer, 1988.

[KM91] H Katsuno and A Mendelzon. On the difference between updating a knowledgebase and revising it. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Second International Conference (KR'91)*, pages 387–394. Morgan Kaufmann, 1991.

[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.

[Lin96] J Lin. Integration of weighted knowledge bases. *Artificial Intelligence*, 83:363–378, 1996.

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

[LSS00] Y Loyer, N Spyratos, and D Stamate. Integration of information in four-valued logics under non-uniform assumptions. In *Proceedings of 30th IEEE International Symposium on Multiple-Valued Logic (ISMVL2000)*. IEEE Press, 2000.

[Mot96] A Motro. Cooperative database systems. *International Journal of Intelligent Systems*, 11:717–732, 1996.

[MT98] V Marek and M Truszczynski. Revision programming. *Theoretical Computer Science*, 190:241–277, 1998.

[PHG+99] A Preece, K Hui, A Gray, P Marti, T Bench-Capon, D Jeans, and Z Cui. The KRAFT architecture for knowledge fusion and transformation. In *Expert Systems*. Springer, 1999.

[PM98] A Poulovassilis and P McBrien. A general formal framework for schema transformation. *Data and Knowledge Engineering*, 28:47–71, 1998.

[PSB+99] N Paton, R Stevens, P Baker, C Goble, S Bechhofer, and A Brass. Query processing in the TAMBIS bioinformatics source integration system. In *Proceedings of the 11th International Conference on Scientific and Statistical Databases*, 1999.

[SA99] A Sahuguet and F Azavant. Building light-weight wrappers for legacy web data-sources using W4F. In *Proceedings of the International Conference on Very Large Databases (VLDB'99)*, 1999.

[SL90] A Sheth and J Larson. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys*, 22:183–236, 1990.

[SO99] K Smith and L Obrst. Unpacking the semantics of source and usage to perform semantic reconciliation in large-scale information systems. In *ACM SIGMOD RECORD*, volume 28, pages 26–31, 1999.

[ZP96] E Zimanyi and A Pirotte. Imperfect information in relational databases. In *Uncertainty Management in Information Systems: From Needs to Solutions*, pages 35–88. Kluwer, 1996.