A Persistent Feature-Object Database for Intelligent Text Archive Systems

Takashi Ninomiya[1 2], Jun'ichi Tsujii[1 2], and Yusuke Miyao[2]

[1] CREST, Japan Science and Technology Agency
Kawaguchi Center Building, 4-1-8, Honcho, Kawaguchi-shi, Saitama
{ninomi, tsujii}@is.s.u-tokyo.ac.jp
[2] Department of Computer Science, University of Tokyo
Hongo 7-3-1, Bunkyo-ku, Tokyo
{ninomi, tsujii, yusuke}@is.s.u-tokyo.ac.jp

Abstract. This paper describes an intelligent text archive system in which typed feature structures are embedded. The aim of the system is to associate feature structures with regions in text, to make indexes for efficient retrieval, to allow users to specify both structure and proximity, and to enable inference on typed feature structures embedded in text. We propose a persistent mechanism for storing typed feature structures and the architecture of the text archive system.

1 Introduction

For the last decade, the main stream of NLP has focussed on studies for practical techniques, such as information retrieval (IR) or information extraction (IE) with widely accessible corpora. For example, the bag-of-words techniques significantly increased IR performance, and studies on IE proved that information can be extracted by matching hand-crafted templates with text. However, these techniques are difficult to use in sophisticated approaches, such as reasoning-based or linguistically-motivated approaches, because the bag-of-words techniques are not inherently structural, and matching templates is not well enough defined to be used for inference procedures. With the increasing importance of knowledge sharing and processing by XML, such as Semantic Web or question and answering (QA), intelligent procedures, like how to express a system of knowledge, how to identify user demands, and where to locate information embedded in text, are required to solve the problem.

The concern of this study is twofold: persistency of database and integration of knowledge management, text archives and NLP. In general, the more complex the framework, the less persistent its database becomes. For example, let us consider data objects in an object-oriented programming language where a class hierarchy is defined, and each object belongs to a class that defines its variables and functions. In such a case, all data become useless when their class definitions change. There are some solutions to this persistency problem, such as versioning. The most effective solution is to separate the data structures and semantics as much as possible, similar to XML or a text database where entries are separated by 'space' and 'new line'.

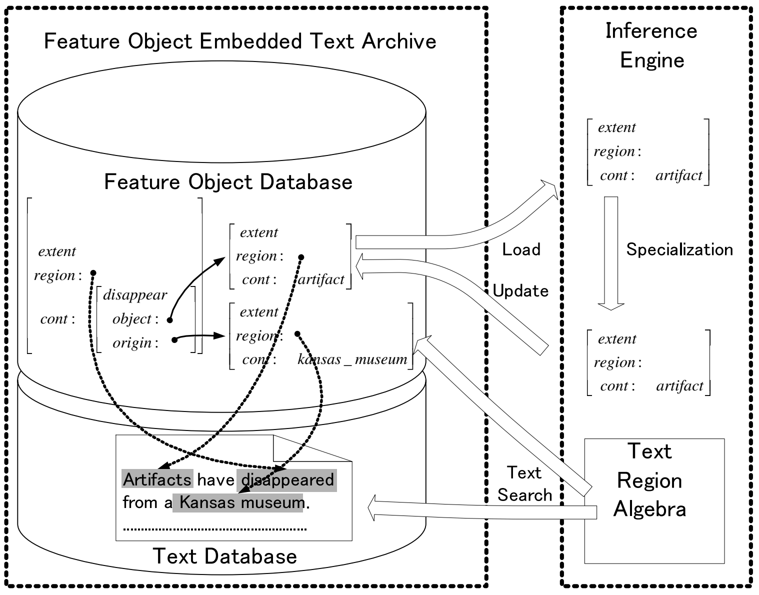


Fig. 1. Overall Foetea architecture

For integration, we assume that knowledge management includes the handling of heterogeneous, dynamic and structural knowledge sources. They are heterogeneous because they are developed and managed by independent groups, organizers, or persons, for different purposes and with different ontologies. They are dynamic because the type of ontology to be used for inference can easily be changed in the process of development and maintenance. Lastly, the knowledge sources should be able to express semantic structural relations like events, anaphora, predicate argument structures, or quasi-logical forms for intelligent IE or QA. We believe that a set of such semantic relations, including syntactic relations, gives more precise and complex text processing than the bag-of-words techniques when they are embedded in text.

In this paper, we introduce an intelligent text archive system, where data structures are persistent, and assigned regions of text, and users can specify both structure and proximity of text, and enable inference over structures embedded in text. We use typed feature structures [1] as the structures, because they are mathematically well-defined, and many linguistically-motivated grammar formalisms [2] are defined by feature structures. With a logic programming language for feature structures [3–5] and indexing scheme for feature structures [6–8], structural relations can efficiently be inferred by deduction, like a deductive database. Users can find feature structures embedded in text by structural specification or proximal specification using region algebra.

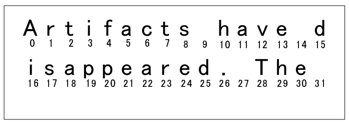


Fig. 2. Text

2 Feature-Object Embedded Text Archive System (Foetea)

Feature-object embedded text archive (Foetea) is an intelligent text archive system consisting of text, data structures and an inference engine. Figure 1 shows the overall architecture of Foetea. According to management policies, the architecture is separated into the following three layers.

Data Representation Layer The data representation layer consists of a *feature-object database* (explained later). The role of this layer is to represent data structures as labeled directed graphs, without any interpretation of the labels assigned to edges and nodes. Only a few operations, e.g., loading, storing and rewriting of data structures, are allowed.

Text Layer Regions in the text are annotated with data structures in the data representation layer. This stand-off approach enables us to annotate heterogeneous data structures and over-lapped regions. The role of this layer is to represent text and regions associated with data structures, without any interpretation of regions and text.

Conceptual Layer In the conceptual layer, data structures and text regions are interpreted and operated. Intuitively, this layer corresponds to the semantics for the data structures. Interpretation and inference of data structures is only allowed in this layer.

This separation makes the data structure persistent because a change in the interpretation of data structures does not impact on the data structures themselves. In general, the size of data structures is supposed to be very large. If the change of interpretation impacts on the data structures, only a small change in the interpretation mechanism will make all the data structures rubbish, or forces us to rewrite all the data structures, one by one.

As a system, Foetea has the following components: *feature-object database* for the data representation layer, *inference module* and *text region algebra module* for the conceptual layer, and *text*, *regions* and *extents* for the text layer.

Feature-Object Database A *feature-object database* is comprised of a set of feature structures and a set of *associations*. Let `Key` be a set of keys and `Assoc` be a set of associations defined in the database. An association is a pair (k, q), where q is a node of a feature structure and k is a *key*, such that k is unique in the set of associations. A *feature object* is a feature structure associated with a *key*. Figure 4 shows an example of a feature-object database. Intuitively, the database is a very

Containment Operators

Containing: $A \rhd B = G(\{c \mid \exists a \in A, b \in B.(a.s < b.s \wedge b.e < a.e \wedge c.s = a.s$
$\wedge c.e = a.e \wedge c.f = (a.f \rhd b.f))\})$

Not Containing: $A \not\rhd B = G(\{c \mid \exists a \in A.(c.s = a.s \wedge c.e = a.e \wedge c.f = (a.f \not\rhd \_)$
$\wedge \nexists b \in B.(a.s < b.s \wedge b.e < a.e))\})$

Contained In: $A \lhd B = G(\{c \mid \exists a \in A, b \in B.(b.s < a.s \wedge a.e < b.e \wedge c.s = a.s$
$\wedge c.e = a.e \wedge c.f = (a.f \lhd b.f))\})$

Not Contained In: $A \not\lhd B = G(\{c \mid \exists a \in A.(c.s = a.s \wedge c.e = a.e \wedge c.f = (a.f \lhd \_)$
$\wedge \nexists b \in B.(b.s < a.s \wedge a.e < b.e))\})$

Combination Operators

Both Of: $A \bigtriangleup B = G(\{c \mid c.s = min(a.s, b.s) \wedge c.e = max(a.e, b.e) \wedge c.f = (a.f \bigtriangleup b.f)\})$

One Of: $A \bigtriangledown B = G(\{c \mid c.s = a.s \wedge c.e = a.e \wedge c.f = (a.f \bigtriangledown \_)\}$
$\cup \{c \mid c.s = b.s \wedge c.e = b.e \wedge c.f = (\_ \bigtriangledown b.f)\})$

Ordering Operator

Followed By: $A \Diamond B = G(\{c \mid \exists a \in A, b \in B.(a.e < b.s \wedge c.s = a.s \wedge c.e = b.e$
$\wedge c.f = (a.f \Diamond b.f))\})$

Index Operator

Index: $I(A) = G(\{c \mid c \in \texttt{Extent} \wedge \exists k, q, F.((k, q) \in \texttt{Assoc} \wedge k = c.f$
$\wedge q$ is the root node of $F \wedge F \sqcup A$ is defined$)\})$

Shortest Matching Operator

GCL: $G(A) = \{a \mid a \in A \wedge \nexists b \in A.(b \neq a \wedge a.s < b.s \wedge b.e < a.e\}$

Fig. 3. Operators of text region algebra

large feature structure without a root node, and associations are global variables that represent sub-structures within the feature structure. This means that the database system allows structure-sharing between different feature objects.

Text, Region and Extent Text is a sequence of character assigned integers starting from zero up to the size of the text. Figure 2 shows an example of the text, “Articles have disappeared”. With integers assigned to the text, a *region* is defined by a pair of integers, a starting position and an ending position in the text. In the example, the region $(0, 9)$ corresponds to the character sequence “Artifacts”. An extent is a pair $\langle r, k \rangle$ where r is a region and k is a key, i.e., a feature structure is associated with a region. Let `Extent` denote the collection of extents in the database. We write $x.s$ for the start position of $x \in$ `Extent`, $x.e$ for the ending position of $x \in$ `Extent`, and $x.f$ for the key associated with $x \in$ `Extent`.

Inference Engine We suppose an inference engine is a system supporting a logic programming language for typed feature structures like PROLOG with typed feature structures as its predicate arguments instead of first order terms [3–5]. Users can retrieve feature objects from the feature-object database, specialize or modify them by inference, and insert or update them to the database. Users can also specify extents in the combination, ordering, and containment relations using text region algebra.

Text Region Algebra Text region algebra [9, 10] is usually used for searches on structured text annotated with tags, e.g., XML documents, to specify both proximity and structure. Text region algebra for tags can be transcribed into that for feature objects. Figure 3 shows the operators of text region algebra for feature objects, that are transcribed from [9]. The difference is that the index operator $I(A)$ retrieves extents that have feature objects unifiable with A. With text region algebra for feature objects, users can specify the relation of the feature objects' position in the text. For example, suppose all paragraphs in the text are annotated with feature objects [**paragraph**], and named entities are annotated by NE taggers. All paragraphs that include both named entities "Mars" and "rover" can be retrieved by a query algebra $I($ [**paragraph**]$) \triangleright (I($ [**mars**]$) \triangle I($ [**rover**]$))$. Text region algebra enables us to find feature objects that are difficult to find using unifiable relations, but are easily found by specifying containment, combination, and ordering relations.

Figure 4 shows an example of text, extents, associations and feature structures. In the example, syntactic and semantic analysis are annotated by semantic parsing. As Foetea allows structure-sharing between different feature objects, a feature object that represents a coreference in a long distance dependency can be represented. Structure-sharing tagged as $\boxed{29}$ represents a coreference of "Investors" and "they".

3 Persistent Database Model for Typed Feature Structures

3.1 Background

Feature Structure

A feature structure is a tuple $F = \langle Q, \bar{q}, \theta, \delta \rangle$, where Q is a set of a feature structure's nodes, $\bar{q}$ is the root node, $\theta(q)$ is a total node typing function that returns the type assigned to q, and $\delta(\pi, q)$ is a partial function that returns a node reached by following path π from q.

Type Constraint

Let $Intro(f)$ be the function that returns the most general type having feature f, and $Approp(f, \sigma)$ be the function that specifies the most general type of value that feature f can have for a node of type σ. A feature structure is said to be *well-typed* if, whenever $\delta(f, q)$ is defined, $Approp(f, \theta(q))$ is defined, and such that $Approp(f, \theta(q)) \sqsubseteq \theta(\delta(f, q))$. A feature structure is called *totally well typed* if it is well-typed and if $q \in Q$ and feature f are such that $Approp(f, \theta(q))$ is defined, then $\delta(f, q)$ is defined. Type constraint is the constraint defined for each type. The idea of type constraint is that a feature structure of type σ must satisfy the constraint defined for σ. Well-typedness and totally well-typedness are also special cases of type constraint.

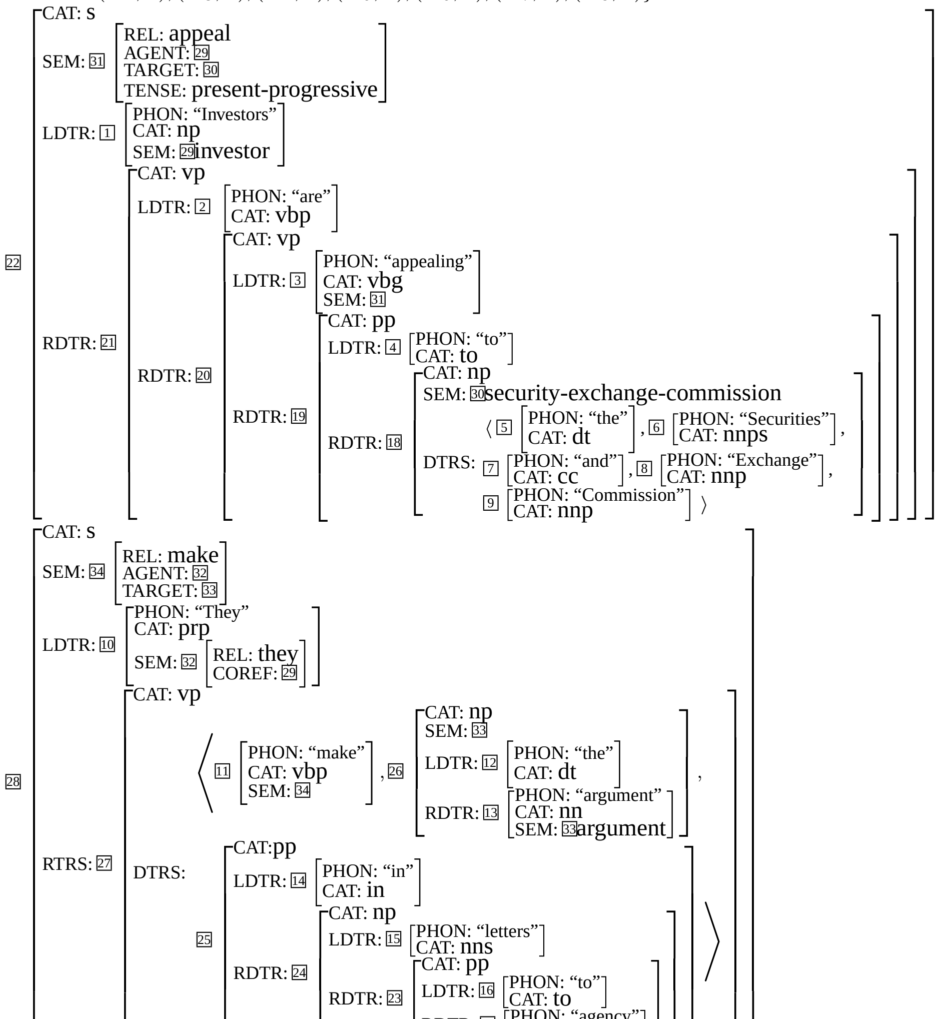
3.2 Interpretation of Data Structures in Conceptual Layer

Feature structures in the conceptual layer are totally well-typed and may have type constraints, but feature structures in the data representation layer are independent of their interpretation, such as type hierarchy, the order of features, appropriateness of

$_0$Investors$_9$ $_{10}$are$_{13}$ $_{14}$appealing$_{23}$ $_{24}$to$_{26}$ $_{27}$the$_{30}$ $_{31}$Securities$_{41}$ $_{42}$and$_{45}$ $_{46}$Exchange$_{54}$ $_{55}$Commission$_{65}$.
....
$_{353}$They$_{357}$ $_{358}$make$_{362}$ $_{363}$the$_{366}$ $_{367}$argument$_{375}$ $_{376}$in$_{378}$ $_{379}$letters$_{386}$ $_{387}$to$_{389}$ $_{390}$agency$_{406}$

$$\begin{aligned}
\texttt{Extent}_W &= \{((0,9),e_1),((10,13),e_2),((14,23),e_3),((24,26),e_4),((27,30),e_5),((31,41),e_6),\\
&\quad ((42,45),e_7),((46,54),e_8),((55,65),e_9),((353,357),e_{10}),((358,362),e_{11}),\\
&\quad ((363,366),e_{12}),((367,375),e_{13}),((376,378),e_{14}),((379,386),e_{15}),((387,389),e_{16}),\\
&\quad ((390,406),e_{17})\}\\
\texttt{Extent}_P &= \{((27,65),e_{18}),((24,65),e_{19}),((14,65),e_{20}),((10,65),e_{21}),((0,65),e_{22}),\\
&\quad ((387,406),e_{23}),((379,406),e_{24}),((376,406),e_{25}),((363,375),e_{26}),((358,406),e_{27}),\\
&\quad ((353,406),e_{28})\}\\
\texttt{Extent} &= \texttt{Extent}_W \cup \texttt{Extent}_P
\end{aligned}$$

$$\begin{aligned}
\texttt{Assoc} = \{&(e_1,\boxed{1}),(e_2,\boxed{2}),(e_3,\boxed{3}),(e_4,\boxed{4}),(e_5,\boxed{5}),(e_6,\boxed{6}),(e_7,\boxed{7}),(e_8,\boxed{8}),(e_9,\boxed{9}),(e_{10},\boxed{10}),(e_{11},\boxed{11}),\\
&(e_{12},\boxed{12}),(e_{13},\boxed{13}),(e_{14},\boxed{14}),(e_{15},\boxed{15}),(e_{16},\boxed{16}),(e_{17},\boxed{17}),(e_{18},\boxed{18}),(e_{19},\boxed{19}),(e_{20},\boxed{20}),(e_{21},\boxed{21}),\\
&(e_{22},\boxed{22}),(e_{23},\boxed{23}),(e_{24},\boxed{24}),(e_{25},\boxed{25}),(e_{26},\boxed{26}),(e_{27},\boxed{27}),(e_{28},\boxed{28})\}
\end{aligned}$$



CAT: s
SEM: 31 REL: appeal AGENT: 29 TARGET: 30 TENSE: present-progressive
LDTR: 1 PHON: "Investors" CAT: np SEM: 29 investor
RDTR: 21 CAT: vp
LDTR: 2 PHON: "are" CAT: vbp
RDTR: 20 CAT: vp
LDTR: 3 PHON: "appealing" CAT: vbg SEM: 31
RDTR: 19 CAT: pp
LDTR: 4 PHON: "to" CAT: to
RDTR: 18 CAT: np SEM: 30 security-exchange-commission
DTRS: ⟨ 5 PHON: "the" CAT: dt, 6 PHON: "Securities" CAT: nnps, 7 PHON: "and" CAT: cc, 8 PHON: "Exchange" CAT: nnp, 9 PHON: "Commission" CAT: nnp ⟩
22
CAT: s
SEM: 34 REL: make AGENT: 32 TARGET: 33
LDTR: 10 PHON: "They" CAT: prp SEM: 32 REL: they COREF: 29
RTRS: 27 CAT: vp
DTRS: ⟨ 11 PHON: "make" CAT: vbp SEM: 34, 26 CAT: np SEM: 33 LDTR: 12 PHON: "the" CAT: dt RDTR: 13 PHON: "argument" CAT: nn SEM: 33 argument,
25 CAT:pp LDTR: 14 PHON: "in" CAT: in RDTR: 24 CAT: np LDTR: 15 PHON: "letters" CAT: nns RDTR: 23 CAT: pp LDTR: 16 PHON: "to" CAT: to RDTR: 17 PHON: "agency" CAT: nn ⟩
28

Fig. 4. Foetea database

types and features, and any other type constraints. Any inference, such as unification, is not allowed in the data representation layer. The operations allowed by the database are *insertion*, *perfect loading*, *partial loading*, and *update*.

3.3 Operation

Insertion

Given a key $k \in \texttt{Key}$ and a feature structure G in the conceptual layer, G is inserted to the database without modifying any feature structures in the database. The association (k, q) for some q is rewritten to $(k, \bar{q'})$ where $\bar{q'}$ is the root node of the inserted feature structure.

Perfect Loading

Let F be a feature structure in the data representation layer. Perfect loading is the operation to calculate the most general feature structure G such that $F \sqsubseteq G$ and G satisfies the type constraints including totally well-typedness. If such a feature structure cannot be interpreted in the conceptual layer, this operation fails. Perfect loading retrieves all information in feature structures in the database if there are no inconsistencies.

Partial Loading

Let F be a feature structure $\langle Q_F, \bar{q}_F, \theta_F, \delta_F \rangle$ in the data representation layer. Partial loading is the operation to calculate the most general feature structure G such that G satisfies the type constraints including totally well-typedness, and $F' \sqsubseteq G$ where F' is a feature structure $\langle Q_F, \bar{q}_F, \theta_F, \delta_{F'} \rangle$ where $\delta_{F'}(f, q) = \delta_F(f, q)$ if $Approp(f, \theta_F(q))$ is defined, otherwise $\delta_{F'}(f, q)$ is undefined. Partial loading retrieves all information of types in F and tries to retrieve the edges of F as possible. For a node q and an edge labeled with feature f such that $\delta(f, q)$ is defined, the edge will not be retrieved if the type assigned to q is prohibited from having feature f. Figure 6 shows an example of feature objects. Suppose '**sem**' does not have the feature TENSE:, i.e., $Approp(\text{TENSE:}, \textbf{sem})$ is not defined. Perfect loading fails to retrieve F_1 because **sem** is not unifiable with any type that has the feature TENSE:. On the other hand, partial loading succeeds in retrieving F_1 by ignoring feature TENSE:. The result of the partial loading of F_1 becomes F_3.

Update

Update is a transaction that takes keys $\kappa \subset K$ and a description of the logic programming language. While the transaction is operating, feature structures associated with κ in the database are retrieved into the conceptual layer first. Next, a description of the programming language is executed. If the execution fails, the transaction finishes without changing the database. If it succeeds, the retrieved feature structures are updated to the database, and the transaction finishes.

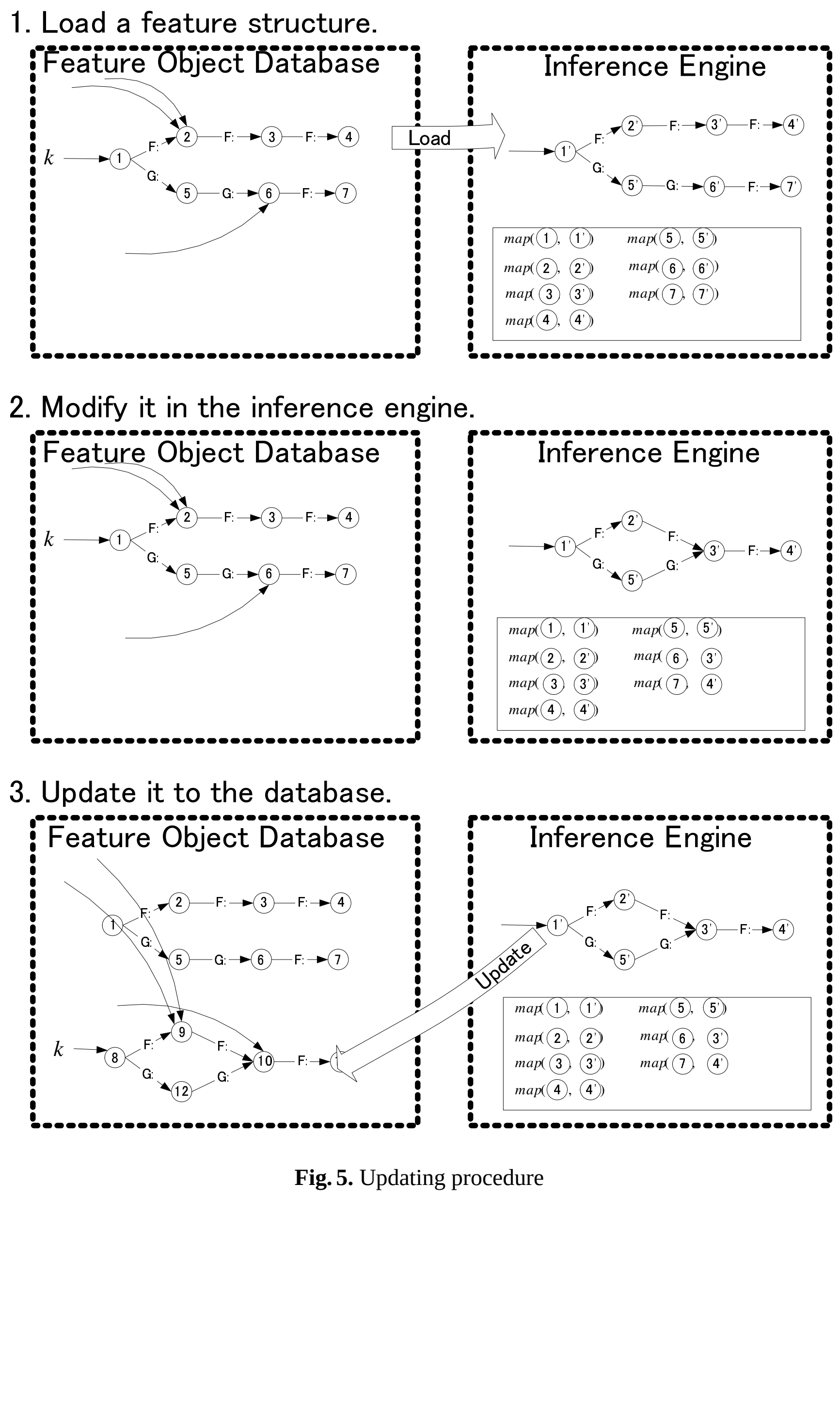


Fig. 5. Updating procedure

$$F_1 = \left[\begin{array}{l}\textbf{sem}\\ \text{REL: }\textbf{appeal}\\ \text{AGENT: }\textbf{investor}\\ \text{TARGET: }\textbf{security-exchange-commission}\\ \text{TENSE: }\textbf{present-progressive}\end{array}\right], F_2 = \left[\begin{array}{ll}\multicolumn{2}{l}{\textbf{sem}}\\ \multicolumn{2}{l}{\text{REL: }\textbf{make}}\\ \text{AGENT:} & \left[\begin{array}{l}\textbf{sem}\\ \text{REL:}\textbf{they}\\ \text{COREF: }\bot\end{array}\right]\\ \text{TARGET:} & \left[\begin{array}{l}\textbf{sem}\\ \text{REL:}\textbf{argument}\end{array}\right]\end{array}\right],$$

$$F_3 = \left[\begin{array}{l}\textbf{sem}\\ \text{REL: }\textbf{appeal}\\ \text{AGENT: }\textbf{investor}\\ \text{TARGET: }\textbf{security-exchange-commission}\end{array}\right],$$

$$F_1' = \left[\begin{array}{l}\textbf{sem}\\ \text{REL: }\textbf{appeal}\\ \text{AGENT: }\boxed{1}\textbf{investor}\\ \text{TARGET: }\textbf{security-exchange-commission}\end{array}\right], F_2' = \left[\begin{array}{ll}\multicolumn{2}{l}{\textbf{sem}}\\ \multicolumn{2}{l}{\text{REL: }\textbf{make}}\\ \text{AGENT:} & \left[\begin{array}{l}\textbf{sem}\\ \text{REL:}\textbf{they}\\ \text{COREF: }\boxed{1}\end{array}\right]\\ \text{TARGET:} & \left[\begin{array}{l}\textbf{sem}\\ \text{REL:}\textbf{argument}\end{array}\right]\end{array}\right]$$

Fig. 6. Feature objects

For simplicity, suppose that only one key $k \in \mathtt{Key}$ is given. Figure 5 shows the process of updating. Let F be a feature structure $\langle Q_F, \bar{q}_F, \theta_F, \delta_F \rangle$ such that there exists an association $(k, \bar{q}_F)$ in the data representation layer. Let $G (= \langle Q_G, \bar{q}_G, \theta_G, \delta_G \rangle)$ be a feature structure acquired by retrieving F from the database. For all $q_F \in Q_F$, $map(q_F, q_G)$ is defined if q_F in F is retrieved. When updating G, G is inserted to the database, and for all q_F and q_G such that $map(q_F, q_G)$, all edges that lead to q_F are rewritten and led to the inserted q_G.

Suppose that we have F_1 and F_2 depicted in Figure 6, and we want to unify the feature structure reached by following AGENT: in F_1 and the feature structure reached by following AGENT:COREF: in F_2. First, F_1 and F_2 are retrieved to the conceptual layer by partial loading, and then they are unified in the conceptual layer. Finally, they are updated to the data representation layer, and the result becomes F_1' and F_2'.

4 Conclusion

We proposed a persistent mechanism for storing typed feature structures and the overall architecture of the text archive system. The aim of the system is to associate typed feature structures with regions in text, to make indexes for efficient retrieval, to allow users to specify both structure and proximity, and to enable inference on feature structures embedded in the text. This persistent mechanism is achieved by separating the role of the typed feature structures into data representation and interpretation. Feature structures are preserved in the data representation layer that corresponds to data representation, and can be modified in the conceptual layer that corresponds to interpretation. Though only a few operations to feature structures are allowed in the data representation layer, we have exemplified that they are sufficient because modified feature structures in the conceptual layer can be updated, as if they were modified in the data representation layer.

References

1. Carpenter, B.: The Logic of Typed Feature Structures. Cambridge University Press, Cambridge, UK (1992)
2. Sag, I.A., Wasow, T.: Syntactic Theory : A Formal Introduction. CSLI Publications, Stanford, CA (1999)
3. Carpenter, B., Penn, G.: ALE: The attribute logic engine. user's guide (version 3.2.1) (2001)
4. Wintner, S., Francez, N.: Efficient implementation of unification-based grammars. Journal of Language and Computation **1** (1999) 53–92
5. Makino, T., Yoshida, M., Torisawa, K., Tsujii, J.: LiLFeS — towards a practical HPSG parser. In: Proc. of COLING-ACL-1998. (1998) 807–811
6. McCune, W.: Experiments with discrimination-tree indexing and path indexing for term retrieval. Automated Reasoning **18** (1992) 147–167
7. Sekar, R., Ramakrishnan, I.V., Voronkov, A.: Term indexing. In: Handbook of Automated Reasoning. Elsevier Science Publishers (2001) 1853–1964
8. Ninomiya, T., Makino, T., Tsujii, J.: An indexing scheme for typed feature structures. In: Proc. of COLING 2002. (2002) 1248–1252
9. Clarke, C.L.A., Cormack, G.V., Burkowski, F.J.: An algebra for structured text search and a framework for its implementation. The computer Journal **38** (1995) 43–56
10. Jaakkola, J., Kilpeläinen, P.: Nested text-region algebra. Technical Report C-1999-2, Department of Computer Science, University of Helsinki (1999)