U. Hedtstück, T. Bollinger, C.-R. Rollinger and R. Studer

Text Understanding in LILOG

Series A: General & Theoretical Papers
ISSN 1435-6473
Essen: LAUD 1988 (2nd ed. with divergent page numbering 2013)
Paper No. 214

Universität Duisburg-Essen
U. Hedtstück, T. Bollinger, C.-R. Rollinger and R. Studer
IBM Deutschland GmbH, Stuttgart, FRG (Germany)

Text Understanding in LILOG
0. Abstract

LILOG (Linguistic and LOGic methods) is a project for the investigation of linguistic and logic tools and methods for the computational understanding of texts. The basic data structure used in LILOG are directed acyclic graphs with labeled edges, provided by the Stuttgart Type Unification Formalism (STUF). The main mechanism is graph unification which is a generalization of the common first-order term unification known from theorem proving. For representing the different kinds of extralinguistic knowledge a representation language, L-LILOG, is currently being designed that is based on a many sorted first-order predicate calculus. We give a brief introduction to the STUF formalism and its operations and present a description of L-LILOG with special emphasis on the concept of sorts of real world objects. The handling of sorts and the relations between sorts are discussed in detail.

1. Introduction

LILOG (Linguistic and LOGic methods) is a project which is currently undertaken by IBM Germany at Stuttgart together with partners from the Universities of Hamburg, Osnabrück, Stuttgart, Tübingen, and from the Technical University of Aachen. The aim of LILOG is the investigation of linguistic and logic tools and methods for the computational understanding of texts. ‘Understanding’, in this context, refers to the construction of a semantic representation of a piece of text or a dialog statement, that is a (partial) model of the situation described in the text. This representation is held in a computer memory and is used by the knowledge processing component, e.g. for answering questions about the text. In 1987 a prototype was implemented able to understand a German text (taken from a tour guide) describing a hiking tour in the Alsace and to answer questions about the domain and the text.

Since the prototype will be the basis for further in text understanding of the diverse subprojects, we deckled for one basic data structure to represent both linguistic and common sense knowledge. By this we will be able to study the interaction of the syntactic, the semantic, and the contextual analysis of natural language texts with respect to anaphora resolution, presupposition handling and disambiguation.

This basic data structure used in the LILOG system are directed acyclic graphs (DAGs) with labeled edges, provided by the Stuttgart Type Unification Formalism (STUF). STUF is a powerful formalism for the declaration and specification of types. Principally, in STUF and object is represented by an attribute-value structure. The attributes have the function of restricting the object to a subtype of a given top level type. Characterizing
objects by type restrictions enables us to process them efficiently using mechanism which incorporate the treatment of taxonomical information. The general usage of STUF types motivated us to introduce an abstract data type (ADT) for STUF types as an extension of VM/PROLOG which is the basis for the implementation of the LILOG prototype. The main operations of the ADT offered to the programmer are graph unification, generalization, application and extraction. In addition there is a procedure testing the subsumption relation. The main mechanism is graph unification which is a generalization of the common first-order term unification known from theorem proving.

For representing the different kinds of extralinguistic knowledge which have to be handled in a natural language understanding system, a knowledge representation language, L-LILOG, is currently being designed that is based on a many-sorted first-order predicate calculus. It offers concepts for describing domain specific as well as domain independent world knowledge on a higher level than STUF types would allow. This knowledge is typically incomplete, vague and/or uncertain. Expressions of L-LILOG are compiled into STUF types (DAG’s) and interpreted by an inference machine using the open world assumption.

We choose STUF as the internal representation for the complete system because within computational linguistics STUF-like data structures were very successfully used for the development of unification grammars (cf. e.g. [Shieber et al. 1983]). Thus the basic operations for the inference process are the STUF operations, especially graph unification provided by the ADT extensions of VM/PROLOG, whereas the control flow of the inference machine was implemented using features of the programming language VM/PROLOG. In [Alt-Kać, Nasr 1986] and [Hedtstück 1988] inference procedures were introduced in which graph unification was used for the unification of type structures representing first-order terms. Principally, full many – representing first-order terms. P managed by the inference machine. At the current stage it incorporates concepts for the treatment of defaults and sort handling mechanisms. The inference strategy is a mixture of forward and backward chaining on the basis of an entrypoint concept. The internal representations of L-LILOG expressions as well as the inference machine are designed in a way that will allow diverse extensions, for instance, the incorporation of a truth maintenance mechanism and the processing of certainly values.

The paper is organized as follows. Section 2 gives a brief information to the STUF-formalism and its operations. Section 3 presents a description of the knowledge representation language L-LILOG with special emphasis on the concept of sorts of real world objects. Section 4 contains a description of the special STUF types representing L-LILOG concepts. In section 5 we discuss the handling of sorts and the relations between sorts in detail.

2. The Stuttgart Type Unification Formalism (STUF)

The Stuttgart Type Unification Formalism we developed to be a universal formalism for the representation of different kinds of objects which one is dealing with in a knowledge based system
for natural language processing. On the one hand, linguistic objects like lexical entries or grammar rules may be represented in STUF, on the other hand, it is possible to store the complete knowledge about the input text and the common sense knowledge as STUF structures in the knowledge base.

For the definition of the STUF expressions we mainly follow [König, Bouma, Uszkoreit 1988]. An introduction into STUF and an algebraic characterization of STUF may also be found in [Belerie, Pietat, Uszkoreit 1988]. For the STUF expressions we use the term ‘STUF type’ or shortly ‘type’ because they may be interpreted to represent sets of objects. Our characterization of STUF types in a special case of the characterization in [König, Bouma, Uszkoreit 1988] since we don’t consider disjunctive types.

Basically, the elements of STUF are attribute-value structures. There are two sorts of identifiers used in a type, the type names and the attribute names. Type names are divided into atomic type names and user defined names. User defined type names may be used in order to give a name to selected STUF types according to the user’s application. STUF has the two built-in type names \(*T*\) (TRUE or TOP) which denotes the type without any information and \(*F*\) (FALSE or FAIL) which may be thought of encoding inconsistent information.

We characterize STUF types by the following recursive definition.

A STUF type is either a type name or a list of attribute-value pairs, the values of which are again STUF types.

STUF types may also be characterized as partial functions from the set of attribute names into the set of STUF types where the type names are nullary functions.

Attribute-value structures may also be described as single-rooted directed graphs with labeled edges (DAGs) like in the PATR-II system (see for example [Shieber etal. 1983] and [Shieber 1986]). If STUF types are represented as DAGs then the edges are labeled by attribute names and the leaves are labeled by atomic type names.

According to the fact that in a graph representation two (or more) edges may point to the same node, in a representation by nested lists we must have the possibility to label the values of attributes by markers which we call ‘coreference markers’ (in [Alt-Kaç 1984] they are called ‘tags’). This allows us to write the value only once and then use the respective coreference marker as a pointer to it. Coreference markers are locally defined by the user.

Here, we make the restriction that there are no ‘cycles’ in a STUF type. This means that a coreference marker may not be used as the value of an attribute in the subtype which is marked by the coreference marker. In the representation of STUF types by directed graphs this restriction is equivalent to acyclicity.
The concept of atomic types in STUF may be motivated by the following considerations. In contrast to the user defined type names which may be used as names for distinguished STUF types atomic type names do not represent any attribute-value structure in the intended world. That means we restrict ourselves to a special part of the real world which is relevant for the actual application. But we want to have the possibility to extend this part of the real world. For example, if in the actual intended world “RED” is an atomic type name it could be necessary to extend the actual world by physical attributes for “RED” like “wavelength”. This extension is still under development.

In the next section we shall extend the set of atomic types by introducing complex expressions which may be used as atomic type names. This enables us to incorporate sort handling mechanisms into the unification procedure.

The main operation for managing STUF types is graph unification, which merges the information contained in two given types. If this may be done consistently the results is a new type which is a restriction of the two given types. In principle the unification procedure looks as follows.

1. The empty type unifies with any type.
2. Two atomic types unify if they are identical.
3. Two types each of which has at least one attribute-value pair unify if for any attribute occurring in both types the respective value types are unifiable.

In any other case the result of graph unification is *F*, the *FAIL* type. Thus, graph unification yields *F* if and only if during the course of the unification process two nonidentical atomic types or an atomic type and a non-atomic type which is not the empty type have to be unified.
3. The Knowledge Representation Language L-LILOG

As part of the LILOG project the knowledge representation language L-LILOG is currently being developed. The design of L-LILOG aims at providing a framework for representing the different kinds of domain knowledge which have to be handled in a natural language understanding system, i.e. concepts have to be offered for representing domain specific as well as domain independent real world knowledge.

More specifically, the following requirements have to be met by L-LILOG in order to be an adequate knowledge representation language within the LILOG project:

- Since a lot of real world knowledge is needed within a natural language understanding system, principles for structuring knowledge bases must be provided in order to be able to constrain the accessibility of knowledge to parts of the knowledge bases.

- According to the specific topics being investigated in different subprojects of LILOG the representation of temporal and spatial knowledge is of importance as well.

- Since real world knowledge is typically incomplete, vague and/or uncertain concepts for representing these types of knowledge have to be provided.

- In order to be able to formally specify the semantics of the language constructs as well as to formally define the semantics of the inference processes L-LILOG should be based on a sound theoretical basis.

Having these requirements in mind a kernel of knowledge representation language L-LILOG has been defined according to the following principles:

- L-LILOG is based on a many-sorted first-order predicate calculus.
- Sorts in L-LILOG are partially order-sorted.
- Attributes specifications define an internal structure for sorts.
- Within the sort hierarchy an inheritance mechanism for attributes is defined.
- Reference objects are used to represent information available about existing real world entities in an object-centered way.
Knowledge packets offer means of structuring knowledge bases in a hierarchical way.

Therefore, a L-LILOG knowledge base is currently defined as follows:

\[
\text{Knowledge\_Base::= Sort\_Declaration Predicate\_Declaration Reference\_Object\_Declaration Knowledge\_Packet\_Structure Knowledge\_Elements}
\]

The first two components currently represent the LILOG concept lexicon. It contains a sort or predicate specification for each concept known to the system.

Reference object declarations introduce internal identifiers for real world entities (see [Habel 1986]). Each identifier is associated with a sort as well as with a set of designations which can be used to refer to particular entity. Currently, only proper nouns may be used as designations. Of course, this limitation has to be eliminated in the future.

The knowledge packet structure defines a hierarchy of knowledge packets organizing conceptual as well as assertional knowledge in a modular way (see [Wachsmuth 1987] for details). Currently, only knowledge elements may be associated with knowledge packets.

In essence knowledge elements are defined by first-order formulas representing facts and rules expressing properties about real world entities.

Let us now consider two components of a L-LILOG knowledge base in some more detail.

3.1. **The Sort Concept in L-LILOG**

Aspects of cognitive adequacy, i.e. how to provide means for classifying real world objects, as well as efficiency aspects of inference methods (see e.g. [Walther 1987]) resulted in the decision to include a sort concept within the first-order calculus approach of L-LILOG.

Basically, a sort is defined by specifying its name, the set of supersorts as well as a set of attributes relating the given sort with an appropriate range sort. The supersort specifications introduce a partial ordering of sorts which is interpreted as a set-subset relationship in the models of L-LILOG. Within the sorts hierarchy an inheritance mechanism for attributes is defined.

E.g. the sort BUILDING might be defined as follows:

\[
\text{SORTED BUILDING}
\]

\[
\text{supersorts: STATIC\_OBJECT}
\]

\[
\text{(architect: HUMAN\_BEING,}
\]

\[
\text{architectural\_style: STYLE).}
\]
I.e. BUILDING is a subsort of the sort STATIC_OBJECT and is further specified by two attributes “architect” and “architectural_style” being range restricted by the sorts HUMAN_BEING and STYLE, respectively.

3.2. Knowledge Elements in L-LILOG

Basically, knowledge elements are specified by first-order formulas which are closed formulas, i.e. each variable has to be quantified. A partial syntax of L-LILOG formulas is specified below.

Atomic formulas are specified by either comparing terms, using a predicate which has been defined as part of the predicate declarations, or by using an attribute which has been defined within a sort specification. Subsequently, we shall discuss only the second alternative.

Complex formulas are built in the usual way by using “not”, “and”, “or”, and “impl”. When considering quantified formulas it is important to see that all quantified variables are associated with a sort restriction. Sort formulas provide means for restricting an arbitrary term to a given sort.

Up to now the inference machine of LILOG interprets all implications as rules, all other formulas as facts. Restrictions we have in our current version of L-LILOG are:

(i) function symbols are not allowed within terms, and

(ii) the handling of equality is not yet integrated.

Beyond that pure logical formalism the current kernel of L-LILOG includes a default concept for making facts and rules as default facts and rules as well as an entry point concept for rules for controlling the inference process. For reasons of clarity these concepts as well as the L-LILOG attributes and the inheritance mechanism, the knowledge packet structure, and the designations of reference objects are not discussed in the subsequent sections.
4. Encoding of L-LILOG in STUF

Any L-LILOG formula is translated by a complier into a STUF type representing an equivalent clausal normal form. A clausal normal form is a conjunction of clauses where a clause is a disjunction of literals. Here, a literal is a negated or unnegated atomic or sort formula. (See e.g. [Andrews 1986] for more details on normal forms in predicate logic.).

For the encoding of L-LILOG formulas in STUF types we introduce a “top level sort” concept in the following sense. Any STUF type may have a “type” attribute the value of which is an atomic type name denoting the top level sort of the object which is to be represented. Further attributes restrict the top level sort. Until now we don’t impose any structure on the top level sorts. They give us the possibility to do type checking during inference processes.

Any STUF type representing an L-LILOG formula has a “sign” attribute which indicates if the formula is negated or not. Its values are either “pos” or “neg”.

Atomic formulas are represented by STUF types whose “type” attributes have the value “pred_form”. The predicate name is represented by a “pred” attribute which has the predicate name as its value. In addition, there is an “args” attribute the value of which is the list of STUF types of the arguments of the predicate. Any of the arguments is characterized by a “role_name” attribute followed by a “vs” attribute which specifies the argument by a value of the form “vs(value, sort)”. Hereby “value” is a L-LILILOG term and “sort” is the L-LILOG sort name which belongs to it. (In the case of a reference object identifier the sort is not explicitly given by the L-LILOG formula like for variables. Thus, the complier has to look in the sort declaration part of the knowledge base in order to determine the appropriate sort.)

Because STUF offers (up to noe) no concept suitable for a sort handling process for L-LILOG sorts (comparable to Ait-Kaçi’s greatest common lower bound operation for the type symbols in [Ait-Kaçi 1984]) we had to introduce the complex vs-expressions as atomic types.

A special kind of atomic formulas are the sort formulas. By a sort formula we express the element relation between an object and a sort. This is done by using the sort as a unary predicate. The “type” attribute of a sort formula in STUF has the value “sort_form” and, analogously to the atomic formulas of the type “pred_form”, they have “pred”, “sign” and “args” attributes. The predicate name (the value of the “pred” attribute) is the name of the sort used in the corresponding element relation. The single argument has the role “element” followed by a “vs” attribute. Like for predicates its value is as “vs(value, sort)” expression.

For the representation of complex formulas the logical connectives “and”, “or”, “impl$\$” are encoded by the top level sorts “and_form”, “or_form” and “impl_form”, respectively. They have the attributes “lhs” (left hand side) and “rhs” (right hand side) the values of which are the STUF types for the two subformulas.
In order to handle the quantifiers in a correct and efficient way we use skolemization. The elements of the knowledge base are negatively which means that all existential quantified variables are replaced by Skolem functions. An L-LILOG formula representing a question is positively skolemized which means that any universally quantified variable is replaced by a Skolem function (see [Bibel 1986] for the usage of the terms positive resp. negative skolemization).

Skolemization allows us to omit the quantifiers in the STUF representations of L-LILOG formulas since all variables of a skolemized formula are either existential or universal. Skolem dependancies are encoded by additional attributes for the arguments of predicates. Their values are the variables the Skolem function depends on.

Example:

L-LILOG formula:

forall x:HIKER exists Y:RUCHSACK carry(agent=x,patient=y)

5. The Handling of Sorts

For the computation of greatest lower bounds (GLB) of L-LILOG sorts we need a special procedure based on a semilattice of sorts. Whereas the subsort relation imposes a partial ordering on the set of L-LILOG sorts in a natural way, in general it is difficult to determine if a given partially ordered set is a semilattice. Thus, we implemented a method for the computation of GLBs which doesn’t need a semillatice as the starting set of sorts but needs only a partially ordered set of sorts. It is based on an algorithm which imbeds any partial ordering into a lower semilattice. The algorithm is derived from the “Dedekind-MacNeille completion” described in [MacNeille 1936].

By the following example we want to demonstrate the principle mechanism of the Dedekind-MacNeille completion. Supposed the two sorts “BUILDING” and “SIGHT”
don’t have a unique GLB. This is the case if, for example, they both have the immediate subsorts “CASTLE”, “LOCK” and “RUIN” for which we may assume that they are pairwise incomparable. By the completion procedure a new sort is introduced as an immediate subsort of both “BUILDING” and “SIGHT”, being now their GLB. It gets exactly the subsorts “CASTLE”, “LOCK” and “RUIN”.

While the new sort is constructed by the algorithm, until now we have to choose the designation by hand. In the example we might choose the name “BUILDING_WORTH_SEEING” for the new sort.

In the inference machine graph unification is used in order to do resolution-like inference steps. For such a resolution step the STUF types representing two atomic formulas have to be unified by graph unification.

Graph unification may be best visualized with the aid of the graph representation of STUF types. It merges the attribute edges starting from the root and then merges the corresponding value graphs.

Recall that graph unification yields the *FAIL* type if two nonidentical atoms or an atom and an attribute-value structure have to be unified during the course of the unification process.

By introducing complex expressions like “vs(value,sort)” used in the position of atomic type names we incorporate special sort handling mechanisms into the process of graph unification. They are invoked if during graph unification two complex expressions have to be unified. We call this extension “atom unification”. Atom unification contains a GLB computation procedure as well as a procedure which tests the subsort relation and the element relation.

Let be given the two complex expressions vs(X1,SORT_1) and vs(X2,SORT_2). X1 and X2 may be variables or constants (reference object identifiers). Terms which are generated by skolemization are treated as constants. We denote the greatest lower bound of SORT_1 and SORT_2 by gib(SORT_1,SORT_2). Here, we say that a GLB exists if it is not the bottom element of the semilattice of sorts. Atom unification of vs-expressions is defined by the following rules:
By an example we want to illustrate when we need such an extension of the usual graph unification. Let the following L-LILOG formulas be in the knowledge base. For brevity we omit the roles of the arguments.

(1) \( r_1 \in \text{HUMAN} \)
(2) \( \text{hiking}(r_1) \)
(3) \( \forall y: \text{HUMAN} \ (\text{hiking}(y) \impl y \in \text{HIKER}) \)
(4) \( \forall z: \text{HIKER} \exists s: \text{RUCKSACK} \ (\text{hiking}(z) \impl \text{carry}(z,s)) \)

Question: \( \exists x: \text{RUCKSACK} \ \text{carry}(r_1,x)? \)

The sort of the reference object \( r_1 \) is given by (1) and is added to the STUF type representing the question.

The predicate “\( \text{carry}(\_,\_) \)” of the goal also appears as the predicate of the conclusion of implication (4). Thus, for the proof of the goal the inference machine first tries to make a typical resolution step. That causes the unification process of unity \( x \) and \( s \) under their sort restrictions which should succeed since both \( x \) and \( s \) are variables of the same sort. The corresponding complex expressions in STUF are \( \text{vs}(X,"\text{RUCKSACK}") \) and \( \text{vs}(S,"\text{RUCKSACK}") \) which are unified by atom unification.

In addition \( r_1 \) of the sort HUMAN has to be unified with \( z \) of the sort HIKER. That means the complex expressions \( \text{vs}(r_1,"\text{HUMAN}") \) and \( \text{vs}(Z,"\text{HIKER}") \) have to be unified by atom unification. Since from (1), (2) and (3) it is possible to derive that \( r_1 \) is an element of the sort HIKER, atom unification yields the result \( \text{vs}(r_1,"\text{HIKER}") \). Now the goal may be unified with the conclusion of (4) and it is easy to see that the premise of (4) may be proven by (2) which terminates the proof of the goal successfully.

Thus, in order to unify a variable and a constant we first test the suitable subsort relation. If this falls we start a separate proof process for the deduction of the element.
relation. Of course, we have to terminate this separate proof after a certain time since. In general, the premises for an element relation might be undecidable. If we don’t get a positive result of the separate proof then unification fails.

6. Future Work

We plan extend the expressive power of L-LILOG to higher order logics. The quantification of predicate variables, the modification of predicates, and the usage of propositions as arguments of e.g. epistemic operators “belief” are extensions necessary in order to provide an adequate semantic representation language for the linguistic research in progress. Since L-LILOG is developed to model open worlds we will investigate the integration of a at least three valued logic. In order to allow nonmonotonic changes of the world model caused by natural language input the development of a reasoning maintenance component for the inference machine will be investigated.

   Concepts for handling uncertainly and vagueness on the one hand and for describing reference objects by complex terms on the other hand are under development as well as the integration of equality reasoning.

7. Acknowledgements

We would like to thank Christoph Belerle, Karl Hans Bläsus, Gert Smolka, and Hans Uszkoreit for some stimulating discussion and comments.

   Josef Gemander and Volker Wenning helped to implement the knowledge processing component of the LILOG system. Jochen Dörre implemented the L-LILOG compiler.
8. References


Shieber, S. M. (1986): AN Introduction to Unification-Based Approaches to Grammar, CSLI Lecture Notes 4, Stanford, California.

