USING TYPED FEATURES LOGICS TO REPRESENT COMPLEX LAXICAL DATA

Lee Fedder

The Centre for Computational Linguistics
UMIST
lee@uk.ac.umist.ccl
Keywords: Lexicography, feature logic

Abstract

The ESPRIT project MULTILEX aims to produce standards and tools for the construction and exchange of large re-useable computational lexica. The proposed structure of the lexical entries is complex, and poses some particular representational problems. In this paper, we show how a Typed Feature Logic (TFL) based representation language provides an elegant formalism for recording this lexical structure. A prototype has been encoded using the POLYGLOSS TFS system version 4 running on a Macintosh LC.

Introduction

The MULTILEX project is developing standards and tools for the construction and exchange of large, re-useable, computational lexica. This, it is hoped, will prevent the repetition of effort currently involved in building lexica for new applications such as translation and word processing.

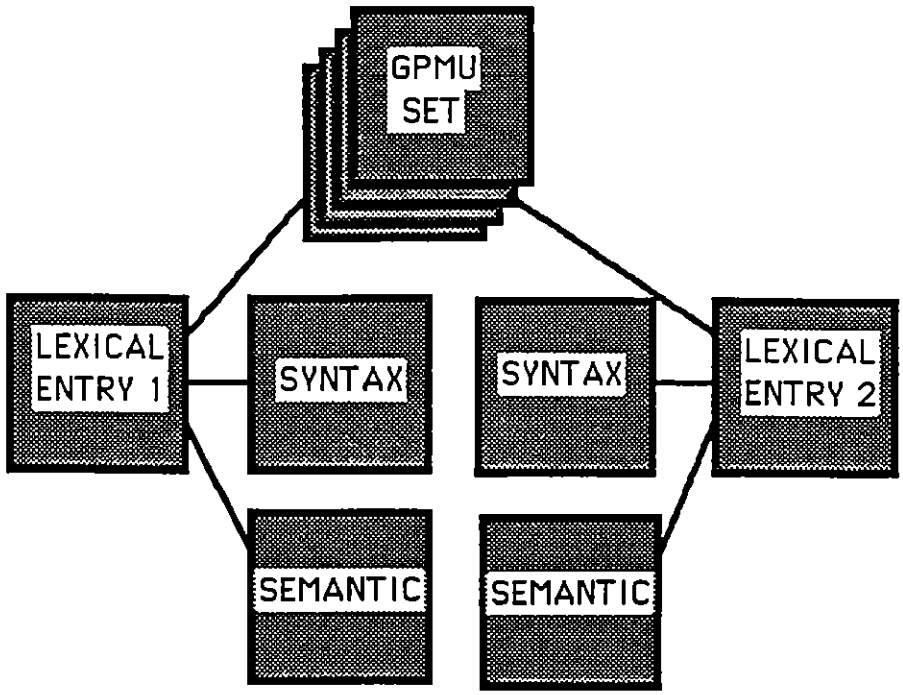

FIGURE 1 - LEXICAL STRUCTURE

In this paper, we show how this data structure can be recorded using a TFL based representation language, provided it has the right properties.

The TFL Based Representation

Typed Feature Logics are a recent development in computational linguistics. They represent a merging of the concepts of unification grammars, from linguistics, and ideas about object-oriented representation and data typing, from the world of computer science.

There are currently several implementations of TFL systems, each with different characteristics dictated by the intended application. We have the ACQUILEX system (Copestake 1991), the Carnegie-Mellon system (Carpenter 1990) and the POLYGLOSS system (Emele and Zajac 1991). The formalism we have proposed for MULTILEX is closely based on the POLYGLOSS system, and this has enabled us to prototype the MULTILEX lexical entries using the POLYGLOSS software. Where our formalism requirements differed from POLYGLOSS, we were able to simulate using existing POLYGLOSS facilities.

TFL offers a space-efficient, declarative, theory independent and linguistically relevant way of recording large amounts of lexical information, along with a lot of built-in data structure and consistency checks. These qualities make it attractive as a medium for building large lexical databases such as MULTILEX.

We begin by describing the basic elements of the TFL. These are all taken from the POLYGLOSS system, and use POLYGLOSS syntax. There are three data structures. Typed attribute-value matrices (AVMs), a hierarchy of types, and macros.

Typed AVMs

Each AVM is "typed". A type constrains the structure of the AVM, and can be atomic or complex. An atomic type is simply a list of values, and a complex type gives the attributes allowed in an AVM, along with type declarations for each of the values. An AVM may not contain attributes which do not appear in its type declaration. In the following example, the type "AGREEMENT" is complex, having the attributes "person" and "number", and types "PERS" and "NUM" are simple. The symbol "|" should be read as disjunction.

AGREEMENT = [person: PERS,number: NUM].
PERS = 1 | 2 | 3.
NUM = sing | plur.

The following is a well-formed AVM of type "AGREEMENT".

AGREEMENT[person: 3,number: sing].

The Type Hierarchy

The types are ordered in a subsumption hierarchy in which the constraints are inherited monotonically from type to sub type. Multiple inheritence is allowed, and the hierarchy can be defined top down or bottom up, using the "|" and "&" operators as follows.

SUPER-TYPE = TYPE1 | TYPE2 | TYPE3
TYPE4 = TYPE1 & TYPE2

This gives the hierarchy shown in Figure 2.

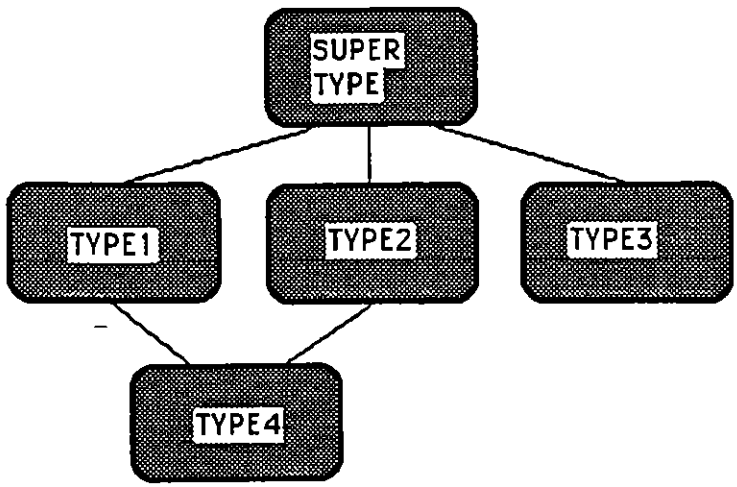

Figure 2. A Type Hierarchy.

Macros

Macros are used where the same data structure is needed in more than one place. They are like types, except that they do not participate in the type hierarchy.

3PS := AGREEMENT[person: 3,number: sing]

TFL and the MULTILEX lexical structure

Looking at the MULTILEX lexical structure, we can see that we must allow for the case where a lexical entry has a set of GPMUs, and for the case where a GPMU or GPMU set is common to several lexical entries. Using TFL to satisfy this combination of requirements is the key issue addressed by this paper.

There are several strategies we might adopt. The first is to use a facility included in the ACQUILEX TFL system whereby one lexical entry may inherit information from another lexical entry. This is done by allowing a value to identify an attribute path which includes the name of the other lexical entry.

```
bank1 =   [gpmu : [canonical-form: "bank",
                   phonology: 'banc'],
           syntax: [head: [syncat: noun]],
           semantics: "financial institution"].

bank2 =   [gpmu: [bank1:gpmu], ;; Inherit from bank1
           syntax: [head: [syncat: noun]],
           semantics: "edge of a river"].
```

However, in MULTILEX we need to allow for sets of GPMUs, and it is difficult to see how lexical inheritance can cope with this.

A second possibility is to use the inheritance hierarchy. This can be done by declaring the GPMU as a type, which is then inherited by any lexical entry that needs it.

```
BANK-GPMU = [gpmu: [canonical-form: "bank",
                    phonology: 'banc']].

bank1 =   [ gpmu: BANK-GPMU,
            syntax: [head: [syncat: noun]],
            semantics: "financial institution"].

bank2 =   [ gpmu: BANK-GPMU,
            syntax: [head: [syncat: noun]],
            semantics: "edge of a river"].
```

If a set of GPMUs is required, the GPMU can be declared as a super-type, where the sub-types are the alternative GPMUs. The normal behaviour of the hierarchy leads the alternative GPMUs to be handled as a set of alternatives.

SUPER-GPMU = GPMU1 | GPMU2 | GPMU3 .....

This is a neat solution, but it means an extension to the type hierarchy for each lexical entry. This is rather contrary to the concept of typing, and would lead to a huge type system.

The final possibility, which we have adopted, is to use the macro facility. The GPMU is declared as a macro. If a set of macros is required, each is declared as a separate macro, and a disjunction

of macros is included in the lexical entry. This provides exactly the properties we need, without overloading the type system.

```
BANK-GPMU := [gpmu:[canonical-form: "bank",
                    phonology: 'banc']].

bank1 =   BANK-GPMU &
          [syntax: [head: [syncat: noun]],
           semantics: "financial institution"].

bank2 =   BANK-GPMU &
          syntax: [head: [syncat: noun]],
          semantics: "edge of a river"].
```

And if a set of GPMUs is needed,

```
bank2 =   BANK-GPMU1 & BANK-GPMU2 & BANK-GPMU3
          syntax: [head: [syncat: noun]],
          semantics: "edge of a river"].
```

In the POLYGLOSS system, disjunction of macros is not allowed, so the prototype uses macros where no set of GPMUs is necessary, and the abovementioned method of declaring the GPMU as a type where sets are required.

Conclusion

Using a TFL based representation language for the MULTILEX Internal Format poses particular problems because of the structure of the lexical entries. In this paper, we present a neat solution which has been implemented using the POLYGLOSS TFS system.

Acknowledgements

Thanks to the ACQUILEX team at Cambridge University, and the POLYGLOSS team at Stuttgart University for their help with this work.

Bibliography

Copestake, A. 1991. Using the LKB. In Proceedings of the ACQUILEX workshop on default inheritence in the lexicon. By Briscoe et al. (eds). Technical Report 238. The Computer Laboratory, Cambridge University.

Carpenter, K. 1990. Typed feature structures: Inheritence, (in) equations and extensionality. In Proceedings of the first International Workshop in Inheritence in Natural Language Processing. Tilburg, The Netherlands.

Emele, M. and R. Zajac 1990. Typed Feature Grammars. In, Proceedings of the 13th International Conference on Computational Linguistics, Helsinki.