A comparison of ALE and PATR; practical experiences

Milena Stefanova  Wilco ter Stal
Department of Computer Science
University of Twente
Postbus 217, 7500 AE Enschede
Email: {stefanov, terstal}@cs.utwente.nl

Abstract

We describe our experiences with two unification-based grammar formalisms, namely ALE and PATR, used for syntactic and semantic analysis of NL texts. These formalisms can be considered to be high-level tools, like expert-system shells, for writing a NL grammar and for parsing sentences.

However, there are a number of aspects, such as ease of expression, augmentability, linguistic felicitousness and computational efficiency, in which ALE and PATR differ.

The experiment is carried out in the context of the Plinius project which requires, a.o. a large sublanguage grammar covering a fragment of English.

We will describe the formalization and implementation of our linguistic ideas in PATR and ALE. Subsequently, we will evaluate both exercises thereby focusing on both practical and theoretical properties of the two formalisms. The purpose of the experiment is to reveal whether there are motivated reasons concerning which kind of formalism can be employed best given the particular Plinius task.

1 Introduction

Currently, there are several unification-based formalisms available which can be put to use for specifying a NL grammar. Examples of such grammar formalisms are PATR, (Gazdar & Mellish [1989]; Shieber [1986]), ALE (Carpenter [1992]), ELU (Estival [1990]) and STUF (Oliva [1991]). A common feature of these formalisms is that no particular linguistic theory is advocated. In other words, these formalisms can be considered high-level tools, like expert-system shells, for writing a grammar. However, there are a number of aspects, such as ease of expression, augmentability, linguistic felicitousness, computational efficiency, in which the aforementioned formalisms differ. Depending on the task at hand one should weight these aspects and make an appropriate choice for one particular formalism. For instance, if a large-scale grammar is to be designed for batch processing of real texts, the flexibility and the augmentability of the formalism seems to be more important than the computational efficiency. At the other hand if one is engaged in the specification of a small grammar for a real-time application, such as a dialogue system, the computational efficiency, i.e. the use of time and memory resources, is of crucial importance.

In this article we will discuss our preliminary experiences with two specific formalisms, namely PATR and ALE. We are experimenting with both formalisms in the context of the Plinius project which requires, a.o., a large sublanguage grammar covering a fragment of English. We will focus on both practical and theoretical properties of the two formalisms. It is not our aim to provide a complete formal, mathematical comparison of the two formalisms.

1.1 Overview of remaining sections

First we will briefly discuss the Plinius project which forms the context of our research. Subsequently, we will have to describe what linguistic 'theory' we would like to formalize in order to make the comparison feasible. In addition we will discuss, in § 2, our engineering approach towards the NLP part of the system and the grammatical information we are using. In § 3 we will show
how this grammatical information is actually encoded in the PATR formalism. We will give an overview of the format of rules and lexical entries and explain the resulting output of parsing a sentence with PATR. In § 4 we will give an short description of ALE and show how the grammatical from § 2 and additional sortal information is specified in ALE. Section 5 contains an evaluation of the two formalisms. In § 6 we will summarize our experiment and point out some directions for further research.

2 The Plinius project

The Plinius project is aimed at developing a system which is capable of semi-automatically extracting domain-specific knowledge from the title and abstract of scientific publications in the field of ceramic materials. The knowledge base resulting from the Plinius project should have economical potential, that is, the benefits of using the knowledge base should outweigh the costs of developing it. In order to attain this goal we have made the following design decisions:

- use of abstractions
- interactive resolution of complex linguistic problems
- limitation to sublanguage
- reuse of (linguistic) resources
- application of an ontology

An ontology consists of a limited vocabulary of unambiguous concepts and their interrelations. The Plinius ontology presents a conceptual framework which structures the domain of ceramics at the knowledge level. Concepts from the Plinius ontology can be employed to express relevant knowledge on ceramic materials, such as their chemical composition, their (mechanical) properties, and the processes to produce them.

For a detailed review of the other design decisions and the Plinius ontology we refer to (Mars et al. [1993]; Vet & Mars [1993]). In the following section we will concentrate on NLP and more specifically on the grammar engineering part within Plinius.

2.1 Grammar engineering in the Plinius project

In this section we will discuss characteristics of the Plinius grammar. Initially, our goal was to construct a grammar by taking an existing broad-coverage grammar and tailoring it to the envisaged need within Plinius. However, large broad-coverage grammars are hardly available and the experiments we carried out with a well-known exception, the grammar of the ALNT project (Briscoe et al. [1989]), yielded unsatisfactory results. Moreover, due to the sublanguage employed in the abstracts we expected to encounter only a limited number of syntactic constructions. Therefore, we decided to develop our grammar in-house.

The second observation concerning the Plinius grammar is that it is rather eclectic. That is to say, that we do not confine ourselves to a particular grammatical theory such as LFG, HPSG or GB. Instead we employ several notions from different theories and implemented grammars, such as the grammar in the ANLT and Pundit (Paramax Systems Corporation [1992]). Examples of such notions are head, modifier, complement and operator.

A third property of our grammar is that we presuppose a close parallelism between the syntax (structural configuration) and the semantic representation of a sentence. However, we will notice a difference in the nature of the semantic representation yielded by PATR in comparison to the representation rendered by ALE.

A final property of our sublanguage grammar is that we would like to integrate so-called selectional or sortal restrictions, see (Alshawi & Carter [1992]), during parsing in order to rule out spurious structural analyses. Structural ambiguities, caused by for instance various potential PP-attachments, are resolved during sentential analysis.

We decided to specify a first version of our grammar in PATR. This decision was motivated by three reasons. The first was that PATR seemed expressive enough for the Plinius purpose. Secondly, the PATR formalism has a wide acceptance amongst linguists and is free to obtain. Thirdly, the SETI group of the University of Twente took interest in developing an efficient parser working with a context-free grammar annotated with features. In order to test and compare the performance of this parser with other parsers it seemed a good idea to express
the grammatical information in a standard formalism, namely PATR.

3 Encoding the Plinius grammar in PATR

3.1 The PATR formalism

The original PATR formalism\(^1\) has been developed at SRI International, Menlo Park, around 1984. However, the formalism rapidly changed into what has become known in the literature as PATR-II. The PATR-II formalism is extensively described in (Shieber [1986]), where it is proposed as a kind of 'lingua franca' to formalize and implement different theories of grammar. From now on we will use the term 'PATR' to denote the PATR-II formalism.

The intuitive idea behind the PATR formalism is that language constructs (sentences, phrases, words) can be associated with information packages. The sort of information associated with the aforementioned language constructs usually concerns syntactic and semantic information. Information packages are notated in so-called feature structures (henceforth: FSs).\(^2\)

Parsing in PATR consists of incrementally unifying the FSs associated with words and phrases according to, or driven by, the rules of the grammar. For a discussion of how the parsing strategy precisely operates on rules and feature structures, we refer to (Verlinden [1993]).

The grammar rules are in fact just like ordinary context-free phrase structure rules of the form \(A \rightarrow B C\), except for the fact that the nodes of the rules are annotated with constraints in terms of path equations of feature structures. For instance, the rule in figure 1 says that an \(S\) can be built from an \(NP\) and a \(VP\), if the associated constraints are satisfied. Similarly, grammar rules in the Plinius grammar consist of a phrase structure part which says how a particular mother node can be rewritten as a sequence of \(n\) daughter nodes and a bundle of path equations which operate on FSs associated with the nodes at the rule. In figure 2 an example of a rule (in Prolog notation) is given.

\[X_1 \rightarrow [X_2, X_3]\]
\[X_1 : \text{category} == s\]
\[X_2 : \text{category} == np\]
\[X_3 : \text{category} == vp\]
\[X_2 : \text{agreement} == X_3 : \text{agreement}\]

Figure 1: An example of a grammar rule in PATR.

\[X_1 \rightarrow [X_2, X_3]:\]
\[X_1 : \text{cat} == s,\]
\[X_2 : \text{cat} == np,\]
\[X_3 : \text{cat} == vp,\]
\[X_3 : \text{syn}_\text{sem} : \text{head} : \text{voice} == \text{active},\]
\[X_1 : \text{syn}_\text{sem} : \text{operator} == \text{null},\]
\[X_1 : \text{syn}_\text{sem} == X_3 : \text{syn}_\text{sem},\]
\[X_1 : \text{syn}_\text{sem} : \text{args} : \text{subject} == X_2 : \text{syn}_\text{sem},\]
\[X_3 : \text{arg} == X_1 : \text{syn}_\text{sem} : \text{content} : \text{arg}0,\]
\[X_1 : \text{syn}_\text{sem} : \text{mod} == \text{null}.\]

Figure 2: An example of a grammar rule in the Plinius grammar.

In the next subsection we will briefly describe which features are actually used in the Plinius grammar.

3.2 Feature structures for lexical and phrasal categories in Plinius

The choice for features in the Plinius grammar is organized in such a way that the result of a successful parse of a sentence yields (1) a conventional parse tree and (2) a FS describing syntactic and semantic information.\(^3\)

In what follows we intend to make clear how the syntactic and semantic information in Plinius is notated in terms of FSs. First, we will give the general format of the information associated with lexical categories (words), and then we will explain the format of phrasal categories. Finally, we give some examples of FSs of (parts of) sentences taken from an abstract text.

\(^1\)PATR is an acronym for 'parsing and translate'.

\(^2\)Also called Attribute-Value Matrices (AVMs), throughout this paper we will use the term feature structures.

\(^3\)Note that when a sentence is syntactically ambiguous the parser will give several FSs (and parse trees) per sentence, unless of course the grammar and lexicon employ selection restrictions in order to rule out spurious parses.
3.2.1 Lexical categories

The overall format of the FSs associated with the lexical entries is the following:

\[
\text{word} \Rightarrow [\text{cat} : \ldots, \text{syn.sem} : [\text{head} : \ldots, \text{content} : \ldots]]
\]

Thus each lexical entry in Plinius consists of a feature \text{cat} with values such as \text{noun}, \text{verb}, \text{adv} etc. This feature is used to construct the parse tree based on the nodes of the context-free skeleton of the grammar. The motivation for generating a parse tree stems for the fact that trees are much easier to inspect. The feature \text{syn.sem} consists of a bundle other features namely \text{head} and \text{content} which should be completed with syntactic and semantic information respectively. The dots after features indicate particular values which can of course differ per word. For instance, the lexical entries for the words \text{materials}, \text{dense} and \text{exhibits} amount to:

\[
\text{materials} \Rightarrow [\text{cat} : \text{noun}, \text{syn.sem} : [\text{head} : [\text{agr} : [\text{num} : \text{plur}]], \text{content} : [\text{relation} : \text{MATERIAL}]], \text{arg}0 : x]
\]

\[
\text{dense} \Rightarrow [\text{cat} : \text{adj}, \text{syn.sem} : [\text{head} : [\text{qua} : -]], \text{content} : [\text{relation} : \text{DENSE}]], \text{arg}0 : x]
\]

\[
\text{exhibits} \Rightarrow [\text{cat} : \text{verb}, \text{syn.sem} : [\text{head} : [\text{agr} : [\text{num} : \text{sing}]], \text{vform} : \text{fin}, \text{tense} : \text{present}, \text{relation} : \text{EXHIBIT.1}], \text{arg}0 : e1, \text{arg}1 : x, \text{arg}2 : y]
\]

3.2.2 Phrasal categories

In our grammar a phrasal category corresponds with a traditional grammatical category such as NP, AP, VP and PP. The general FS format of a phrasal category is:

\[
\text{phrase} \Rightarrow [\text{syn.sem} : [\text{operator} : \ldots, \text{head} : \ldots, \text{args} : \ldots, \text{mod} : \ldots, \text{content} : \ldots]]
\]

We will shortly discuss the function of these features.

The \text{operator} feature

The purpose of the \text{syn.sem|operator} feature is to provide a placeholder for linguistic expressions which function as some kind of operator. Examples of such expressions are \text{quantifiers} and \text{co-ordinators}. The idea is that these expressions operate on the syntactic and semantic content of a phrasal category and should be marked.

The \text{head} feature

The \text{syn.sem|head} feature again contains an FS representing syntactic information concerning the \text{head} of a phrase. The \text{syn.sem|content} feature contains the semantic information of the head of the phrase.

Informally, the head of a phrase coincides with the most prominent word in the phrase. For instance, in an NP the head usually is the noun, in a VP the main verb usually constitutes the head. A number of syntactic and semantic properties of an entire phrase are derived from the head of the phrase. Examples of these properties are: number, tense, predicate/argument structure.

In the Plinius grammar the \text{head} feature can have several values specific to the different categories. For instance, it does not make sense to talk about the tense of an adjectival phrase, hence the feature \text{tense} will not occur as a value in the \text{head} feature of an AP.

The \text{args} feature

The \text{syn.sem|args} feature of a phrase can be instantiated with the so-called \text{complements} of a phrase. This can be best illustrated by means of an example. Take for instance the verb \text{exhibits}. This transitive verb requires a subject-NP and an object-NP. In case a sentence like (1) is encountered, the grammar constructs an FS equal

\footnote{A path or a path equation of a FS is written as \text{feature|feature or feature|feature|feature==value}.}
(1) Materials exhibited elongation

\[
\begin{align*}
\text{operator} & : \ldots \\
\text{head} & : \begin{cases}
agr: & \begin{array}{l}
\text{num: Var} \\
\text{per: Var}
\end{array} \\
\text{vform: fin} \\
\text{voice: active} \\
\text{tense: past}
\end{cases} \\
\text{syn.sem} & : \\
\text{args} & : \begin{cases}
\text{subject: NP1} \\
\text{object: NP2}
\end{cases} \\
\text{mod} & : \ldots \\
\text{content} & : \\
& \begin{cases}
\text{relation: EXHIBIT.1} \\
\arg0: e1 \\
\arg1: x1 \\
\arg2: x3
\end{cases}
\end{align*}
\]

Note that the path equations \( \text{syn.sem|args|subject} = NP1 \) and \( \text{syn.sem|args|object} = NP2 \) are abbreviations. Parsing the whole sentence will yield an FS where the syn.sem features for the NPs materials and elongation are substituted for NP1 and NP2.

The \text{mod} feature

The \text{mod} feature is a storage place for those constituents which are modifiers of the head. For instance, the \text{syn.sem} features of the modifying PP in a test in sentence (2) are stored in the \text{mod} feature of (2').

(2) Materials exhibited elongation in a test

\[
\begin{align*}
\text{operator} & : \ldots \\
\text{head} & : \begin{cases}
agr: & \begin{array}{l}
\text{num: sing} \\
\text{per: S}
\end{array} \\
\text{vform: fin} \\
\text{voice: active} \\
\text{tense: past}
\end{cases} \\
\text{syn.sem} & : \\
\text{args} & : \ldots \\
\text{mod} & : \begin{cases}
\text{relation: EXHIBIT.1} \\
\arg0: e1 \\
\arg1: x1 \\
\arg2: x3
\end{cases} \\
\text{content} & : \\
& \begin{cases}
\end{cases}
\end{align*}
\]

The \text{content} feature

The semantic translation of most nouns, verbs, adjectives and prepositions, given in terms of predicate-argument structures, is expressed in the \text{content} feature.

An important aspect of (2'), which holds for FSs in general, is the fact that parts of an FS can be \text{structure shared} with another FS. In particular we make use of this property in case of the \text{content} feature. In case of the FS (2') this is illustrated by the fact that the first argument of the \text{EXHIBIT.1} relation, \( e1 \), functions as a \text{parameter} used as the second argument in the \text{IN.1} relation.

The values of the \text{content} features in the FS per sentence, are used to construct a so-called Quasi Logical Form (QLF). The idea of QLFs, as described in [Eijck & Alshawi [1992]; Sloat & Rentier [1993]; van der Vet et al. [1993]], is that they form a suitable datastructure for storing grammatical information relevant to further semantic and discourse processing. That is to say that QLFs form the input for an additional process which maps expressions in QLF onto the final representation in terms of the Plinius ontology.

An example of a QLF for sentence (2) is given in (2''). Note that the QLF in fact is a linear notation of the collected \text{content} features in (2').

\[
(2'') [\text{MATERIAL}(x1) \& \text{ELONGATION}(x2) \& \\
\text{EXHIBIT.1}(e1, x1, x2) \& \text{TEST}(x3) \& \text{IN.1}(e1, x3)]
\]

4 Encoding the Plinius grammar in ALE

In this section we describe relevant properties of the Attribute Logic Engine (ALE) and our experiences with developing a grammar in ALE. The version of ALE we are using runs under Quintus Prolog.

\footnote{Note that this is only one possible reading of the sentence, the parser also returns another analysis attaching the PP to the NP \text{elongation}.}
4.1 Description of the Attribute Logic Engine

ALE combines properties of three types of grammar and knowledge representation formalisms, namely, unification-based formalisms such as PATR and STUF, KL-ONE-like systems such as CLASSIC and Definite Clause Grammars.

ALE allows to model linguistic knowledge and knowledge about a certain domain as two separate parts in a single formalism. In fact, in our ALE grammar we are able to distinguish between two ontologies - one, describing concepts and relations in the Plinius domain, and another one describing the domain of linguistic knowledge. Elements of the linguistic domain are abstract entities like grammatical categories, inflexions of verbs, agreement, etc. In this fashion, a complex domain consisting of two subdomains can be specified in ALE as depicted in figure 3.

Elements of a domain are modeled by so-called *typed feature terms* (or descriptions). Every typed feature term denotes a set of typed feature graphs in a feature graph algebra, which coincide with the abstract objects of the linguistic domain and the objects of the Plinius ontology.

An advantage of this approach is that one might not only describe in the lexical entries syntax and syntax-driven (compositional) semantics, but also the actual meaning of the words in terms of the non-linguistic ontology. In such a way we are able to combine the benefits of both ontology-driven and syntax-driven semantics for representing sentence (and text) meaning.

The main properties of ALE can be summarized as follows. Firstly, ALE is a classification oriented formalism, which has much in common with systems for terminological knowledge representation. Types used in ALE play the same role as concepts used in KL-ONE-like systems for representing knowledge about a certain domain. As is well-known, see Brachman et al. [1991], these systems consist of two parts, namely a terminological part, or T-Box, and an assertional part, or A-Box. In a T-Box concepts about a certain domain are defined. In the assertional part we can define the objects of the domain under consideration in terms of the T-Box. Two important operations are employed in such systems namely concept and object classification. Concept classification computes the hierarchy (or subsumption preorder) of the concepts, and object classification performs pattern matching between components of the A-Box and proper concepts of the T-Box.

The state of affairs in ALE is similar. The type system in ALE corresponds to the T-Box, and the lexicon and the grammar rules - to the A-Box of a KL-ONE system. Every type is defined as a subtype of another type and, optionally, a set of features with possible values are listed.

Assertions in ALE can be static or dynamical. Statical assertions are in the lexicon, whereas dynamical assertions are made in the grammar rules.

The operation of concept classification is performed during compile time when the full type hierarchy is computed. There is an additional inference operation which is similar to object classification.

The second property of ALE is that it is a unification-based formalism. An ALE grammar is a context-free grammar in which feature graphs coincide with non-terminals and where natural language words are considered to be terminals. The most important operation is unification which is necessary when structure sharing is indicated in a feature graph. What differs ALE from the PATR is that feature structures employed in ALE are well-typed, see (Carpenter [1982]). This means that every feature structure is of a certain type and that every feature appropriate for this type must be present and must have a proper value. Using typed feature structures increases significantly time and memory ef-
ficiency of the parsing algorithm, see Carpenter [1993].

The third characteristic of ALE is that definite clause programming can be integrated into an ALE grammar. That is to say that Prolog-like goals can be added to grammar rules. However, instead of first-order logic terms only feature terms may be used. Conditions under which a certain goal is true are specified as definite clauses in the same way as in Prolog. Having these procedural attachments, arithmetics, processing of lists and other recursive data operations can be easily incorporated into the parsing process.

For a more detailed overview of ALE the reader is referred to (Carpenter [1992]).

4.2 Implementing the Plinius grammar in ALE

4.2.1 Format of an ALE grammar

In an ALE grammar one can distinguish three main components, namely the type hierarchy, the lexicon, and the grammar rules. Additional components, such as lexical rules and definite clauses, can be used to facilitate the writing of the lexicon and the grammar rules, respectively. We will explain briefly the format of the three main components in an ALE grammar.

The first thing to be specified is the type hierarchy. In order to do this we need two finite sets TYPE and FEAT of types and features respectively. In fact, the type hierarchy can be specified by defining two relations $\text{Sub} \subseteq \text{TYPE} \times \text{TYPE}$ and $\text{Intro} \subseteq \text{TYPE} \times \text{FEAT} \times \text{TYPE}$. Type specifications have the following format\(^8\):

\begin{align*}
\text{a sub } & \left[a_1, a_2, \ldots, a_n\right], \\
\text{intro } & \left[f_1 : b_1, f_2 : b_2, \ldots, f_k : b_k\right].
\end{align*}

There are several restrictions on the relations Sub and Intro, see Carpenter [1991], so that these relations meet the conditions appropriate for inheritance hierarchies and totally well-typed systems. In § 4.2.2 we give an example of a part of the type hierarchy for Plinius.

Besides the type hierarchy, we have to specify lexical entries. Lexical entries are notated in the following format:

\begin{align*}
\text{<word> } & \text{===> } \text{D}.
\end{align*}

Here \(<\text{word}>\) is a natural language word and \(D\) is a feature term. Note that it is not necessary to describe lexical entries fully in terms of the type hierarchy. One may specify only the features and types known for the moment, the missing types and most general features are inferred by the compiler. Additionally, the lexicon developer is able to define macros. These macros are abbreviations for complex feature structures and are expanded during compile time. For instance a word like car can be specified in the lexicon as \(3\) and amounts to \(3'\) after compilation.

\begin{align*}
(3) & \text{ car } \text{===> (syn: (cat: noun, agr: agreement(3, sg)))} \\
(3') & \text{ sign} \text{ SYN: syntax} \\
& \text{ CAT: noun} \\
& \text{ AGR: agreement} \\
& \text{ PERSON: 3} \\
& \text{ NUMBER: sg} \\
& \text{ SUBCAT: list_sign}
\end{align*}

In the remaining sections of the paper we will use the following orthographic convention. Features are written in uppercase letters, types are written in lowercase letters.

The format of grammar rules is as follows:

\begin{align*}
\text{<rule name> } & \text{ rule (D)} \\
& \text{ ===> } \\
& \text{ cat} > \text{ (D1)}, \\
& \text{ cat} > \text{ (D2)}, \\
& \vdots \\
& \text{ cat} > \text{ (Dn)}, \\
& \text{ goal} > \text{ G1, G2, ..., Gk}.
\end{align*}

Here \(D, D1, D2, \ldots, Dn\) are feature terms and \(G1, G2, \ldots, Gk\) are Prolog-like goals in which instead of first-order logic terms, feature terms are employed. Note that adding these goals is not obligatory for writing a grammar rule. In section 4.2.4, we give an example of a grammar rule in ALE.

4.2.2 Specifying the PLINIUS type hierarchy in ALE

In order to implement the PLINIUS grammar in ALE we first have to specify the type hierarchy. As was mentioned in § 4.1, in the type specification we will define two ontologies - one for the linguistic knowledge and one for the knowledge of
The main subtype is property. Semantic representations of common nouns, verbs, adjectives, prepositions are of type property. Every FS of type property has the following features defined on it: PARA (from parameter) standing for a particular world object having this property, RESTR (from restrictions) which value has a number of relations in which the parameter takes place, and the feature DET standing for a certain quantifier for every phrase. In fact FSs of type sem.obj encode in a proper way the representations which are the intended output of the grammar.

There is a subtype world of the type sem.obj which represent the domain under consideration (in our case the domain of material science). The type world has a number of subtypes standing for the concepts of the ontology. A graphical representation of this type hierarchy is given in fig. 7. An overview of the linguistic type hierarchy is given in fig. 6, for both figures the reader is referred to the appendices.

In the next section we will explain briefly the specification of the Plinius lexicon in ALE.

### 4.2.3 Lexical entries in ALE

We will give a few examples of some Plinius lexical entries in ALE, in order to clarify the relation between the linguistic and non-linguistic ontology.

Feature structures for lexical entries for the substantive parts of speech (verbs, nouns, etc.) amount, after compilation time, to:

\[
\text{syensem} \\
\text{CATEGORY:} \quad \begin{array}{l}
\text{HEAD:}\text{parts. of speech} \\
\text{SUBCAT:}\text{list. syensem}
\end{array} \\
\text{CONTENT:} \quad \begin{array}{l}
\text{property} \\
\text{DET:}\text{determiner} \\
\text{PARA:}\text{world} \\
\text{RESTR:non. empty. set}
\end{array}
\]

Note that the parameter, PARA is of a type which is a subtype of world, i.e., it is a FS representing a certain concept of the ontology. For example, the value of the PARA for the word temperature can be the following FS:

\[
\text{quantity} \\
\text{HAS.NAME: temperature} \\
\text{HAS.UNIT: kelvin} \\
\text{HAS.VALUE: value}
\]

Using such kind of structured parameters allows

---

9 Corresponding to the bar-levels in the k-scheme in GB.
10 The syntactic information about the constituents of a phrase is not directly relevant for the ultimate goal of Plinius.
to point exact links among the parameters of a relation, i.e. to give a detailed semantics to this relation. In case of verbs this can be illustrated as follows. The parameter of a verb is taken to be an event which may occur in the restricted world under consideration. The parameters of the subcategorization phrases of a verb should fill certain slots in the verb parameter. In such a way the exact relation between verb and its arguments is established. An example of the CONTENT feature of the verb exhibits is given below:

```
property
<table>
<thead>
<tr>
<th>DET: determiner</th>
</tr>
</thead>
<tbody>
<tr>
<td>PARA: [1] having matériel property</td>
</tr>
<tr>
<td>MATERIAL: [2] matériel</td>
</tr>
<tr>
<td>PROPERTY: [3] matériel property</td>
</tr>
<tr>
<td>non_empty_set</td>
</tr>
<tr>
<td>REST: [4] Relation</td>
</tr>
<tr>
<td>ARG0: [1]</td>
</tr>
<tr>
<td>ARG1: PARA: [9]</td>
</tr>
<tr>
<td>ARG2: PARA: [3]</td>
</tr>
<tr>
<td>empty_set</td>
</tr>
</tbody>
</table>
```

When the verb exhibits is combined with its arguments from the subcategorization list, the values of the features, ARG1 and ARG2, are filled with more specific information. Specifically, they can have values which are subtypes of the types matériel and matériel property respectively. In such a way the parser is able to construct a semantic representation of the sentence in terms of the non-linguistic ontology.

### 4.2.4 Grammar rules in ALE

Grammar rules have the format described in 4.2.1. The main difference which distinguishes ALE rules from PATR rules, i.e. grammar rules augmented with feature-value pairs, is that additional Prolog-like goals can be incorporated.

Having this possibility one can (partially) solve problems concerning the semantic representation of modifiers, plurals, coordination, and negation (see § 5.1), as well as cases in which real numbers are used as values of certain quantities.

For instance, the semantic principle behind the role of modifiers is that modifiers place additional restrictions on the parameter of the modified phrase. One can easily see that this principle may be implemented by the following rule for prepositional postmodifiers:

```
x1_x1_pp rule
(x1,
 synsem:(category:X1_category,
 content:(det:X1_determiner,
 para:X1_para,
 restr:L)))

====>

(x1,
 synsem:(category:X1_category,
 content:(det:X1_determiner,
 para:X1_para,
 restr:L))),

cat>(x2,
 synsem:(category:(head:(prep,
 predicative:yes),
 subcat: null)),
 content:(det:X1_det,
 para:X1_para,
 restr:L2)))

goal> append(L1,L2,L).
```

Definite clauses may also be used to deal with plural nouns. One would like to express that plural nouns represent some kind of set. In order to obtain an appropriate representation one can employ instead of single-value parameters, parameters with list-values. Then, in the lexicon one can specify that singular nouns have a parameter list with exactly one element. Plural nouns are as-
associated with a parameter list containing at least two elements. In case a plural noun is encountered in phrases like three materials, the following goal can be added to the rule for recognizing this NP:

\[
\text{goal} \rightarrow \text{length}(X, N).
\]

where \( X \) is the parameter of the word \textit{materials}, which is a list, and \( N \) is a natural number representing the parameter of \textit{three}, in this case \( N = 3 \).

5 Evaluation of ALE and PATR

In this section we will discuss and compare different aspects of the two formalisms. We will first focus on the theoretical aspects of the two formalisms and then discuss our practical experiences.

5.1 Theoretical issues

Underlying feature algebra

We already mentioned that the interpretation of the feature terms of ALE is given in a feature graph algebra. In fact the logic of ALE is a simplified form of feature logic, similar to the logic described in (Smolka [1988]). A special property of this feature logic is that the feature graphs employed for interpretation are totally well typed, since they conform to a number of appropriateness and type hierarchy conditions, see (Carpenter [1991]).

In case of PATR it is possible to give an interpretation in terms of a feature graph algebra, but these feature graphs will be untyped.

Disjunction and negation

In ALE it is possible to specify disjunctive and negative information. In the following example we show how negation can be modeled by using definite clause programming. Suppose that we would like to state that the \textit{PERSON} feature of \textit{agreement} is not 3. Then, one can use the following rule:

\[
\text{xx rule}
\]

...  

\[\Rightarrow \text{cat} \ldots\]

\[
\text{cat} \ \text{synsem:content:agr:X, cat} \ldots\]

\[
\text{goal} \rightarrow \not \text{third}(X). \quad 13
\]

\[
\text{not-third}(1).
\]

\[
\text{not-third}(2).
\]

In PATR it is not possible to specify negative or disjunctive information.

Explicit versus Implicit typing

An advantage of ALE is that, due to explicit typing of FSs, errors are detected already at compile time. That is to say that most of the work of unification is done in advance, i.e. before parsing a sentence.

PATR employs so-called \textit{implicit typing}. The compiler assigns a type to every FS and in case this is not possible the FS is not consistent with previous FSs. Unfortunately, this error can only be discovered during parsing.

Singleton types

There are no singleton types in the logic of ALE. That means that \textit{structure sharing} has to be indicated where it is necessary to point out that two FSs are identical, even in case where these FSs are intended to be just singleton types.

In PATR singleton types coincide with atoms.

5.2 Practical experiences

Simplicity of the formalism

It may be obvious from the description of ALE that writing a grammar in ALE is rather laborious. Especially, the type hierarchy forces a grammar writer to be very precise in choosing features. Fortunately, the formalism is \textit{lexical oriented} which means that the number of rules remains limited and that most of the grammatical information is specified at the lexical entries.

The learning curve of PATR is much lower and the less rigid use of features allows the developer to create a grammar much faster.

Correcting and augmenting the grammar

In ALE corrections should be made on a global level, i.e. in the type hierarchy. PATR allows a more flexible and local view on grammar rules. Corrections can be implemented directly in a rule, as long as the feature perlocations remain intact.

\[\quad \text{Or } (\not \text{third}(3)), \text{ where third}(3).\]
As far as concerns augmenting both grammars we cannot make any conclusive remarks yet. Constructions occurring in our Plinius corpus that should be incorporated in both formalisms are a.o.: \textit{Relative clauses}, \textit{Comparative constructions}, \textit{Partitive NPs} and \textit{Coordinated phrases}. Our intuitions concerning the implementation of these linguistic phenomena is that the PATR framework provides a better point of departure.

\textbf{Incorporating selectional restrictions}

In § 2.1, we stated that we would like to employ selectional restrictions in order to resolve structural ambiguities. It is straightforward that the strong typing used in ALE can be applied for this task. For instance, the structured parameters, as presented in § 4.2.3, prevent the combination of certain modifiers and modified phrases.

In PATR we could obtain the same effect by specifying an additional feature \textit{TYPE} at lexical entries and by writing extra path equations at a grammar rule. In these constraints we would check, for instance, whether the \textit{TYPE} of a modifier is compatible with the \textit{TYPE} of a modified phrase. It is clear that this is a rather cumbersome solution, compared to the built-in type system and type inferencing mechanism in ALE.

\textbf{Nature of output representation}

We have shown that ALE allows to integrate non-linguistic knowledge into the process of parsing. Every meaningful word is linked to a proper concept (type) of the Plinius ontology. One of the consequences of this aspect of ALE is that parsing a sentence yields directly a semantic representation in terms of the Plinius ontology. Further discourse processing can operate upon these output \textit{FSs} and results in the envisaged output of the Plinius project.

The current version of the Plinius grammar in PATR is implemented in a such a way that generated semantic representations are underspecified. After the parser delivers a QLF we have to design an additional process which links the intermediate representation to the final representational structures of the ontology.

\textbf{Coverage of the grammars}

The coverage of the grammar in ALE is rather limited, we experimented with a grammar which could handle: \textit{simple active sentences, VP complement structures, VPs with postmodifiers, NPs with adjectival and prepositional modifiers}.

The grammar of PATR has a wider scope, it includes the structures described above extended with: \textit{Passives, Coordinated APs, Numerical APs, Coordinated PPs, Infinitival constructions, simple Comparatives and Compounds}.

\textbf{Performance}

Unfortunately, we cannot (yet) provide any detailed figures regarding the parsing times of both formalisms. The experiences up until now with parsing sentences in both formalisms are very good. Note that in ALE most of the time is taken by compiling the type hierarchy (Carpenter [1993]), once this process is completed parsing a sentence of 10 words takes less than a second.

For details regarding time and memory efficiency of the head-corner parser which operates on the PATR rules we refer to Verlinden [1993].

\section{Discussion and further research}

We gave an overview of our experiences with coding a grammar in two unification-based formalisms, namely PATR and ALE. It is clear that the formalisms demonstrate both advantages and disadvantages. A major advantage of PATR is its flexibility which is useful for developing a grammar rapidly. A disadvantage of PATR is that it appears to be not very suitable for semantic processing of sentences. For instance, the incorporation of semantic (domain) knowledge in the same formalism, in order to prevent ambiguities and to generate an appropriate semantic structure, is not straightforward. Contrary to PATR it is very well possible in ALE to link domain knowledge with the process of parsing. A consequent of this aspect of ALE is that knowledge and grammar engineers need to collaborate closely.

Currently, we are not able to draw any firm conclusions regarding the choice in favor of one particular formalism. This is mainly due to lack of profound experience with constructing a real-size grammar in ALE. However, we plan to proceed with (the new version) of ALE. In addition, we would like to investigate the idea of combining 'best of both worlds'. In practice, this amounts to parsing sentences with PATR into QLFs, see § 3.2.2, and translating these QLFs into typed FSs in ALE. Inferencing, for instance for discourse purposes, will then take place in ALE which is more suitable for this task.

Another option is to investigate whether the PLEUK environment (Calder & Humphreys
[1993]) offers valuable support for the task of specifying the Plinius grammar. Ultimately, the usefulness of these experiments can only be estimated on the basis of the rapid development of a grammar with a sufficient coverage and appropriate output.

Acknowledgment
We would like to thank Hidde de Jong for proofreading a draft version of this paper.

References


G. Gazdar & C. Mellish [1988], Natural Language Processing in Prolog: an introduction to computational linguistics, Addison-Wesley, Wokingham.


Stuart Shieber [1986], "An introduction to Unification-Based Approaches to Grammar," CSLI Lecture Notes No. 4, Stanford, CA.


Paul E. van der Vet & Nicolaas J.I. Mars [1993],
"An ontology of ceramics, Version 2,"
University of Twente, Memorandum UT-KBS-93-13, Enschede, The Netherlands.

"Plinius intermediate report," University of Twente, Memorandum UT-KBS-93-36, Enschede, the Netherlands.