Features in Abstract and Concrete Syntax

Aarne Ranta
Department of Computer Science and Engineering
Chalmers University of Technology and Göteborg University

Abstract. *The division of grammars into abstract and concrete syntax is universally used in compilers. In linguistics, Curry's distinction between tectogrammar and phenogrammar is a similar idea. This architecture has become popular in recent years, exemplified by formalisms such as GF, ACG, and HOG. These formalisms give a perspective on language that is very different from the perspective in feature-based formalisms. The topic of this paper is how to "put the features back" into an abstract-syntax based formalism. After discussing a number of possibilities, we end up with the customary solution in GF, which uses records and record types and is thereby in many ways similar to typed feature structures. The viability of this approach is demonstrated by the GF Resource Grammar Library, which covers comprehensive fragments of 15 languages with a shared abstract syntax.*

1 Introduction

What is language-independent in a grammar? The units that are invariably present in all grammars of the Western tradition are certain parts of speech - noun, adjective, verb, etc - and certain syntactic constructs - sentences, questions, relative clauses, etc. Looking at the table of contents of a grammar book shows these concepts as the titles of major chapters. Thus a reader familiar with grammars can easily find information on a new language.

In addition to the language-invariant concepts, the grammar of each language needs concepts proper to that language. The major ingredient here is what the tradition calls grammatical categories: gender, number, case, tense, etc. While the names of these categories are again kept constant across languages, their contents show a considerable variation.

Not explicit in grammars, but more so in the philosophical tradition of "universal grammar", is that the parts of speech are essential to language, whereas the categories are, perhaps in varying grade, accidental. This view is partly confirmed by what is shown in grammar books: parts of speech are the same in all languages, but the grammatical categories are not.

In the formal grammars of modern linguistics, traditional concepts are still used for describing languages. The division into parts of speech and grammatical categories is shown in the distinction between what is now called *categories* and *features*, respectively. The category structure is, so to say, the backbone of most grammar formalisms, whereas features are secondary to this primary structure.

The real novelty in modern grammars is the systematic recognition of phrases as grammatical objects: noun phrases, adjectival phrases, verb phrases, etc. Just like words, phrases can take features, often inherited from or realized by the words functioning as their their heads.

For example, many grammars have an equivalent of a rule of predication, which combines a noun phrase (NP) and a verb phrase (VP) into a sentence (S). In pure phrase structure grammar, the rule reads

```
S -> NP VP
```

This rule does not say anything about *agreement*. Restricting attention to the number

agreement of English present-tense third-person sentences, agreement means that *the bird flies* and *the birds fly* are correct, whereas e.g. *the birds flies* is not.

In DCG (Definite Clause Grammar, (Pereira and Warren, 1980)), the predication rule with agreement is expressed by using a feature variable `n` ranging over numbers:

```
S -> NP(n) VP(n)
```

In feature structure grammars, such as PATR (Shieber, 1986), the grammatical units are presented as structures which put together a phonetic string, its category, and its features:

```
{phon = "birds" ; cat = N ; num = Pl}
```

2 The essence of the predication rule

In both DCG and PATR formats, the categories and features are intimately mixed. Even though it is possible to discern the "context-free backbone" of the grammar by retaining the categories and ignoring the features, this backbone cannot be formally separated from the features and recombined with a different system of features.

Another restriction, inherited from phrase structure grammars, is fixed word order. A verb-initial language cannot be given the same context-free skeleton as English. Yet, in a sense, word order is an accidental property of languages: it is possible to speak about predication in an abstract way, independently of whether it is realized in e.g. the SVO or the VSO manner.

All these problems can be solved if we distinguish between the *abstract syntax* and the *concrete syntax* of a language. This distinction, commonplace in computer science since (McCarthy, 1962), was introduced in linguistics by (Curry, 1963). The main idea is that

- the abstract syntax is a system of syntax trees (a free algebra)
- the concrete syntax is a mapping (a homomorphism) of the abstract syntax to some other structure (e.g. strings)

An abstract syntax is defined by

- a set of *categories*: types of syntax trees
- a set of *constructors*: functions between categories

The predication rule involves three categories and one constructor. Let us write this grammar fragment by using the notation of GF (Grammatical Framework, (Ranta, 2004)), which uses the keyword `cat` for judgements introducing categories, and `fun` for judgements introducing constructors.

```
cat S ; NP ; VP
fun Pred : NP -> VP -> S
```

The concrete syntax is defined in terms of *linearization rules* (`lin`), which can place the noun phrase before the verb phrase,

```
lin Pred np vp = np ++ vp
```

but also, obviously, in the reverse order. The variables `np` and `vp` refer to the linearizations of the subtrees, not to the subtrees themselves. Linearization is, in other words, *compositional*, which means that the linearization of a tree is a function of the linearizations of its subtrees. In the ordinary homomorphism notation, and expressing linearization with the asterisk, the above linearization rule says

```
(Pred np vp)* = np* ++ vp*
```

3 Features and abstract syntax

Having seen how word order is expressed in an abstract-syntax-based grammar, let us turn attention to features. A considerable variation is shown across the different approaches implementing Curry's architecture.

3.1 Features as part of abstract syntax

Following DCG, categories can be made dependent on features. Agreement can then be expressed in abstract syntax, by using a feature variable:

```
Pred : (n:Num) -> NP(n) -> VP(n) -> S
```

From the type-theoretical perspective, this is an instance of *dependent types*. Dependent types are available in GF and also in ACG (Abstract Categorial Grammars, (de Groote, 2001)), and

have recently been used to model agreement in ACG (Philippe de Groote, private communication). In HOG (Higher Order Grammar, (Pollard, 2004)), a related device of *subtypes* is used to introduce agreement in abstract syntax.

If agreement is expressed in the way above, it becomes problematic to instantiate the rules in languages with different feature systems. To be fair, multilinguality is not among the goals stated in ACG and HOG. Moreover, including features in abstract syntax is a powerful technique and can handle some rules that are difficult without; see Section 6.1 below.

3.2 Sets of linearization functions

(Montague, 1974) used the Curry architecture without mentioning it. In his grammar fragments, he did not include plural because of difficulties in interpreting them semantically. However, modelling an agreement-sensitive predication rule after some of his other rules gives the following formulation for the function F_4 Montague used for predication:

> $F_4(\alpha, \delta)$ is $\alpha\delta'$, where δ' is the result of turning the first verb of δ is to the singular or the plural depending on whether α is singular or plural.

Since the rule involves a change of the linearization of y, it is not compositional. But it is easy to reformulate it compositionally, by making the linearization of verb phrases dependent on number.

```
linVP : VP -> Number -> Str
```

Linearizing sentences and noun phrases needs no extra arguments,

```
linS  : S  -> Str
linNP : NP -> Str
```

but the number of noun phrases has to be computable,

```
numNP : NP -> Number
```

Now we can define

```
linS (Pred np vp) =
  linNP np ++ linVP (numNP np) vp
```

The abstract syntax is now free from features. It is also efficient to use the grammar in the direction of generation: no unification of features is needed, since there is always a direction in which one constituent passes its features to another.

Formally, implementing features by sets of functions is similar to attribute grammars (Knuth, 1968) limited to *synthesized attributes* only. Linguistically, it is a direct implementation of the distinction between *variable and inherent features*, which is present in traditional grammars but disappears in unification grammars. In the example above, number is inherent for noun phrases but variable for verb phrases. A representative of tradition, (Grevisse, 1993) systematically lists the variable and inherent features when introducing the different parts of speech in French. For example, "a *noun* or *substantive* is a word that carries a gender and is susceptible to vary in number" (paragraph 449).

The method of sets of functions was adopted in (Ranta, 1994). But it turned out tedious and error-prone to work with systems of many functions instead of just one linearization function per category. Moreover, the definition of compositionality for families of functions is complicated.

3.3 Linearization types

The French linearization function of nouns is

```
linN : N -> Number -> Str
```

Bearing in mind that the function type arrow is right-associative, `linN` is in effect a function from `N` to functions from `Number` to `Str`. This gives rise to the notion of *linearization type*, which is a language-dependent type indicating the value type of linearizing a given category. It is analogous to the *domain of possible denotations* of a category in Montague's semantic homomorphisms.

To define linearization types of categories, GF provides judgements with the keyword `lincat`. Thus, we can write

```
lincat N = Number -> Str
```

for French, whereas in English, we write

```
lincat N = Number -> Case -> Str
```

As a general rule, the linearization function `lin` has the dependent product type

```
lin : (C : Cat) -> C -> lincat C
```

This is exactly what is needed for a rigorous definition of compositionality: if

$$f \ : \ C_1 \to \cdots \to C_n \to C$$

then (denoting both linearization types and linearizations with an asterisk)

$$f^* \ : \ C_1^* \to \cdots \to C_n^* \to C^*$$

and

$$(f a_1 \ldots a_n)^* \ = \ f^* a_1^* \ldots a_n^*$$

We are not quite ready with the concept of a linearization type, because we have not shown how the inherent features fit in them. In French, nouns have gender as an inherent feature. Instead of using a separate gender function, we can now include gender as a part of the linearization type of nouns by using a complex type: a *labelled record type*,

```
lincat N = {
  s : Number => Str ;
  g : Gender
  }
```

With such linearization types, which collect together all the information assigned to a tree by a concrete syntax, compositionality is easily verified: it follows the above scheme precisely. (The double arrow `=>` denotes a *finite function type*, which is an important concept in the metatheory of GF; see Section 5 below.)

A precise formulation of the agreement rule can now be given. The linearization types are

```
lincat S  = {s : Str}
lincat NP = {s : Str ; n : Number}
lincat VP = {s : Number => Str}
```

The linearization rule for predication is

```
lin Pred np vp =
  {s = np.s ++ vp.s np.n}
```

3.4 Grammatical objects

The step from families of linearization functions to linearization types can be seen as a step to object-oriented programming. The linearization is now, so to say, in the data objects themselves rather in the functions that manipulate them. A linearization type is thus like a *class* in the object-oriented sense, and the linearization of a tree is like an *object* of such a class. The *methods* of the class are the functions that yield strings, and the *attributes* are the inherent features.

The object-oriented move is beneficial for the maintenance of grammars: if we want to add a new rule or a new lexical entry, we don't need to go and change many different already existing functions, but just write a a new rule.

4 Labelled records and typed feature structures

The labelled records of GF have a structural type system, similar to records in Pascal and structures in C. They are thus weaker than the typed feature structures of HPSG (Pollard and Sag, 1994). The latter use circular references to model recursion; in GF, recursion is modelled by trees in the abstract syntax.

Even more importantly, the way records are used in GF is different from feature-based unification grammars. This follows directly from the abstrac syntax based architecture: In GF, a record represents the information needed for linearizing a tree. In unification grammars, a record represents the information obtained from parsing a string. For instance, the abstract French word `Maison` has a linearization record

```
{s = table {
  Sg => "maison" ;
  Pl => "maisons"} ;
g = Fem}
```

In a PATR-like system, two feature structures are involved: one for the string *maison* and one for *maisons*:

```
{s = "maison" ;          {s = "maisons" ;
 n = Sg ;                 n = Pl ;
 g = Fem]                 g = Fem}
```

5 Expressivity and complexity

GF is equivalent to Parallel Multiple Context-Free Grammars (Seki et al., 1991) and has therefore an unbounded polynomial parsing complexity. This result holds under some restrictions on dependent types in abstract syntax (Ljunglöf, 2004). Linearization can be made linear in the size of trees by using partial evaluation to a simpler format (Ranta, 2004). Both these results use in a crucial way the fact that parameter-dependent functions can only take a finite number of different arguments and are therefore representable as records at runtime.

6 The GF Resource Grammar Library

Typical applications of GF are multilingual grammar systems based on a shared abstract syntax. To ease the creation of such applications, a grammar library has been developed to take care of linguistic details (Ranta 2007, El Dada & al. 2006). The main part of the library has a common abstract syntax, called the Ground API. The library is currently available in 15 languages, of which 10 have a full implementation of the Ground API. (Danish, English, Finnish, French, German, Italian, Norwegian, Russian, Spanish, and Swedish) and 5 are work in progress (Arabic, Catalan, Swahili, Thai, and Urdu).

The Ground API covers a fragment comparable to CLE (Core Language Engine, (Rayner et al., 2000)). Apart from the practical advantage of having a common interface to different languages, there is a typological interest in being able to formally compare the realization of grammatical structures in different languages. In this respect, the Ground API is similar to the LinGO grammar matrix (Bender and Flickinger, 2005), which is a method of grammar development starting with a questionnaire including a few crucial questions (such as how negation and questions are formed). Our approach takes this idea to the extreme, covering almost all of the grammar, in a formal way permitted by the abstract syntax architecture.

The functioning of a common abstract syntax is of course heavily dependent on the use of different features and linearization types for different languages. To take just one example, we can consider the category of verb phrases (VP). Verb phrases are used for building clauses (Cl),

```
PredVP : NP -> VP -> Cl
```

which are like sentences (S) but have a variable tense and polarity (negatedness). Clauses can also be converted to yes-no questions and must therefore support inversion in many languages. In addition to clauses, verb phrases are used for building relative clauses and wh-questions, and as complements of certain verbs. Verb phrases are constructed from different kinds of verbs and their appropriate complements.

The challenge in implementing rules involving VP varies from one language to another. In German and the Scandinavian languages, the topological structure (Diderichsen, 1962) implies that a VP consists of different parts that can be combined in diffent ways in main clauses, subordinate clauses, and various topicalizations. The Scandinavian VP linearization type is

```
{s : VPForm => {
   v1 : Str ;       -- har
   v2 : Str         -- lovat
   } ;
 a1 : Pol => Str ; -- inte
 n2 : Agr => Str ; -- dig
 a2 : Str ;        -- idag
 ext : Str         -- att sluta
 }
```

Together with a subject, e.g. *jag*, this permits the different combinations *jag har inte lovat dig idag att sluta* ("I haven't promised you today to quit", unmarked), *jag inte har lovat dig idag att sluta* (subordinate), *har jag inte lovat dig idag att sluta* (question), *dig har jag inte lovat idag att sluta* (emphasis on "you"), etc.

In Romance languages, the main problems of VPs are different: clitic pronouns and the agreement of the participle in compound tenses. Therefore, the linearization type is quite different from Scandinavian. (However, by using *parametrized modules*, the library manages to have just one definition for the Scandinavian

family and another one for the Romance family; cf. (Ranta, 2007).)

6.1 Language-dependent parts

In addition to the ground API, there are language-dependent extensions. Some structures cannot be naturally implemented in all the included languages. In particular, there are rules whose formulation requires features to be present in abstract syntax, to control what trees are constructible. One example is two-place-verb coordination in case-rich languages. With some variations, the usual requirement is that the complement cases be the same:

```
fun coord : Conj -> (c : Case) ->
          V2(c) -> V2(c) -> V2(c)
```

7 Related work

Several frameworks based on a distinction between abstract and concrete syntax have appeared in the last 5 years: ACG (de Groote, 2001), HOG (Pollard, 2004), and Lambda grammars (Muskens, 2001). Multilinguality has not been in focus in these formalisms. Possibly for this reason, the treatment of features in terms of linearization types is unique to GF.

Multilingual grammar packages in other formalisms include the LinGO matrix (Bender and Flickinger, 2005), ParGram (Butt et al., 2002), CLE (Rayner et al., 2000), and Regulus resource grammars (Rayner et al., 2006). None of these uses a formally shared representation similar to GF's abstract syntax. The number of languages covered in these projects is comparable to GF in LinGO and ParGram, half of it in CLE and Regulus. The coverage in terms of structures varies from one language to another in each of these packages.

References

E. M. Bender and D. Flickinger. 2005. Rapid prototyping of scalable grammars: Towards modularity in extensions to a language-independent core. In *Proceedings of the 2nd International Joint Conference on Natural Language Processing IJCNLP-05 (Posters/Demos)*, Jeju Island, Korea.

M. Butt, H. Dyvik, T. Holloway King, H. Masuichi, and C. Rohrer. 2002. The Parallel Grammar Project. In *COLING 2002, Workshop on Grammar Engineering and Evaluation*, pages 1–7.

H. B. Curry. 1963. Some logical aspects of grammatical structure. In Roman Jakobson, editor, *Structure of Language and its Mathematical Aspects: Proceedings of the Twelfth Symposium in Applied Mathematics*, pages 56–68. American Mathematical Society.

A. El Dada, J. Khegai, and A. Ranta. 2006. The GF Resource Grammar Library. Software and documentation, available on GF library homepage `http://www.cs.chalmers.se/~aarne/GF/lib/`.

Ph. de Groote. 2001. Towards Abstract Categorial Grammars. In *Association for Computational Linguistics, 39th Annual Meeting and 10th Conference of the European Chapter, Toulouse, France*, pages 148–155.

P. Diderichsen. 1962. *Elementaer dansk grammatik*. Kobenhavn.

M. Grevisse. 1993. *Le bon usage, 13me edition*. Duculot, Paris.

D. Knuth. 1968. Semantics of context-free languages. *Mathematical Systems Theory*, 2:127–145.

P. Ljunglöf. 2004. *The Expressivity and Complexity of Grammatical Framework*. Ph.D. thesis, Dept. of Computing Science, Chalmers University of Technology and Gothenburg University.

J. McCarthy. 1962. Towards a mathematical science of computation. In *Proceedings of the Information Processing Cong. 62*, pages 21–28, Munich, West Germany, August. North-Holland.

R. Montague. 1974. *Formal Philosophy*. Yale University Press, New Haven. Collected papers edited by Richmond Thomason.

R. Muskens. 2001. Lambda Grammars and the Syntax-Semantics Interface. In R. van Rooy and M. Stokhof, editors, *Proceedings of the Thirteenth Amsterdam Colloquium*, pages 150–155, Amsterdam.

F. Pereira and D. Warren. 1980. Definite clause grammars for language analysis—a survey of the formalism and a comparison with augmented transition networks. *Artificial Intelligence*, 13:231–278.

C. Pollard and I. Sag. 1994. *Head-Driven Phrase Structure Grammar*. University of Chicago Press.

C. Pollard. 2004. Higher-Order Categorial Grammar. In M. Moortgat, editor, *Proceedings of the Conference on Categorial Grammars (CG2004), Montpellier, France*, pages 340–361.

A. Ranta. 1994. *Type Theoretical Grammar*. Oxford University Press.

A. Ranta. 2004. Grammatical Framework: A Type-theoretical Grammar Formalism. *The Journal of Functional Programming*, 14(2):145–189.

A. Ranta. 2007. Modular Grammar Engineering in GF. *Research on Language and Computation*, to appear.

M. Rayner, D. Carter, P. Bouillon, V. Digalakis, and M. Wirn. 2000. *The Spoken Language Translator*. Cambridge University Press, Cambridge.

M. Rayner, B. A. Hockey, and P. Bouillon. 2006. *Putting Linguistics into Speech Recognition: The Regulus Grammar Compiler*. CSLI.

H. Seki, T. Matsumura, M. Fujii, and T. Kasami. 1991. On multiple context-free grammars. *Theoretical Computer Science*, 88:191–229.

S. Shieber. 1986. *An Introduction to Unification-Based Approaches to Grammars*. University of Chicago Press.