Electronic Notes in Theoretical Computer Science 43 (2001)
URL: http://www.elsevier.nl/locate/entcs/volume43.html

Grammatical Specification in ASL: Germanic Dependent Clause Order

Michael Newton [1]

Aurema Pty. Ltd.
Sydney, Australia

Abstract

In this paper, I consider the use of the algebraic specification language ASL ([10]) in treating dependent clause word order in English, German and Dutch. ASL is a *wide-spectrum* language, in which one can write *loose specifications*, admitting of non-isomorphic models. Within the language, one can describe the relationship between one's abstract, inspecific notion of the properties a grammar should have, and the specific grammars which one employs in seeking to test and refine this notion. A loose specification can also act as a *parameter* in a specification embodying formally the idea that a grammatical contruction, intended to account, say, for a particular linguistic phenomenon, could be made to work equally well in a variety of grammatical frameworks, or in accounting for a variety of natural languages.

1 Dependent Clause Order

A dependent clause is an embedded clause like *that Michael saw Harold swim*, in *I believe that Michael saw Harold swim.* In English, the word order of the material following *that* is much the same as it would be in a main clause: *Michael saw Harold swim.* In German and Dutch, however, this is not so. For instance, in Dutch, we would have for the main clause. In these languages, it is often taken that the embedded order is canonical. For instance, in German one might derive main clause order by a combination of main verb inversion, and topicalisation (as for instance in [9]). In [7], I showed how topicalisation may be dealt with by a modular extension, at least for the case of English. Thus dependent clause order is a reasonable choice for a Germanic "core" grammar, which may be built upon by modular extension to broader coverage.

For the sake of simplicity, I will limit my consideration here to verbs which take nominal and verbal arguments only (so, for instance, I will not deal with

[1] Email: kimba@aurema.com

©2001 Published by Elsevier Science B. V. Open access under CC BY-NC-ND license.

place, as in *place the salt on the table*). In English we may indicate constituent (phrasal) structure by brackets:

that [Michael$_1$ saw$_1$ [Harold$_2$ swim$_2$]$_1$].

(Here the numerical subscripts are meant to indicate arguments to the verb). Similarly in German:

dass [Michael$_1$ [Harold$_2$ schwimmen$_2$]$_1$ sah$_1$].

Dutch is less obviously susceptible to a constituency treatment, because the dependencies (indicated by the subscripts) *cross* ([2]):

dat Michael$_1$ Harold$_2$ zag$_1$ zwemmen$_2$.

However the less restrictive notion of *dependency* can still be useful, at the very least to provide a descriptive vocabulary.

2 Dependency Grammar

From this simple example it might be thought that we could describe the Dutch case by first saying that *zag* subcategorises directly for an NP (noun phrase) and a VP (verb phrase), instead for an S (sentence), but if we replace *zwemmen* with the transitive verb *kussen* (to kiss), we see this will not help:

dat Michael$_1$ Harold$_2$ Maria$_2$ zag$_1$ kussen$_2$.

We could try to push the strategy even further, and say for instance that *zag* subcategorises for a collection of elements which could themselves form a sentence (differently ordered). But it will be much simpler if we simply say that *zag* subcategorises for a verb, in this case *kussen*, which in turn subcategorises for a subject *Harold* and an object *Maria*. Then we might describe the Dutch order (in part) by saying that a verb v which is argument to some r must succeed r, but other arguments to r precede it. Having eliminated the S and VP level entities, we may as well eliminate NP too, simply saying that *zwem* (for example) subcategorises for a noun, and that a proper noun (such as *Harold*) need take no arguments, but a (singular) common noun (like *man*) subcategorises for a determiner (say *een*) which must come immediately to its left.

Thus we eliminate all phrasal representations, and are left just with lexical entries. In a dependency grammar (see for instance [6]), we say that the lexical entries for *Michael* and *zwemmen* are *dependents* of that of *zag*, or, equivalently, that the entry for *zag* is *head* to those of *Michael* and *zwemmen.*[2] Every entry is thus associated with a particular collection (multiset) of dependent entries. (In dependency grammar it is usual to describe adjuncts —

[2] My terminology here is perhaps somewhat non-standard, in that I refer to the fully grounded entities which stand for words as lexical entries, rather than some underspecified entities which must be filled in before use. In examples like ADependencyModel below, such underspecified entities will instead correspond to constructor functions.

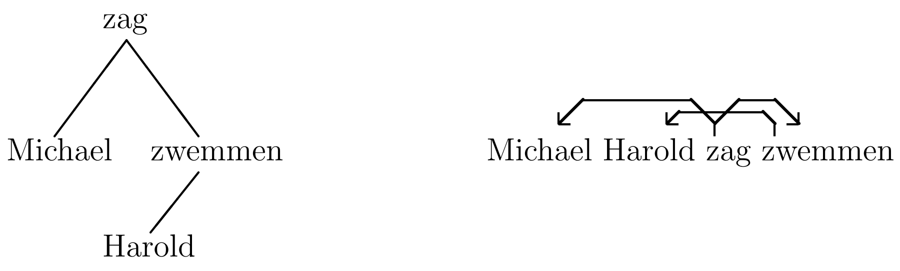

Fig. 1. A dependency tree

like *quickly* in *Harold swam quickly* — as well as subcategorised-for arguments as dependents.) Since each dependent entry has its own associated multiset of dependents, any entry can be seen as the root of a tree, with its dependents forming the daughters, as in Figure 1. On the left, the dependencies are drawn in a familiar tree form, with mothers (heads) at the top of branches, and daughters (dependents) at the bottom. On the right, we see the conventional representation used in dependency grammar, called a *dependency diagram*, with arrows running from head to dependent. If s appears immediately under r in the tree on the left, it is a dependent. If it appears *somewhere* under r, it is called a *subordinate* of r. Subordinacy is the reflexive, transitive closure of dependency. (We could use the term *proper subordinacy* to refer to closure under transitivity alone — so r is not properly subordinate to r.)

Dependency is the same sort of relation which exists between a head daughter and its sisters in phrase structure grammars like HPSG ([8]) — the sisters depend on the head — except that instead of having separate entities for a head daughter and mother, the entity associated with the head must itself, in some fashion, license some realisation(s) as string(s) of words. The existence of dependents already associates with any entry a particular multiset of words, namely the words for which its subordinates are entries; the collection formed from the word associated with the entry, plus the words associated with its dependents, plus the words associated with *their* dependents, and so on. For instance, consider the sentence *Michael saw Harold swim.* In a dependency grammar, this would be licensed by a lexical entry for the word *saw.* This entry would have as dependents entries for *Michael* and *swim.* The entry for *Michael* has no dependents; that for *swim* has as a dependent an entry for *Harold* (which also has no dependents). Thus such an entry for *saw* is associated with the multiset of words $\{saw, Michael, swim, Harold\}$. In this way, dependency can be used to license the collection of words which appear in well-formed utterances, but not their order.

3 Classes of Models

I will begin by writing some ASL specifications which attempt to constrain what it *means* to model dependent clause order. ASL is really a family of languages: for an explication of the particular variant employed here, see [7].

To begin with, we are talking about strings of words.

```
Strings =
    extend sort Word
    by  sort Word*
        opn _ · _ : Word* × Word* ⟶ Word*
        opn e :⟶ Word*
        axiom Word ⊆ Word*
        axiom $x \cdot (y \cdot z) = (x \cdot y) \cdot z$
        axiom $x \cdot \mathsf{e} = \mathsf{e} \cdot x = x$
```

The concept of free extension is related to that of initiality. The free extension of a model M by some new syntax and axioms is, up to isomorphism, formed by considering the elements of M as constants, and forming a model populated by equivalence classes of terms, in the familiar way, over those constants and the new syntax. If M is a model of S, its free extension by *new syntax and axioms* is a model of the specification **extend** S **by** *new syntax and axioms.* The equivalence classes for a model of Strings contain all alternative bracketings of a string of elements from `Word`, plus arbitrary occurrences of `e`, joined with _ · _. This is isomorphic to the set of strings over `Word`, with `e` naming the empty string, and _ · _ interpreted by concatenation. The semantics of Strings is thus the class of string algebras over a sort `Word`. It is a *loose* specification, admitting of non-isomorphic models, precisely to the extent that **sort** `Word` is. The specification $\lambda\mathcal{X}$: **sort** `Word` . $\mathcal{X}$ + Strings is a *parameterised specification*: it is interpreted by the map which takes any specification $\mathcal{X}$ whose models are also models of **sort** `Word`, and produces the specification $\mathcal{X}$ + Strings, whose models are those models of the combined syntax of $\mathcal{X}$ and Strings which are also models of both $\mathcal{X}$ and `Strings` (if one ignores, in each case, syntax irrelevant in that case). It happens that this parameterised specification is also a *parameterised implementation*, in that if models of $\mathcal{X}$ are mutually isomorphic, so are models of $\mathcal{X}$ + Strings.

```
WellFormed =
    enrich Strings by
        pred well_formed ⊆ Word*
```

WellFormed picks out a subset of these strings. The specification **enrich** S **by** *new syntax and axioms* allows all models which account for both the new syntax and axioms, and, ignoring that new syntax, the old specification S. Thus, since WellFormed says nothing about which subset is picked out, it admits models picking out any subset. Therefore $\lambda\mathcal{X}$: **sort** `Word` . $\mathcal{X}$ + `WellFormed` is *not* a parameterised implementation, since even if models of $\mathcal{X}$ are mutually isomorphic, models of $\mathcal{X}$ + `WellFormed` need not be, since they can pick out different strings as well-formed. It does, however, express perhaps the very broadest idea of what a grammar is: for a given set of words, it must characterise which strings of words are well-formed.

I take the position that `Word` is supposed to represent the observable, context-independent defining characteristics of a word, such as its phonological form. Write $w \sim r$ to mean that r is a lexical entry for the word w. M is a model of Lexicon (below) if it maps `Word` and `Entry` to sets, and $\sim$ to a binary relation across those sets.

```
Lexicon =
    sorts Word,Entry
    pred _ ∼ _ ⊆ Word × Entry


Multisets =
    extend sort Entry
    by  sort Entry⁺
        opn _ + _ : Entry⁺ × Entry⁺ → Entry⁺
        opn 0 :→ Entry⁺
        axiom Entry ⊆ Entry⁺
        axiom $x + y = y + x$
        axiom $x + (y + z) = (x + y) + z$
        axiom $x + 0 = 0 + x = x$
Dependency =
    enrich Multisets by
        opn dependents : Entry → Entry⁺
Subordinates =
    extend Dependency by
        opn subordinates : Entry⁺ → Entry⁺
        axiom ∀r:Entry . subordinates(r)
                   = r + subordinates(dependents(r))
        axiom subordinates(0) = 0
        axiom subordinates(x + y)
                 = subordinates(x) + subordinates(y)
(Relate strings to multisets)
StringToMultiset =
    extend Lexicon + Strings + Multisets by
        opn _ ∼ _ ⊆ Word* × Entry⁺
        axiom e ∼ 0
        axiom $x \sim y \wedge x' \sim y' \to x \cdot x' \sim y + y'$
MultisetLicensing =
    enrich Subordinates + StringToMultiset + WellFormed by
        axiom well_formed(x) → ∃r:Entry . x ∼ subordinates(r)
```

Multisets is very similar to Strings, but the addition of the commutativity axiom $x + y = y + x$ puts together terms which are the same except for order, so we end up with models isomorphic to multisets, or bags (like sets in which an element can occur multiple times), with $+$ representing union, and 0, the empty set. In a model of Dependency, the function interpreting `dependents` must map any entry to a multiset of entries. Subordinates embodies the definition of subordinacy as the transitive closure of dependency. In a similarly constructive vein, StringToMultiset distributes $\_ \sim \_$ over $\_ \cdot \_$ and $\_ + \_$, relating a multiset of words to a string of entries if each element of the multiset is related to a different element of the string. MultisetLicensing then insists that a well-formed string is composed from the lexical forms of all the subordinates of some head entry r. Modulo some extra syntax, MultisetLicensing is a *refinement* of WellFormed, not because the former was constructed from the latter, but because every model of MultisetLicensing also models WellFormed. This example of *stepwise refinement* ([10]) may be viewed as embodying the claim that any language may be accounted for by a dependency system. As it stands, however, this is not a very interesting claim, since any model of WellFormed may form the basis of a model of MultisetLicensing. Its being a refinement is of more interest as a link in a chain of refinements, continued below.

Any specific dependency grammar must furnish a set of lexical entries (model of Lexicon) and an account of dependency (model of Dependency). A *system* of dependency grammars will map a model which implements Lexicon, and Dependency, and something to take care of word order, to a model of MultisetLicensing.

DependencySystem =
$\lambda \mathcal{X}$: (Lexicon + Dependency + OrderLicensing) . MultisetLicensing

To make DependencySystem into a meaningful specification, we need to define OrderLicensing. The preceding formal vocabulary is insufficient to allow a statement such as, "nouns precede verbs", since the same word might act as either a noun or a verb in different contexts, even within the one sentence. One needs to establish a correspondence between, on the one hand, each licensing lexical entry, and on the other, a word-in-context. One way to achieve this is to consider the different possible orderings of a multiset of entries, as strings of entries. EntryStrings, below, defines strings of entries, by a renaming of Strings. If M is a model of S and σ maps into the syntax of S, the functional composite of σ and the mappings which make up M is a model of the source syntax of σ, and therefore a model of **derive from** S **by** σ. OrderLicensing picks out a well-formed subset of these entry strings.

EntryStrings =
derive from Strings **by** $[\texttt{Word} \longmapsto \texttt{Entry}, \texttt{Word}^* \longmapsto \texttt{Entry}^*]$

OrderLicensing =
enrich EntryStrings **by**
pred $\texttt{well\_formed} \subseteq \texttt{Entry}^*$

MultisetLicensing is still a *loose* specification, admitting of non-isomorphic models, as it says nothing about the order of words. For this, we will need to relate the order of words, in WellFormed, to the order of entries, in OrderLicensing. This will allow us to produce a refinement of MultisetLicensing which also shows how a dependency grammar can license order. Projections, below, gives us the vocabulary to identify the multiset of entries with the same elements as a given string of entries. Indexes distributes $\sim$ over $\texttt{Entry}^*$ much as StringToMultiset did for $\texttt{Entry}^+$.

Projections $=$
    extend EntryStrings $+$ Multisets **by**
        opn $|\_| : \texttt{Entry}^* \rightarrow \texttt{Entry}^+$
        axiom $|\mathsf{e}| = 0$
        axiom $\forall r\!:\!\texttt{Entry}, x\!:\!\texttt{Entry}^* . \, |r \cdot x| = r + |x|$

Indexes $=$
    extend Lexicon $+$Strings $+$ EntryStrings **by**
        opn $\_ \sim \_ \subseteq \texttt{Word}^* \times \texttt{Entry}^*$
        axiom $\mathsf{e} \sim \mathsf{e}$
        axiom $x \sim y \land x' \sim y' \rightarrow x \cdot x' \sim y \cdot y'$

DependencyGrammar $=$
    $\lambda\mathcal{X}$: (Lexicon $+$ Dependency $+$ OrderLicensing) .
        enrich $\mathcal{X}$ $+$ Subordinates $+$ WellFormed $+$ Projections $+$ Indexes **by**
            axiom $\forall x\!:\!\texttt{Word}^* . \, \texttt{well\_formed}(x) \leftrightarrow \exists r\!:\!\texttt{Entry}, y\!:\!\texttt{EntryString} .$
                $|y| = \texttt{subordinates}(r) \land x \sim y \land \texttt{well\_formed}(y)$

In this way we transfer the need to impose order from $\texttt{Word}^*$ to $\texttt{Entry}^*$. Every instantiation of DependencyGrammar is a refinement of MultisetLicensing. Further, DependencyGrammar is a parameterised implementation, in that it maps a single model of Lexicon, Dependency, and OrderLicensing, to a single model[3] of WellFormed. We can make another refinement which illustrates the way in which the process of stepwise refinement can be used to encompass the idea of Chomsky ([3]) that the task of linguistics is to define the *space* of potential human languages, somewhere between the context-free and type zero: what Chomsky calls Universal Grammar. In this case, we incorporate the claim implicit in most accounts of linear precedence (e.g. [4]) that the predicate $\texttt{well\_formed}$ on $\texttt{Entry}^*$ can be characterised by a binary relation on $\texttt{Entry}$.

BinaryOrder $=$
    enrich OrderLicensing **by**
        axiom $\forall w\!:\!\texttt{Word}^* . \, \texttt{well\_formed}(w) \leftrightarrow (\forall xyz\!:\!\texttt{Entry}^*, rs\!:\!\texttt{Entry} .$
            $w = x \cdot r \cdot y \cdot s \cdot z \rightarrow \texttt{well\_formed}(r \cdot s))$

BinaryDependencyGrammar $=$
    $\lambda\mathcal{X}$: (Lexicon $+$ Dependency $+$ BinaryOrder) . $\mathcal{X}$ $+$ DependencyGrammar($\mathcal{X}$)

Any BinaryDependencyGrammar($\mathcal{X}$) is a refinement of DependencyGrammar($\mathcal{X}$), for the same $\mathcal{X}$.

[3] up to isomorphism

4 English Word Order

In order to produce a specific grammar it is now necessary only to specify a model of Lexicon, Dependency, and BinaryOrder. It will be useful in specifying conditions on the order of entries to have syntax for the binary relation between an entry and a subordinate. This is just a convenience which will serve to make subsequent specifications a little shorter. Also, for English and Dutch, it will be useful to model the concept of grammatical subject.

```
Subordinacy =
    enrich Subordinates by
        pred _ ≫ _ ⊆ Entry × Entry
        axiom $r \gg s \leftrightarrow \exists x{:}\mathtt{Entry}^{+}\,.\,\mathtt{subordinates}(r) = s + x$
Subject =
    sorts Noun,Verb,Entry
    axiom Noun, Verb ⊆ Entry
    opn subject : Verb → Noun
EnglishOrder =
    enrich Subject + Subordinacy + BinaryOrder by
        axiom $\forall v{:}\mathtt{Verb}, n{:}\mathtt{Noun}, rst{:}\mathtt{Entry}, x{:}\mathtt{Entry}^{+}\,.$
          $v \gg r \wedge n \gg s \wedge \mathtt{dependents}(t) = v + n + x$
           $\rightarrow \neg\mathtt{well\_formed}(r \cdot s)$
        axiom $\forall v{:}\mathtt{Verb}, n{:}\mathtt{Noun}, rs{:}\mathtt{Entry}\,.$
          $n = \mathtt{subject}(v) \wedge n \gg s \wedge v \gg r \wedge$
          $\mathtt{well\_formed}(r \cdot s) \rightarrow n \gg r$
        axiom $\forall v{:}\mathtt{Verb}, r{:}\mathtt{Entry}, x{:}\mathtt{Entry}^{+}\,.$
          $v \gg r \wedge v \neq r \wedge \mathtt{well\_formed}(r \cdot v)$
           $\rightarrow \mathtt{subject}(v) \gg r$
```

The three axioms here give restrictions on models of linear precedence in English. This is a specification in the broad, and the restrictions are a good deal stronger than will be required for our simple examples, but could be further strengthened by enrichment with additional axioms should that prove necessary as we investigate the space of grammars through more sophisticated accounts of fragments of English. The first axiom says that where both a noun and a verb occur as dependents of some entry, the noun and all its subordinates come before the verb and all of its subordinates. The second says that the subject of a verb, and all its subordinates, must precede the verb, and all its (other) subordinates. The last axiom says that the only subordinates of the verb which may precede the verb are subordinates of the subject.

5 Toward Implementation

One way to proceed to an implementation of EnglishOrder is to specify that `well_formed` admit *all* pairs not excluded by the axioms of EnglishOrder.

EnglishOrderModel =
  enrich Subject + Subordinacy + BinaryOrder **by**
    axiom $\forall rs\!:\!\texttt{Entry}\,.\,\neg\texttt{well\_formed}(r \cdot s) \leftrightarrow \exists v\!:\!\texttt{Verb}\,.$
      $(\exists n\!:\!\texttt{Noun}, t\!:\!\texttt{Entry}, x\!:\!\texttt{Entry}^+\,.$
        $v \gg r \wedge n \gg s \wedge \texttt{dependents}(t) = v + n + x)$
      $\vee\ (\exists n\!:\!\texttt{Noun}\,.\,n = \texttt{subject}(v) \wedge n \gg s \wedge v \gg r \wedge \neg n \gg r)$
      $\vee\ (s = v \wedge v \gg r \wedge v \neq r \wedge \neg\texttt{subject}(v) \gg r)$

Having a model of word order, it remains (for English) to model Lexicon, which is concerned with the form of words, and Dependency. Like word order, Lexicon will be highly language dependent, but Dependency, at least across the specific languages under consideration, might be a candidate for cross-linguistic specification.

Subcategorisation =
  initial
    sorts Noun,Verb,Entry
    axiom $\texttt{Noun}, \texttt{Verb} \subseteq \texttt{Entry}$
    opns $\texttt{proper\_noun} :\to \texttt{Noun}$
    opn $\texttt{intransitive} : \texttt{Noun} \to \texttt{Verb}$
    opn $\texttt{object\_transitive} : \texttt{Noun} \times \texttt{Noun} \to \texttt{Verb}$
    opn $\texttt{sentential\_transitive} : \texttt{Noun} \times \texttt{Verb} \to \texttt{Verb}$

Dependency1 =
  enrich Subcategorisation + Dependency **by**
    axiom $\texttt{dependents}(\texttt{proper\_noun}) = 0$
    axiom $\texttt{dependents}(\texttt{intransitive}(n)) = n$
    axiom $\texttt{dependents}(\texttt{object\_transitive}(n, n')) = n + n'$
    axiom $\texttt{dependents}(\texttt{sentential\_transitive}(n, v)) = n + v$

Dependency1 may seem like a useful implementation of Dependency. Unfortunately, it will not be useful in modelling Dependency+BinaryOrder. For instance, wanting subjects to precede verbs would imply we want

$$\texttt{well\_formed}(\texttt{proper\_noun} \cdot \texttt{object\_transitive}(\texttt{proper\_noun}, \texttt{proper\_noun}))$$

but *not* wanting *objects* to precede verbs would imply the opposite. One direction to try to remedy this situation might be to go for a more localised treatment of order, but this is difficult and arguably unintuitive, especially in the broader specifications, and especially in dealing with the cross-dependencies in Dutch. The alternative is to insist that different entries are used for each word.

DistinguishedDependents =
  enrich Subordinates **by**
    axiom $\forall rst\!:\!\texttt{Entry}, x\!:\!\texttt{Entry}^+\,.$
      $\texttt{subordinates}(r) = s + t + x \to s \neq t$

This is an example of how attempts at implementation can inform the task of specification-in-the-broad. How now might we proceed to implement DistinguishedDependents? One way is to employ grammatical functions such as `subject` as constructors of new entries.

```
Roots =
    derive from Subcategorisation
    by [$\texttt{Root} \longmapsto \texttt{Entry}, \texttt{N} \longmapsto \texttt{Noun}, \texttt{V} \longmapsto \texttt{Verb}$]
SomeEntries =
    extend Roots by
        sorts Entry,Noun,Verb
        opn $\texttt{root} : \texttt{Entry} \to \texttt{Root}$
        opn $\texttt{subject} : \texttt{Verb} \to \texttt{Noun}$
        opn $\texttt{object} : \texttt{Verb} \overset{\cdot}{\to} \texttt{Noun}$
        opn $\texttt{clause} : \texttt{Verb} \overset{\cdot}{\to} \texttt{Verb}$
        axiom $\texttt{Root} \subseteq \texttt{Entry}$
        axiom $\texttt{N} \subseteq \texttt{Noun}$
        axiom $\texttt{V} \subseteq \texttt{Verb}$
        axiom $\forall r{:}\texttt{Root} \,.\, \texttt{root}(r) = r$
        axiom $\texttt{root}(s) = \texttt{intransitive}(n) \to \texttt{root}(\texttt{subject}(s)) = n$
        axiom $\texttt{root}(s) = \texttt{object\_transitive}(n, n')$
            $\to \texttt{root}(\texttt{subject}(s)) = n \wedge \texttt{root}(\texttt{object}(s)) = n'$
        axiom $\texttt{root}(s) = \texttt{sentential\_transitive}(n, v)$
            $\to \texttt{root}(\texttt{subject}(s)) = n \wedge \texttt{root}(\texttt{clause}(s)) = v$
ADependencyModel =
    enrich SomeEntries+Dependency by
        axiom $\texttt{root}(s) = \texttt{proper\_noun} \to \texttt{dependents}(s) = 0$
        axiom $\texttt{root}(s) = \texttt{intransitive}(n) \to \texttt{dependents}(s) = \texttt{subject}(s)$
        axiom $\texttt{root}(s) = \texttt{object\_transitive}(n, n')$
            $\to \texttt{dependents}(s) = \texttt{subject}(s) + \texttt{object}(s)$
        axiom $\texttt{root}(s) = \texttt{sentential\_transitive}(n, n')$
            $\to \texttt{dependents}(s) = \texttt{subject}(s) + \texttt{clause}(s)$
```

We rename the sorts of Subcategorisation to allow them to become the starting points for larger sorts in SomeEntries. Note that $\overset{\cdot}{\to}$ denotes a partial operation, which in a free extension will be defined only at points where the axioms insist that it be so.

In ADependencyModel, the dependents of

$\texttt{sentential\_transitive}(\texttt{proper\_noun}, \texttt{intransitive}(\texttt{proper\_noun}))$

are

$\texttt{subject}(\texttt{sentential\_transitive}(\texttt{proper\_noun}, \texttt{intransitive}(\texttt{proper\_noun})))$

and

$\texttt{clause}(\texttt{sentential\_transitive}(\texttt{proper\_noun}, \texttt{intransitive}(\texttt{proper\_noun})))$,

and to complete the collection of subordinates, you must further add

$\texttt{subject(clause(sentential\_transitive(proper\_noun,}$
$\quad\texttt{intransitive(proper\_noun)))).}$

The operation `root` maintains a relationship between each new entry and the original ones inhabiting `Root`, which is necessary, for instance, so that we know that

$\texttt{object(clause(sentential\_transitive(proper\_noun,}$
$\quad\texttt{intransitive(proper\_noun))))}$

is not defined, because

$\texttt{root(clause(sentential\_transitive(proper\_noun,}$
$\quad\texttt{intransitive(proper\_noun)))) = intransitive(proper\_noun).}$

Clearly it is somewhat unsatisfactory to have to independently specify that for each distinct root constructor, `dependents` maps the result of its application to the multiset formed by the further application of whichever grammatical functions are defined there. A more satisfactory solution would be to employ a specialised *institution*, in which the concept of grammatical function, and the definition of `dependents`, is built into the underlying logic. An institution is a formalisation of what it means to be a (first-order) logic, defined in terms of category theory ([5]). In [7], I define some example institutions specialised for use in a linguistic setting. It might also seem preferable to find a way to build the `subject` operation into the specialised institution, as always corresponding to the first argument of the verb, but perhaps not, since one might want to be able to deal with a verb like *promise*, in *Harold promised to swim* along these lines:

$$\begin{array}{l}\textbf{opn}\ \texttt{promise\_transitive} : \mathsf{V} \rightarrow \mathsf{V}\\ \textbf{axiom}\ \texttt{root}(s) = \texttt{promise\_transitive}(v)\\ \qquad \rightarrow \texttt{root}(\texttt{subject}(s)) = \texttt{root}(\texttt{subject}(v))\end{array}$$

ParametricGermanic is a parametric grammar aimed at dependent clause order for a fragment of English, Dutch and German.

$$\begin{array}{l}\mathsf{ParametricGermanic} =\\ \quad \lambda\mathcal{X} : (\mathsf{Lexicon} + \mathsf{BinaryOrder} + \mathsf{SomeEntries})\,.\\ \qquad \mathsf{BinaryDependencyGrammar}(\mathcal{X} + \mathsf{ADependencyModel})\end{array}$$

The appearance of SomeEntries "typing" the parameter is necessary to say that this is the space of entries on which Lexicon and BinaryOrder are to operate. [4] In order to produce a specific grammar, it is now only necessary to furnish a model of Lexicon and BinaryOrder over that space, that is, to give a lexicon corresponding to the lexical entries, and to give the appropriate restrictions on their ordering.

[4] Technically, because the argument to BinaryDependencyGrammar has more syntax than that required by its definition, one must employ the **derive** operation to "forget" the extra syntax, then add the argument to the result with + to get it back. Because it is obvious where this is needed, I take it as read.

6 Grammars

EnglishLexicon takes care of associating a particular (English) lexical item with every entry.

EnglishLexicon $=$
  extend SomeEntries **by**
    sort $\texttt{Word}$
    opn $\_ \sim \_ \subseteq \texttt{Word} \times \texttt{Entry}$
    opns $\texttt{michael}, \texttt{harold}, \texttt{swim}, \texttt{saw} :\rightarrow \texttt{Word}$
    axiom $\texttt{root}(s) = \texttt{proper\_noun} \rightarrow \texttt{michael} \sim s$
    axiom $\texttt{root}(s) = \texttt{proper\_noun} \rightarrow \texttt{harold} \sim s$
    axiom $\texttt{root}(s) = \texttt{intransitive}(n) \rightarrow \texttt{swim} \sim s$
    axiom $\texttt{root}(s) = \texttt{object\_transitive}(n, n') \rightarrow \texttt{saw} \sim s$
    axiom $\texttt{root}(s) = \texttt{sentential\_transitive}(n, v) \rightarrow \texttt{saw} \sim s$

Now ParametricGermanic(EnglishLexicon+EnglishOrderModel) is a specific grammar covering dependent clauses such as *Michael saw Harold swim.* For German and Dutch, I will not bother with the broader specification, *a la* EnglishOrder, but just give the model specifications.

GermanLexicon $=$
  derive from EnglishLexicon
    by $[\texttt{schwimmen} \longmapsto \texttt{swim}, \texttt{sah} \longmapsto \texttt{saw}]$

GermanOrderModel $=$
  enrich ADependencyModel + BinaryOrder **by**
    axiom $\neg\texttt{well\_formed}(r \cdot s) \leftrightarrow \exists v : \texttt{Verb}\,.$
      $(\exists n : \texttt{Noun}, t : \texttt{Entry}, x : \texttt{Entry}^{+}\,.$
        $v \gg r \wedge n \gg s \wedge \texttt{dependents}(t) = v + n + x)$
      $\vee\ (r = v \wedge v \gg s \wedge v \neq s)$

The fact that we are able to use for German, and later Dutch, the same dependency system ADependencyModel used for English, is partly a reflection of the closeness of the languages (though of course it also has a lot to do with the triviality of the example). However we need a different precedence relation, and of course the actual words are different. The first disjunct of the axiom of GermanOrderModel is the same as for EnglishOrderModel. The second says that a verb succeeds all its subordinates. ParametricGermanic(GermanLexicon+GermanOrderModel) is a specific grammar covering dependent clauses such as *Michael Harold schwimmen sah.*

DutchLexicon $=$
  derive from EnglishLexicon
    by $[\texttt{zwemmen} \longmapsto \texttt{swim}, \texttt{zag} \longmapsto \texttt{saw}]$

DutchOrderModel $=$
  enrich ADependencyModel + BinaryOrder **by**
    axiom $\neg\texttt{well\_formed}(r \cdot s) \leftrightarrow \exists v : \texttt{Verb}\,.$

$$\begin{aligned}&(\exists n{:}\texttt{Noun}, t{:}\texttt{Entry}, x{:}\texttt{Entry}^+ .\\ &\quad v \gg r \wedge n \gg s \wedge \texttt{dependents}(t) = v + n + x)\\ &\vee (\exists n{:}\texttt{Noun} . n = \texttt{subject}(v) \wedge n \gg s \wedge v \gg r \wedge \neg n \gg r)\\ &\vee (\exists x{:}\texttt{Entry}^+ . r = v \wedge \texttt{dependents}(s) = v + x)\end{aligned}$$

The first two disjuncts of the axiom are the same as for EnglishOrderModel. The last says that dependent verbs follow their head.

ParametricGermanic(DutchLexicon+DutchOrderModel) is a specific grammar covering dependent clauses such as *Michael Harold zag zwemmen.*

7 Dependency Constituents

What allows us to assign constituent structures to the English

that [Michael$_1$ saw$_1$ [Harold$_2$ swim$_2$]$_1$]

and the German

dass [Michael$_1$ [Harold$_2$ schwimmen$_2$]$_1$ sah$_1$]

but not the Dutch

dat Michael$_1$ Harold$_2$ zag$_1$ zwemmen$_2$

is the fact that in English and German (at least in these fragments), all words subordinate to any particular word must appear contiguously. We can express this by insisting that treatments of English and German must refine the following specification:

```
AdjacencyCondition =
    **enrich** Subordinacy + OrderLicensing **by**
        **axiom** $\texttt{well\_formed}(w \cdot r \cdot x \cdot s \cdot y \cdot t \cdot z) \wedge r \gg t \rightarrow r \gg s$
        **axiom** $\texttt{well\_formed}(w \cdot t \cdot x \cdot s \cdot y \cdot r \cdot z) \wedge r \gg t \rightarrow r \gg s$
```

This just says that, in an acceptable sequence of entries, if t is subordinate to r, and s occurs between them, then s must also be subordinate to r. From any dependency grammar refining AdjacencyCondition, we can derive a constituency (phrasal) grammar with the same coverage. We need a space of entities which, like `Entries`, takes care of ambiguity, but can also distinguish words and phrases.

```
DependencyConstituents =
    **extend sorts** Entry **by**
        **sort** Syn
        **opn** phrasal : Entry → Syn
        **opn** lexical : Entry → Syn
        **opn** |_| : Syn → Entry
        **axiom** $|\texttt{phrasal}(r)| = r$
        **axiom** $|\texttt{lexical}(r)| = r$
```

Syn has two copies of each Entry, one for the word, which will form a leaf in parse trees, and one for the phrase formed by the word and all its subordinates.

```
SynMultisets =
    derive from Multisets by[Syn ⟼ Entry, Syn⁺ ⟼ Entry⁺]
PhrasalMultisets =
    extend DependencyConstituents + Multisets + SynMultisets by
        opn phrasal : Entry⁺ → Syn⁺
        axiom phrasal(0) = 0
        axiom phrasal(x + y) = phrasal(x) + phrasal(y)
DependencyDominance =
    extend PhrasalMultisets+Dependents by
        pred immediate_dominance ⊆ Syn × Syn⁺
        axiom immediate_dominance(phrasal(r),
            lexical(r) + phrasal(dependents(r)))
```

PhrasalMultisets extends phrasal across multisets, allowing us in DependencyDominance to encode immediate dominance by saying that a phrasal entity dominates the multiset consisting of the phrasal entities of its dependents, plus the corresponding lexical entity.

```
SynStrings =
    derive from Strings by[Syn ⟼ Word, Syn* ⟼ Word*]
BackMap =
    extend DependencyConstituents + EntryStrings + SynStrings by
        opn |_| : Syn* → Entry*
        axiom |e| = e
        axiom |x · y| = |x| · |y|
SynOrder =
    extend AdjacencyCondition+BackMap by
        pred well_formed ⊆ Syn*
        axiom ∀x:Syn* . well_formed(|x|) → well_formed(x)
DependencyConstituency =
    extend DependencyDominance + SynOrder by
        pred constituency ⊆ Syn × Syn*
        axiom immediate_dominance(r, x) ∧ well_formed(x)
          → constituency(r, x)
```

BackMap extends |_| across strings. This allows SynOrder to define well_formed on Syn* from its definition on Entry*. DependencyConstituency then puts together these accounts of immediate dominance and linear precedence in the standard way to give a definition of constituency.

```
SynLexicon =
    extend Lexicon+DependencyConstituents by
        pred _ ∼ _ ⊆ Word × Syn
        axiom $\forall r$:Entry . $w \sim r \rightarrow w \sim$ lexical$(r)$
SynIndexes =
    derive from Indexes by [Syn $\longmapsto$ Entry, Syn$^*$ $\longmapsto$ Entry$^*$]
Parse =
    extend DependencyConstituency by
        pred parse $\subseteq$ Syn$^*$ $\times$ Syn$^*$
        axiom parse$(x, x)$
        axiom constituency$(r, x) \rightarrow$ parse$(r, x)$
        axiom parse$(x, y) \wedge$ parse$(x', y') \rightarrow$ parse$(x \cdot x', y \cdot y')$
DCGrammar =
    $\lambda \mathcal{X}$ : (Lexicon + Dependency + OrderLicensing + AdjacencyCondition) .
        enrich $\mathcal{X}$ + WellFormed + SynLexicon + SynIndexes + Parse by
          axiom well_formed$(x) \leftrightarrow$
                  $\exists r$:Syn, $y$:Syn$^*$ . parse$(r, y) \wedge x \sim y$
```

If a grammar G specifies (up to isomorphism) a model of Lexicon+Dependency+ OrderLicensing + AdjacencyCondition, then DCGrammar(G) licenses the same strings as DependencyGrammar(G).

8 Coda

In this article I have attempted to demonstrate the utility of the algebraic specification language ASL in so-called *grammar engineering*. The class-of-model semantics employed by ASL makes it useful both in tracking the broad notion of what it means to be a solution, as well as in producing specific grammars which implement the broad concept. I have illustrated how parametrised specification may be used to embody the use of the same construction in different contexts, including across languages. This, and the notion of stepwise refinement, are seen to accord well with the principles-and-parameters approach of contemporary linguistics.

References

[1] Bloom, S. L. and E. G. Wagner, *Many-sorted theories and their algebras with some applications to data types*, Chapter 4 in Nivat, M. and Reynolds, J. C. (eds.) "Algebraic methods in semantics", pp134-168, CPU, 1985.

[2] Bresnan, J., R.M. Kaplan, S. Peters and A. Zaenan, *Cross-serial dependencies in Dutch*, Linguistic Inquiry **13** (1982), 613-635.

[3] Chomsky, N. "Knowledge of Language: Its Nature, Origin and Use", Praeger, New York, 1986.

[4] Gazdar, G., E. Klein, G. Pullum and I. Sag, "Generalized Phrase Structure Grammar", Basil Blackwell, London, 1985.

[5] Goguen, J. A. and R. M. Burstall, *Institutions: Abstract Model Theory for Computer Science*. Report No. CSLI-85-30, Center for the Study of Language and Information, August, 1985.

[6] Matthews, P.H. “Syntax”, Cambridge University Press, 1981.

[7] Newton, M. “Algebraic Specification of Grammar”, Ph.D. thesis, Centre for Cognitive Science, University of Edinburgh, 1992.

[8] Pollard, C. and I. Sag “An Information-Based Approach to Syntax and Semantics: Volume 1 Fundamentals”, Center for the Study of Language and Information, Stanford, Ca., 1987.

[9] Reape M., *A theory of word order and discontinuous constituency in West Continental Germanic*, in E. Engdahl and M. Reape, eds., “Parametric Variation in Germanic and Romance: Preliminary Investigations”, DYANA Report R1.1.A, Centre for Cognitive Science, University of Edinburgh, pp. 1-7, 1990.

[10] Sannella, D.T. and M. Wirsing, *A kernel language for algebraic specification and implementation*, Report No. CSR-131-83, Dept. of Computer Science, Univ. of Edinburgh, 1983.