27 Feature Grammars and Unification

Drew McDermott `drew.mcdermott@yale.edu`

2015-11-20, 11-30, 2016-10-28 — Yale CS470/570

1 Rules in a Feature Grammar

Grammar rules look like this example (using Shieber's PATR-II notation, but coming from a different angle):

```
S → NP VP
  S⟨head-daughter⟩ = VP
  S⟨head⟩ = VP⟨head⟩
  S⟨head subject⟩ = NP⟨head⟩
  VP⟨head agreement⟩ = NP⟨head agreement⟩
```

After the familiar rule in line 1, we have some constraints. The details here are motivated by linguistic theory, in which the term "head" is central, as we'll see. The rule says you can build an F-structure for S out of two F-structures, one for an NP and one for a VP, provided that these path equalities hold. The path equalities are constraints, imposed by unification, in a way that will become clear shortly.

For a CFG we imagine "rewriting" `S` as "`NP VP`". Here we imagine starting with an F-structure with just one slot:

(1) `{cat: S}`

then moving to a new F-structure

```
(2) {cat: S, daughters: <{cat: NP, head: {cat: N}},
                         {cat: VP, head: {cat: V}}>}
```

One may well ask how all this structure emerged from "NP VP," but any linguist would insist that an xP is defined as the "maximal projection of an x," which in the terminology we're relying on here means a phrase whose head word is of category x. Because we started with a blank slate, working top-down, any further information must emerge from constraints supplied by the grammar and the lexicon. ("Daughters," sometimes abbreviated "dtrs," is synonymous with the phrase's *constituents*, the subphrases that it comprises.)

The first constraint is

`S⟨head-daughter⟩ = VP`

In this context, imposing the constraint means moving to this data structure:

```
{cat: S,
 daughters: <{cat: NP, head: {cat: N}},
             =1={cat: VP, head: {cat: V}}>,
 head-daughter: =1}
```

So that the `head-daughter` feature is now synonymous with the second daughter. The next constraint is that "The `head` of S must the same as the substructure labeled "`head`" in the `VP`." To reflect this constraint, we move to this F-structure:

```
{cat: S,
 daughters: <{cat: NP, head: {cat: N}},
             =1={cat: VP, head: =2={cat: V}}>,
 head-daughter: =1,
 head: =2}
```

Are we allowed to add index numbers to an existing structure this way? Actually, all we added was new features with new arcs to existing subgraphs. Remember that the index numbers' sole function is to provide a way to describe DAGs in a convenient non-graphical notation. They are labels, not real parts of the data structure.

The last two constraints

```
S⟨head subject⟩ = NP⟨head⟩
VP⟨head agreement⟩ = NP⟨head agreement⟩
```

result in this F-DAG:

```
{cat: S,
 daughters: <{cat: NP, head: =3={cat: N, agreement: =4}},
             =1={cat: VP, head: =2={cat: V,
                                    subject: =3,
                                    agreement: =4={?}}}>,
 head-daughter: =1,
 head: =2}
```

There are two things we need to do to get our unification algorithm (see next note) to build this structure for us. The first is to understand the right-hand side of the rule "S $\rightarrow$ NP VP" to set up F-Dag (2), and to store the equalities

$$\begin{aligned} \mathrm{S} = &\quad G\langle\rangle \\ \mathrm{NP} = &\quad G\langle\texttt{daughters, first}\rangle \\ \mathrm{VP} = &\quad G\langle\texttt{daughters, rest, first}\rangle \end{aligned}$$

Imposing each constraint requires creating a minimal graph that extends (2) just enough to make the constraint true. For example, if we start with the last constraint

$$\texttt{VP}\langle\texttt{head agreement}\rangle = \texttt{NP}\langle\texttt{head agreement}\rangle$$

we can build this F-DAG:

```
{daughters: <{head: {agreement: =1= {?}}},
             {head: {agreement: =1}}>}
```

When we unify this F-DAG with (2), which is left as an exercise for the reader, we get a graph that reflects the structural assumption and this agreement requirement, even before we stick in any information about what the syntactic categories of the first and second daughter are. It doesn't matter what order a series of unifications is done in; you'll get the same result (up to orderings, perhaps, and index numbers, which are artifacts of the printing process).

[[From here on the notes are inconsistent, although interesting; they haven't been updated from last year's notation, which I made adjustments to.]]

The `subject` of the `head` of S must be the `head` of the `NP`. So if we start with the `NP`

```
{cat: NP, head: {agreement: {number: singular, person: third}}}
```

and the `VP`

```
{cat: VP, head: {form: finite,
                 subject: {agreement: {number: singular,
                                       person: third}}}
```

then we wind up with

```
{cat: S, head: =1}
{cat: NP, head: =2}
{cat: VP, head: =1{form: finite,
                   subject: =2{agreement: {number: singular,
                                           person: third}}}}
```

To make all this happen, all we had to do was tie together existing pieces of DAG, which we got away with because they were actually equal. For instance, $\mathtt{S}\langle\mathtt{head}\rangle = \mathtt{VP}\langle\mathtt{head}\rangle$ and $\mathtt{NP}\langle\mathtt{head}\rangle = \mathtt{S}\langle\mathtt{head\ subject}\rangle = \mathtt{VP}\langle\mathtt{head\ subject}\rangle$. If two subDAGs are not identically equal, then we must replace both of them with their *most general unification* (MGU).

Shieber explains unification pretty well. In a nutshell (or two nutshells):

- If F_1 and F_2 are two F-structures, F_1 subsumes F_2 (written $F_1 \sqsubseteq F_2$) if F_2 "contains at least as much information" as F_1, or F_1 "is at least as general" as F_2.[1] More precisely: (a) the paths through F_1 are a subset of the paths through F_2; and (b) at all the paths p they have in common, $F_1\,p \sqsubseteq F_2\,p$;[2] and (c) for all pairs of paths p_1 and p_2, if $F_1\,p_1 = F_1\,p_2$ then $F_2\,p_1 = F_2\,p_2$. Condition (c) is to make sure that the graph structure of F_2 preserves the graph structure of F_1.

- The MGU of two F-structures F_1 and F_2, written $F_1 \sqcup F_2$, is the least specific F-structure subsumed by F_1 and F_2. That is, if $F_u = F_1 \sqcup F_2$, then (a) $F_1 \sqsubseteq F_u$ and $F_2 \sqsubseteq F_u$; and (b) if F'_u satisfies (a) as well, then $F_u \sqsubseteq F'_u$.[3]

There is no guarantee that there is an F-structure meeting part (a) of the definition of $F_1 \sqcup F_2$, but if there is, then there is always an F-structure satisfying part (b).[4]

If we return to our grammar, a rule

[1] Shieber says "more information" and "more general," and these are the cases we're interested in, but he's careful to define $\sqsubseteq$ so it's reflexive, and his notation indicates that: He chose "$\sqsubseteq$" instead of "$\sqsubset$."

[2] This recursive definition bottoms out because the substructures in question are smaller; eventually we get to atomic values.

[3] In fewer words, $\sqcup$ defines a semilattice, and $F_1 \sqsubseteq F_2$ iff $F_1 \sqcup F_2 = F_2$.

[4] Much of the terminology regarding unification originated in the literature on mechanical theorem proving, but one's intuition about how to transpose concepts from one to the other can be wrong. (Those not familiar with this literature can skip this footnote.) There is a temptation to identify indices with variable names in theorem proving, but they're not the same. In theorem proving, the output of the unification algorithm is a *substitution,* i.e., a specification of the values of some of the variables. Substitutions are viewed as functions on formulas, which replace some of their variables. When unifying two F-structures, no attempt is made to separate the substitution from the act of applying it, that is, actually replacing variables and other underspecified substructures with new versions with slots filled in or replaced with better versions. So no substitution is ever brought into being when F-structures are unified.

Although Shieber uses the term "variable" for the empty F-structure, that's just a common but special case: a node with no edges that when unified acquires all the edges of the node it's paired with. In theorem proving, variables have to have names to make unification "interesting," because a variable that occurs only once never fails to match

$$
\begin{array}{l}
C_0 \rightarrow C_1 \dots C_n \\
\quad (\Gamma_1\, p_1 = \Gamma_2\, p_2)^*
\end{array}
$$

means that if F-structure X_i describes a list of words w_i, with $X_i\langle \texttt{cat}\rangle = C_i$, for $1 \le i \le n$, then X_0 describes w_1 ::: w_2 ::: ... ::: w_n (using the Scala "append" operation ":::"), provided that the constraints, each of the form $\Gamma_1\, p_1 = \Gamma_2\, p_2$, are satisfied, where Γ_1, Γ_2 are two (not necessarily distinct) elements of $\{X_0, \dots, X_n\}$ and p_1 and p_2 are paths. (The "*" means that we can have zero or more such constraints.) In what follows, I'll continue to use X_i to refer to the i'th F-structure mentioned in a rule; $X_i\, \langle cat\rangle = C_i$.[5]

Where in our rule does it say that the C_i describe phrases in contiguous segments that together make a C_0? Nowhere, so let's fix that. Introduce a feature `span`, whose value is an F-structure with two features `begin` and `end` that describe points in a word string.[6]

A point in a word string is, as usual, denoted by the number of words to its left, so that a string of N words has $N+1$ "points": 0 through N. An analysis of the entire string will span points 0 to N.

Now we'll take our rule to imply the constraints:[7]

$$
\begin{array}{l}
C_i\langle \texttt{span end}\rangle = C_{i+1}\langle \texttt{span begin}\rangle \;\; (1 \le i \le n-1) \\
C_0\langle \texttt{span begin}\rangle = C_1\langle \texttt{span begin}\rangle \\
C_0\langle \texttt{span end}\rangle = C_n\langle \texttt{span end}\rangle
\end{array}
$$

At this point we really must explain where the words are coming from. There's a module delivering a stream[8] of F-structures describing each word. A sentence beginning "She kept..." would produce something like this:

```
{cat: pronoun, lemma: she, span: {begin: 0, end: 1},
 agreement: {number:  singular, person:  3rd, gender:  fem},...}
{cat: verb, lemma: keep, tns: past, span: {begin: 1, end: 2}, ...}
```

anything. In the world of F-structures, multiple paths to the same graph node play the role of multiple occurrences of the same variable name in theorem proving.

[5] If, say, two noun phrases (NPs) occur on the right-hand side of a rule, we would call them $\texttt{NP}_1$ and $\texttt{NP}_2$. But this doesn't affect the X_i numbering scheme, which just orders from left to right.

[6] Or a string of phonemes or morphemes. Actually, even for text we should use the neutral term *lexeme*, which allows for the possibility that in some preprocesing step a word might get broken into pieces (or otherwise altered), so that (e.g.) "keeping" might become "keep" + "-ing" and "kept" might become "keep" + "-ed," (although in what follows we prefer to let lexemes be words with features indicating what endings they had).

[7] Actually, we don't need most of these, except to keep track of the beginning and end of phrases we've found. The parsing algorithm (we hope!) will not try to put non-contiguous phrases together.

[8] But not necessarily a `Stream`.

The *lemma* of a word is its "root," the "unmarked" form from which other formas might be derived by adding prefixes or suffixes indicating number, case, tense, or whatever features are appropriate for the word category in question. In the interest of brevity, I've omitted many of these features.

The parsing problem is to generate an F-structure with `cat:` S that spans the entire sentence. Each F-structure above the word level is licensed by some grammar rule, in that the `span` constraints and the explicit constraints are satisfied.

The constraint notation raises some new questions. It duplicates the functionality supplied by the coindexation notation, in that both tell us which pieces have to be the same. Furthermore, while we explained what unification meant for two F-structures, we didn't explain what it meant for a bunch of F-structures and constraints.

To provide all these explanations, we treat the constraint notation as syntactic sugar for the coindexation notation we already have. Given our typical rule:

$$
\begin{array}{l}
C_0 \rightarrow C_1 \ldots C_n \\
\quad X_{i_1}\, p_{11} = X_{j_1}\, p_{12} \\
\quad X_{i_2}\, p_{21} = X_{j_2}\, p_{22} \\
\quad \ldots \\
\quad X_{i_m}\, p_{m1} = X_{j_m}\, p_{m2}
\end{array}
$$

we'll construct a feature structure F_R that contains the same information. See figure 1.

There are two aspects of this figure that bear further explanation. First, the components of the top-level F-structure, `X0` through `X`n, are interrupted by F-structures for `X`i_1 and `X`j_1. These are not additional components; i_1 might be 2 and j_1 might be 3, so that the piece labeled `X`i_1 is the same as the piece labeled `X2` that appears just above it, and the piece labeled `X`j_1 is the component next in the sequence. The caption of figure 1 explains the triple subscripts on p that identify labels along paths p_{11} and p_{12}. The only way to mention a path in the coindexation notation is to provide it, so all the frames along the way are filled in, even if only skeletally. When we reach the end of each path we put the label `=`$(n+1)$ to indicate that they must be the same.[9] One of these paths may end in an already known structure; if not, then one is picked at random and made a variable.[10] Only an example

[9] The use of numbers rather than identifiers to indicate coindexation gets tiresome at this point.

[10] It's not allowed for two of them to end in nontrivial structure; instead, unify the two and replace them with the result. If they fail to unify, the rule makes no sense.

$$
\begin{array}{l}
\{\texttt{X0:}\ \{\texttt{cat:}\ C_0\texttt{,} \\
\quad\quad \texttt{daughters:}\ \{\texttt{first:}\ \texttt{=1,} \\
\quad\quad\quad\quad\quad\quad \texttt{rest:}\ \{\texttt{first:}\ \texttt{=2,} \\
\quad\quad\quad\quad\quad\quad\quad\quad \texttt{rest:}\ \{\ldots\{\texttt{first:}\ \texttt{=}n\texttt{,} \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad \texttt{rest:}\ \texttt{Nil}\}\}\}\}\}\texttt{,} \\
\ \texttt{X1:}\ \texttt{=1=}\{\texttt{cat:}\ C_1\}\texttt{,} \\
\ \texttt{X2:}\ \texttt{=2=}\{\texttt{cat:}\ C_2\}\texttt{,} \\
\ \ldots \\
\ \texttt{X}i_1\texttt{:}\ \{\texttt{:}\ p_{111}\{\ldots\{p_{11l_1}\texttt{:}\ \texttt{=}n+1\}\}\}\texttt{,} \\
\ \ldots \\
\ \texttt{X}j_1\texttt{:}\ \{p_{121}\texttt{:}\ \{\ldots\{p_{12m_1}\texttt{:}\ \texttt{=}n+1\}\}\}\texttt{,} \\
\ \ldots \\
\ \texttt{X}_n\texttt{:}\ \texttt{=}n\texttt{=}\{\texttt{cat:}\ C_n\}\}
\end{array}
$$

Figure 1: Schema for F_R, the F-structure for the rule with coindexation constraints, where $p_{11} = \langle p_{111}\, p_{112}\, \ldots\, p_{11l_1}\rangle$ and $p_{12} = \langle p_{121}\, p_{122}\, \ldots\, p_{12m_1}\rangle$,

will make this clear, but first let's explain more fully the notation we use for lists to avoid the clumsy `first-rest` notation.

With this abbreviation, figure 1 becomes figure 2.

Let's get back our running example, before abstraction overflows our skulls. The rule is `S` $\rightarrow$ `NP VP`. and the constraints are (this time using the X_i notation — X_0, X_1, X_2 — referring to the `S` (under construction), the `NP`, and the `VP`, respectively):

$$
\begin{array}{l}
X_0\langle\texttt{head}\rangle = X_2\langle\texttt{head}\rangle \\
X_0\langle\texttt{head subject}\rangle = X_1\langle\texttt{head}\rangle \\
X_1\langle\texttt{head agreement}\rangle = X_2\langle\texttt{head agreement}\rangle
\end{array}
$$

to which we'll add two more:

$$
\begin{array}{l}
X_0\langle\texttt{sem fcn}\rangle = X_1\langle\texttt{sem}\rangle \\
X_0\langle\texttt{sem arg1}\rangle = X_2\langle\texttt{sem}\rangle
\end{array}
$$

just to remind us that we're ultimately interested in the semantics (internal representation) or our sentence. The two extra constraints are the Montagovian $\texttt{sem}(X_0) = \texttt{sem}(X_1)(\texttt{sem}(X_2))$, expressed in "F-structurese." Figure 3 shows what F_R looks like.

Looking at figure 3, the most likely reaction seems to be, What happened to the arrow? The answer is that the arrow was always an illusion to some

```
{X0: {cat: C_0,
      daughters: <=1,
                  =2,
                  ...
                  =n>
 X1: =1= {cat: C_1},
 X2: =2= {cat: C_2},
 ...
 Xi_1: {p_111: {...{p_11l_1: =(n+1)}}},
 ...
 Xj_1: {p_121: {...{p_12m_1: =(n+1)}}},
 ...
 X_n: =n= {cat: C_n}}
```

Figure 2: Schema for F_R, with list notation (cf. figure 1)

```
{X0: {cat: S,
      daughters: <=1,=2>,
      span: {begin: =3= {?}, end: =4= {?}}
      sem: {class: funapp,
            numargs: 1,
            fcn: =5,
            arg1: =6},
      head: =7{subject: =8}}}
 X1: =1{cat: NP,
        span: {begin: =3, end: =9},
        sem: =5= {?},
        head: =8{agreement: =10= {?}}
 X2: =2{cat: VP,
        span: {begin: =9= {?}, end: =4},
        sem: =6= {?},
        head: =7= {agreement: =10}}}
```

We have added variables to make sure each of the labels 3, 4, 5, 6, 9, and 10 is defined in one of the two spots where it occurs.

Figure 3: F_R for `S` $\rightarrow$ `NP VP`

`{X0: {daughters: <=1, =2>},`
` X1: =1{cat:` C_1`,`
`        span: {begin:` i_0`, end:` i_1`},`
`        `—*everything else we know about the tree* X_1—`},`
` X2: =1{cat:` C_2`,`
`        span: {begin:` i_1`, end:` i_2`},`
`        `—*everything else we know about the tree* X_2—`},`
` ...`
` X`n`: =`n`= {cat:` C_2`,`
`          span: {begin:` i_{n-1}`, end:` i_n`},`
`          `—*everything else we know about the tree* X_n—`}},`

Figure 4: Schema for F_S, the "situational" F-structure

extent. It can be interpreted left-to-right, as if one is trying to generate the entire language.[11] But it can also be interpreted right-to-left, when the task is to decide whether a word list is in the language. Or inside-out and sideways. Grammar rules are better thought of as a system of constraints on phrase structures, and the unification formalism is ideal for this. Shieber's paper discusses some of the systems that had been developed as of the 1980s, and many more have been developed since. HPSG ("Head-driven phrase-structure grammar") is especially worthy of mention.

For F_R to do any work, it must be unified with something. If we're parsing, at a minimum what has to happen is that n contiguous phrases that have been found already must be packaged up into a structure F_S ("S" for "situation") that resembles F_R at the top level. See figure 4.

The values $i_0, \ldots, i_n$ are known quantities at this point, defining the boundaries of the phrases. F_S describes in detail the structures already found, but says nothing at all about the bigger structure to be built, except that it consists of these pieces. Obviously, most parsers know more about the bigger structure, because they wouldn't be considering putting these pieces together unless there was a rule justifying it. The point is that unification doesn't depend on the information being located in F_S if it's already in F_R.

Figure 5 is the schema of figure 4 instantiated for a particular analyzed word string, "Napoleon retreated."[12]

Now we can apply unification to our parsing problem. We unify F_R and

[11] Which is why the tradition started by Chomsky sixty years ago is known as "generative

```
{X0: {daughters: <{}=1, =2>{}},
 X1:=1{cat: NP,
       span: {begin: 0, end: 1},
       head: {cat: properNoun,
              lemma: "Napoleon",
              agreement: {person: 3rd, number: singular}},
       sem: {class: lambda,
             var: p,
             exp: {class: funapp,
                   fcn: p,
                   numargs: 1,
                   arg1: nb2}}},
 X2:=2{cat: VP,
       span: {begin: 1, end: 2},
       head: {cat: verb,
              lemma: "retreat"
              tns: past},
       sem: {class: lambda,
             var: x
             exp: {class: funapp,
                   fcn: retreat,
                   numargs: 1,
                   arg1: x}}}}
```

Figure 5: F_S for a particular situation: “Napoleon” and “retreated”

F_S. If they unify, $F_R \sqcup F_S$ describes the S spanning positions i_0 to i_2. In fact, for our example the unification succeeds, yielding the structure shown in figure 6.

grammar."

[12]The semantics ("**`sem`**") of the children are λ-expressions. The first would be written $\lambda(p)(p\,\texttt{nb2})$ in the λ-calculus. The atom `nb2` is the internal name for Napoleon Bonaparte; think of it as his "netid."

```
{X0: {daughters: <{}=1, =2>{},
      span: {begin: 0, end: 2},
      head:=11,
      sem: {class: funapp,
            fcn:=3,
            numargs=1,
            arg1:=4}},
 X1:=1{cat: NP,
       span: {begin: 0, end: 1},
       head:=12{cat: properNoun,
                lemma: "Napoleon",
                agreement:=14{person: 3rd, number: sing}},
       sem:=3{class: lambda,
              var: p,
              exp: {class: funapp,
                    fcn: p,
                    numargs: 1,
                    arg1: nb2}}},
 X2:=2{cat: VP,
       span: {begin: 1, end: 2},
       head:=11{cat: verb,
                lemma: "retreat",
                tns: past,
                agreement:=14,
                subject:=12},
       sem:=4{class: lambda,
              var: x
              exp: {class: funapp,
                    fcn: retreat,
                    numargs: 1,
                    arg1: x}}}}
```

Figure 6: Result of unifying F_R (figure 3) and F_S (figure 5)