We start with what Shieber\textsuperscript{1} calls \textit{F-structures}. These are DAGs with labeled edges — almost.

Shieber’s notation for F-structures is rather unwieldy. Let’s provide an alternative notation as a grammar and explain what it means below:\textsuperscript{2}

\begin{verbatim}
F-structure ::= '{' | '{' nameVal (, nameVal)* '}'

nameVal ::= name ':' (= int)? (\epsilon | atom | F-structure)

atom ::= symbol | integer | string
\end{verbatim}

\(\epsilon\) means “no F-structure here”; see below.

The F-structure

\{first: "foo", rest: {first: "bar", rest: Nil}\}

describes a tree with two arcs labeled \textit{first} and \textit{rest}. The object at the end of the \textit{head} arc is the \textbf{atom} "foo". (Exactly what data types we allow as atoms is application-dependent, but we should at least provide for integers and \textit{symbols}, a special sort of uniquified strings that can be included in programs by writing \textit{`sym.`} The object at the end of the \textit{rest} arc is another F-structure. The whole thing is a rather verbose description of a list of length two. (Nonetheless, this representation is often used in unification grammars, even when tuples would seem more appropriate.)

The mechanism for turning trees into DAGs is the use of \textit{indices}. In Shieber’s version, these are the numbers in boxes. In the BNF above, they’re the optional “= int” bits.\textsuperscript{3} The idea is that we mark with “=k” a piece of the (erstwhile) tree that we want to occur in more than one place, and elsewhere a terminal “=k” means “make that subgraph occur here, too.” Two ground rules must be observed:

\begin{itemize}
  \item \textsuperscript{1}Stuart Shieber 1986 An introduction to unification-based approaches to gram-
        mar. Stanford, California: CSLI Publications. Available online at URL
        \url{http://nrs.harvard.edu/urn-3:HUL.InstRepos:11576719}.
  \item \textsuperscript{2}The notation is pretty standard BNF. Literal characters are enclosed in single quotes.
        Parentheses are part of the meta-syntax. “\(a?\)” means an optional occurrence of \(a\). “\(a*\)” means zero or more occurrences of \(a\).
  \item \textsuperscript{3}Through the wonders of \LaTeX, it would be no trouble to draw boxes around the
        \textbf{int}s, but other arrangements are going to have to be made for practical computer-
        readable notations; might as well start now.
\end{itemize}
1. Exactly one occurrence of "=k" for a given \( k \) can label a nonempty F-structure; all the others must have content \( \epsilon \).\(^4\)

2. The graph you get by substituting pieces of tree as prescribed by the indices must not contain any cycles. Consider, for example:

\[
\{a:=1\{b:=2\}, \{c:=2\{d:=1\}\}\}
\]

Both piece \#1 and piece \#2 occur as subgraphs of themselves. That’s not allowed. (After all, a DAG is a directed acyclic graph.)

I said “exactly one” subgraph labeled with a given index “=i” can be non-\( \epsilon \). However, that one can label an empty F-structure (“\{\}”). Let \( F \) be the overall F-structure under consideration, such that somewhere inside it is an occurrence of \{\}. At some later time, we may “add” information at this point in \( F \).\(^5\) The change is “seen” at all the other occurrences of “=i”. Another way to think of it is that an index corresponds to a set of paths: If a point labeled \( =n \) is the \( l_k \) slot of the \( l_{k-1} \) slot of \ldots the \( l_1 \) slot of the F-structure, then \( \langle l_1, \ldots, l_k \rangle \) is the path to that occurrence of \( =n \):

\[
\{ \ldots, l_1: \{ \ldots, l_2: \{ \ldots, l_k: =n, \ldots \} \ldots \} \ldots \}
\]

I’ll use the notation \( F(l_1l_2 \ldots l_k) \) to indicate the subgraph reached by starting at the top of \( F \) and working down. More precisely, define \( \text{Dom}(F) \) to be the set of labels that occur at the top level of F-structure \( F \). For \( l \in \text{Dom}(F) \), define \( F(l) \) as the subgraph at the end of the arc labeled \( l \). Then \( F(l_1 \ldots l_k) = F(l_1)(l_2) \ldots (l_k) \), provided \( l_{i+1} \in \text{Dom}(F(l_1 \ldots l_i)) \) (\( 0 \leq i \leq n-1 \)).\[^6\]

Every index corresponds to a set of paths, all the paths that lead to a point labeled with that index. If that point is defined as \{\}, that represents

\[^4\]The \( \epsilon \) structure is not the same as an empty F-structure, denoted “\{\}”. The latter is what Shieber calls a variable, an F-structure with no content that will match any other F-structure; see below. By contrast, \( \epsilon \) just means, “This subgraph is defined somewhere else”; the subgraph may or may not be empty once you find it.

\[^5\]An F-structure is a DAG, a mathematical object. Strictly speaking, you can no more add information to \( F \) than you can add information to the number 1016. What really happens, as I’ll discuss shortly, is that in the course of carrying out an algorithm, we shift from a less constrained F-structure to a more constrained one.

\[^6\]This is not quite Shieber’s notation, but close enough you may not notice the difference.
a constraint that in any future instantiation of \( F \) in which one of these paths leads to a non-empty subgraph, all of them must lead to the same subgraph.

(Another occurrence of \{\} might be labeled with a different index; different occurrences of \{\} are distinct objects, or distinct tokens of the type \{\}.)

Graphs get instantiated through unification, which will require some explanation. Grammar rules look like this example (following Shieber, sort of, but starting at a different point):

\[
\begin{align*}
S &\rightarrow NP \ VP \\
S(\text{head}) &= VP(\text{head}) \\
S(\text{head subject}) &= NP(\text{head}) \\
VP(\text{head agreement}) &= NP(\text{head agreement})
\end{align*}
\]

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 — according to many linguists), but the idea is that 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.

Okay, but suppose they don’t? Then we must find a way to constrain them so they do. Suppose \( F_{\text{NP}} \) and \( F_{\text{VP}} \) are F-structures describing a certain noun phrase and verb phrase. Each constraint tells us how to create or shape the F-structure for the left-hand side, which I’ll call \( S \). The symbol \( S \) is a shorthand meaning that the “cat” slot of \( S \) must be \( S \). So we start with an F-structure with just one slot:

\[
\{\text{cat}: S\}
\]

The head of \( S \) must the same as the substructure labeled “head” in \( V \) (the VP). Suppose \( V \) looks like this:

\[
\{\text{cat}: VP, \text{head}: \{\text{cat}: V, \ldots\}, \ldots\}
\]

The constraint changes both \( S \) and \( V \):\footnote{Meaning, as mentioned above (p. ??), that if \( S \) and \( V \) are variables in our program, the values of these variables changes. The graph structures are immutable.}

\[
\begin{align*}
\{\text{cat}: S, \text{head}:1\} \\
\{\text{cat}: VP, \text{head}:1(\text{cat}: V, \ldots), \ldots\}
\end{align*}
\]

Are we allowed to add index numbers to an existing structure this way? Actually, we didn’t add anything. Remember that the index numbers’ sole function is to provide a way to describe DAGs in a convenient non-graphical...
notation. We just tacked “=1” on to indicate that the DAG $V(\text{head})$ should be the same as $S(\text{head})$. If you drew a picture, the same DAG would occur in both places.

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

$$\{\text{cat: NP}, \text{head:}\{\text{agreement:}\{\text{number: singular, person: third}\}}\}$$

and the VP

$$\{\text{cat: VP}, \text{head:}\{\text{form: finite}, \text{subject:}\{\text{agreement:}\{\text{number: singular, person: third}\}}\}\}$$

then we wind up with

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

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, $S(\text{head}) = VP(\text{head})$ and $NP(\text{head}) = S(\text{head subject}) = VP(\text{head subject})$. 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 $F_2$.

  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, $F_1 p \sqsubseteq F_2 p$; 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 $F_1$.

---

8Shieber 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 “⊆” instead of “⊂.”

9This recursive definition bottoms out because the substructures in question are smaller; eventually we get to atomic values.
• 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 $F_u \sqsubseteq F'_u$.\textsuperscript{10}

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).\textsuperscript{11}

If we return to our grammar, a rule

$$C_0 \rightarrow C_1 \ldots C_n$$

$$(\Gamma_1 p_1 = \Gamma_2 p_2)^*$$

means that if F-structure $X_i$ describes a list of words $w_i$, with $X_i \langle \text{cat} \rangle = C_i$, for $1 \leq i \leq n$, then $X_0$ describes $w_1 :: w_2 :: \ldots :: w_n$ (using the Scala “append” operation “:::”), provided that the constraints, each of the form $\Gamma_1 p_1 = \Gamma_2 p_2$, are satisfied, where $\Gamma_1, \Gamma_2$ are two (not necessarily distinct) elements of $\{X_0, \ldots, X_n\}$ 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; $X_i \langle \text{cat} \rangle = C_i$.\textsuperscript{12}

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.\textsuperscript{13}

\textsuperscript{10}In fewer words, $\sqcup$ defines a semilattice, and $F_1 \sqsubseteq F_2$ iff $F_1 \sqcup F_2 = F_2$.

\textsuperscript{11}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. Instead of binding variables, we’re adding edges to graph nodes. 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 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.

\textsuperscript{12}If, say, two noun phrases (NPs) occur on the right-hand side of a rule, we would call them NP\textsubscript{1} and NP\textsubscript{2}. But this doesn’t affect the $X_i$ numbering scheme, which just orders from left to right.

\textsuperscript{13}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 preprocessing 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).
A point in a word string is, as usual, denoted by the number of word 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:\(^1\)

\[
C_i\langle\text{span end}\rangle = C_{i+1}\langle\text{span begin}\rangle \quad (1 \leq i \leq n - 1)
\]

\[
C_0\langle\text{span begin}\rangle = C_1\langle\text{span begin}\rangle
\]

\[
C_0\langle\text{span end}\rangle = C_n\langle\text{span end}\rangle
\]

At this point we really must explain where the words are coming from. There’s a module delivering a stream\(^2\) of F-structures describing each word. A sentence beginning “She kept…” would produce something like this:

\[
\{
\text{cat: pronoun, lemma: she, span: \{begin: 0, end: 1\},}
\text{agreement: \{number: singular, person: 3rd, gender: fem\},}...
\}
\]

\[
\{
\text{cat: verb, lemma: keep, tns: past, span: \{begin: 1, end: 2\},}...
\}
\]

The lemma of a word is its “root,” the “unmarked” form from which other forms 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 \text{cat: S} that spans the entire sentence. Each F-structure above the word level is licensed by some grammar rule, in that the \text{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:

\[
C_0 \rightarrow C_1 \cdots C_n
\]

\[
X_{i1} p_{11} = X_{j1} p_{12}
\]

\[
X_{i2} p_{21} = X_{j2} p_{22}
\]

\[
\ldots
\]

\[
X_{im} p_{m1} = X_{jm} p_{m2}
\]

\(^1\)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.

\(^2\)But not necessarily a Stream.
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, $X_0$ through $X_n$, are interrupted by F-structures for $X_i$ and $X_j$. These are not additional components; $i_1$ might be 2 and $j_1$ might be 3, so that the piece labeled $X_i$ is the same as the piece labeled $X_2$ that appears just above it, and the piece labeled $X_j$ 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.\footnote{The use of numbers rather than identifiers to indicate coindexation gets tiresome at this point.} One of these paths may end in an already known structure; if not, then one is picked at random and made a variable.\footnote{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.} Only an example will make this clear, but first let’s explain the rest of our issues with figure 1.

These have to do with the first-rest notation for lists. While it’s standard, it is too verbose and clumsy for a data structure that is just as useful in F-structures as in programming. To streamline it, we’ll borrow a notation...
for lists from the Prolog programming language. If an F-structure $F$ has just two features, $\text{first}: H_0$ and $\text{rest}: T$, then the list notation for $F$, $F^L = [H_0 \mid T^L]$. If $T^L = [H_1, H_2, \ldots, H_k \mid T']$, then $F^L$ may be written $[H_0, H_1, H_2, \ldots, H_k \mid T']$. If $T' = \text{nil}$, $F^L$ may be written as simply $[H_0, H_1, H_2, \ldots, H_k]$. 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):

- $X_0(\text{head}) = X_2(\text{head})$
- $X_0(\text{head subject}) = X_1(\text{head})$
- $X_1(\text{head agreement}) = X_2(\text{head agreement})$

to which we’ll add two more:

- $X_0(\text{sem fcn}) = X_1(\text{sem})$
- $X_0(\text{sem arg1}) = X_2(\text{sem})$

just to remind us that we’re ultimately interested in the semantics (internal representation) or our sentence. The two extra constraints are the Montagovian $\text{sem}(X_0) = \text{sem}(X_1)(\text{sem}(X_2))$, 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
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: \{constituents: [:=1, :=2]\},
X1:=1 \{cat: C_1,
    \text{span: \{begin: }i_0, \text{end: }i_1,\}
    \text{—everything else we know about the tree } X_1 \—\},
X2:=1 \{cat: C_2,
    \text{span: \{begin: }i_1, \text{end: }i_2,\}
    \text{—everything else we know about the tree } X_2 \—\},
\ldots
X_n:=n \{cat: C_2,
    \text{span: \{begin: }i_{n-1}, \text{end: }i_n,\}
    \text{—everything else we know about the tree } X_n \—\}\}

Figure 4: Schema for \textit{F}_S, the “situational” F-structure

extent. It can be interpreted left-to-right, as if one is trying to generate the entire language.\textsuperscript{18} 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 \textit{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 \textit{n} contiguous phrases that have been found already must be packaged up into a structure \textit{F}_S (“S” for “situation”) that resembles \textit{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. \textit{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 \textit{F}_S if it’s already in \textit{F}_R.

Figure 5 is the schema of figure 4 instantiated for a particular analyzed word string, “Napoleon retreated.”\textsuperscript{19}

Now we can apply unification to our parsing problem. We unify \textit{F}_R and

\textsuperscript{18}Which is why the tradition started by Chomsky sixty years ago is known as “generative
Figure 5: $F_S$ for a particular situation: “Napoleon” and “retreated”
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.

[[Unification algorithm — conspicuously absent.]]
Figure 6: Result of unifying $F_R$ (figure 3) and $F_S$ (figure 5)