26 Feature Structures

Drew McDermott drew.mcdermott@yale.edu

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

1 F-structure and their Notation

We start with what Shieber[1] calls *F-structures* — *feature structures*. These are DAGs with labeled edges.

Shieber's notation for F-structures does not seem plausible for input and output to a computer. Let's provide an alternative notation as a grammar and explain what it means below:[2]

F-structure ::= *Atom* | *Fvar* | *FScomplex*

FScomplex ::= '{' *NameVal* (',' *NameVal*)* '}'

NameVal ::= *Fname* ':' *Fval*

Fname ::= *Symbol*
Fval ::= *F-structure* | '='*UnsInt* | '='*UnsInt*'=' *F-structure*

Fvar ::= '{?}'
Atom ::= *Fnil* | *Quot Symbol* | *integer* | *string* | ...

Fnil ::= '<>'
Symbol ::= *Alpha AlphaNumeric**
AlphaNumeric ::= *Alpha* | *Digit*
Alpha ::= A | B | ... | Z | a | ... | z
Digit ::= 0 | 1 | ... | 9
UnsInt ::= *unsigned integer*
Quot ::= *right single quote*

The F-structure

[1] Stuart Shieber 1986 An introduction to unification-based approaches to grammar. Stanford, California: CSLI Publications. Available online at URL http://nrs.harvard.edu/urn-3:HUL.InstRepos:11576719.

[2] The notation is pretty standard BNF. Literal characters understood as part of the language being defined are enclosed in single quotes. Parentheses (without quotes) are part of the meta-syntax. "α?" means an optional occurrence of α. "α*" means zero or more occurrences of α; "α+" means one or more.

```
{first: "foo", rest: {first: "bar", rest: <>}}
```

describes a tree with two arcs labeled `first` and `rest`. The object at the end of the `first` arc is the *Atom* `"foo"`. (Exactly what data types we allow as atoms is application-dependent, but we should at least provide for integers, strings, `<>` (the empty list), and *Symbols*, a special sort of uniquified string that can be included in F-structures (and Scala programs) by writing *'Symbol*.

The object at the end of the `rest` arc is another F-structure. The whole thing is a rather verbose description of a list of length two. Verbose it may be, but it's the normal way lists are represented as F-structures. We want to provide a more concise syntax to use in printing and reading lists. We'll use angle brackets, treating

$$\texttt{<}f_0\texttt{,}\ f_1\texttt{,}\ \ldots\texttt{,}\ f_{n-1}\texttt{>}$$

as an abbreviation of

$$\texttt{\{first:}\ f_0\texttt{,}\ \texttt{rest:}\ \texttt{\{first:}\ f_1\texttt{,}\ldots\texttt{,}\ \texttt{\{first:}\ f_{n-1}\texttt{,}\ \texttt{rest:}\ \texttt{<>\}\}\}}$$

Besides the use of angle brackets, the only other issue is what to do when the value of the `rest` feature is neither `<>` nor an F-structure with a `first` and a `rest`. (The value of the `rest` feature might be a variable, for instance.) To deal with this case, we'll borrow notation from the Prolog programming language. Suppose the first k elements (x_0 to x_{k-1} are specified, but the kth tail is a variable. We'll write this as

$$\texttt{<}x_0\texttt{,}\ x_1\texttt{,}\ \ldots\texttt{,}\ x_{k-1}\ \texttt{|}\ t_k\texttt{>}$$

using a vertical bar[3] to separate x_{k-1} from the `rest`. We were imagining that t_k was `{?}`, but it can be any legal *Fval*.

The mechanism for turning trees into DAGs is the use of *indices*. In Shieber's version, these are the numbers in boxes. In the BNF above, they're the optional "'`=`'`UnsInt`" bits.[4] 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:

[3] I hope it's clear that this is distinct from the vertical bar of our BNF notation (|).

[4] Through the wonders of LaTeX, it would be no trouble to draw boxes around the $\boxed{\texttt{int}}$s, but other arrangements are going to have to be made for practical computer-readable notations; might as well start now.

Figure 1: Cyclic (Illegal) DAG

1. There should be exactly one occurrence of "=k=" (an *index definition*) for a given k that appears as an *pointer index* (of the form "=k").

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: {e: =1}}}
   ```

   Both piece #1 and piece #2 occur as subgraphs of themselves. (A graphical representation is shown in figure 1.) That's not allowed. (After all, a DAG is a directed *acylic* graph.)

2 F-DAGs

I said "exactly one" subgraph can be labeled as an index definition "=i=" for an index i that occurs as a pointer, and phrases like that raise a question we (and Shieber) have glossed over: *Over what context* are we keeping track of how many times an index is used or defined? F-structures are sometimes thought of as DAGs, and sometimes as nodes in a DAG. Put another way, each F-structure can be viewed as the root of a DAG, but we can't *always* look at it that way. Once we've chosen the root node, all the other F-structures reachable from it must be viewed as non-root nodes of the resulting DAG.[5] Until and unless the DAG is defined, deciding whether every index pointer occurs as a pointer definition somewhere is meaningless. I will use the term *F-DAG* for a DAG rooted in an F-structure when it's important to distinguish one from DAGs in general. For most of the lecture notes the distinction won't be important, and "the DAG" will be synonymous with "the F-DAG."

With these distinctions sorted out, let's turn to exactly what sort of data structure we're talking about. Pointer definitions have varied properties

[5]Even though in other contexts we might take that very same F-structure as the root of its own DAG

depending on how the index is defined, i.e., what sort of F-structure follows the second equal-sign in "=k=F":

1. F is a *FScomplex*: This is really the simplest case. All occurrences of "=k" elsewhere in the DAG are to be treated as occurrences of this node.

2. F is "{?}": All occurrences of "=k" are treated as occurrences of a single variable, and all other occurrences of "{?}" are *distinct* variables. When F-structures are unified, if this variable is matched against a non-variable and the entire unification succeeds, then in the resulting (new) F-DAG, all occurrences of this variable will be replaced by the same F-structure.

3. F is an *Atom* A: This is a tricky case. Do we view other occurrences of "=k" as referring to the same occurrence of A, or as occurrences of atoms equal to A? Is this even a meaningful distinction? The answer to the second question, somewhat surprisingly, is "Yes." If every F-structure is a node in a graph, then *Atom*s are nodes in a graph, too, and two different nodes can hold equal atoms. So, if the first question is meaningful, what is the answer? It may seem arbitrary, but it affects how unification proceeds. It will turn out to be a pointless expense to force node equality during unification, so we don't enforce it at all.

   The bottom line is that for *Atom*s, an occurrence of "=k" must be an *Atom* equal to A.[6]

Another way to approach all this is to this of an index as defining a set of *paths*: If a point labeled =n is the l_k slot of the l_{k-1} slot of ... the l_1 slot of the root F-structure of an F-DAG, then $\langle l_1, \ldots, l_k \rangle$ is the *path* to that occurrence of =n:

$$\begin{array}{l} \{\ldots,\ l_1\colon\{\ \ldots, \\ \qquad\qquad l_2\colon\{\ldots,\ \ldots\{\ldots,\ l_k\colon{=}n\ldots,\ \ldots\}\ldots\} \\ \qquad\qquad \ldots\}\} \end{array}$$

I'll use the notation $R\langle l_1 l_2 \ldots l_k \rangle$ to indicate the subgraph reached by starting at the root R of the DAG and working down. More precisely, define $\mathrm{Dom}(F)$

[6] Another thing to point out is how odd it is to use empty brace pairs to represent variables, as Shieber seems to do. Precisely what set of feature-value pairs does an atom like `plural` correspond to? Although one can argue that one marker is as good as another, and {?} has no obvious merit compared to {} if the latter is treated as a meaningless marker, in fact it certainly *seems* to mean the empty F-structure, the same way <> seems to — and does — represent the empty list.

to be the set of labels that occur at the top level of F-structure F. For $l \in \mathrm{Dom}(F)$, define $F(l)$ as the subgraph at the end of the arc labeled l. Then $R\langle l_1 \ldots l_k \rangle = R(l_1)(l_2)\ldots(l_k)$, provided $l_{i+1} \in \mathrm{Dom}(F\langle l_1 \ldots l_i \rangle)$ $(0 \leq i \leq n-1)$.[[[7]]

Given the root F-structure R, every index i reachable from R 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 a constraint that in any future instantiation of R in which one of these paths leads to a non-empty subgraph, all of them must lead to the same subgraph.

The reason for all this machinery will become clearer in the next couple of notes.

[7] This is not quite Shieber's notation, but close enough you may not notice the difference.