Goals

1. Explore the mechanics of meaning representation and composition in the LKB;
2. make a relatively small number of changes to turn our chart parser into a generator.

Obtain the Starting Package and Bring up the LKB

(a) As always, bring up bash, obtain a starting package by typing `cvs checkout week7` and start the LKB.
(b) Open the files ‘`generate.lsp`’ and ‘`mrs.lsp`’, where we will be making changes throughout this exercise.

1 Minimal Recursion Semantics (5 + 5 + 10 Points)

We have added semantics to our grammar from the previous exercise, so that when we parse a sentence, the resulting feature structure also contains a meaning representation. Since our approach to generation requires as input the semantics for the sentence to be generated, we start by gaining some familiarity with the way meanings of words and phrases are represented in this grammar.

(a) Using the LKB, parse the sentence `the dog gave the cat to the aardvark` (by selecting the LKB menu item `Parse – Parse input`). To view the semantics, left-click inside the little tree window that popped up, and select the menu item `MRS` (for Minimal Recursion Semantics, the representation language we’re using for semantics). A window will appear showing the semantics for that sentence, consisting of a list of relations (elementary predications, aka eps) in the `RELS` attribute, as well as the `INDEX` value and another attribute `LTOP` which we will rather cheerfully ignore for this exercise. Now parse the sentence `the dog gave the aardvark the cat` and view its semantics as well. In a few sentences, discuss two aspects of this pair of semantic representations:

(i) Why are they equivalent even though they are not identical?
(ii) How does the grammar enable this equivalence?

Recall that the only two semantic attributes we employ in signs are `RELS` (a list of relations) and `INDEX` (the ‘hook’ used to combine the semantics of that sign with another).

(b) Still using the LKB, parse the sentence `the dog barks happily` and view its semantics. Do the same for `happily the dog barks`, and discuss in a few sentences how the grammar again ensures that equivalent meaning representations are computed for these two sentences.

(c) With semantics enabled, we can now use the grammar to generate in the LKB, by first parsing a sentence, and then using the resulting semantics as the input to the generator (in the real world, though, the input semantics would be supplied from the transfer component of a machine translation system or the ‘what to say’ module of a natural language dialogue system). To call the generator for a sentence you have parsed, click on the little parse tree window again, but instead of choosing the `MRS` menu item, choose `Generate` instead. The generator will efficiently produce all of the sentences which the grammar says are semantically equivalent to the input. Experiment with the generator for ten examples and discuss in a few sentences what you observe about its behavior. Remember that you can inspect the file `lexicon.tdl` to see your vocabulary choices for the examples. Make sure you use all of the lexical entries at least once in your examples, and make them as varied as you can.

2 Generation (15 + 10 + 5 Points)

The role of the chart in generation is slightly different from parsing. Edges will be organized by semantic indices, corresponding to variables from the input semantics. The chart will contain as many cells as there are distinct variables in the input semantics, plus one designated cell for edges that have an underspecified index (either because they have no index of their own—as is the case with semantically empty lexical entries...
like case marking prepositions—or because they correspond to partial phrases for which the argument position supplying the index value(s) has yet to be instantiated). The cell providing ‘shelter’ for edges with underspecified indices (‘drifters’, in a sense), we will call the pen of the chart. Conceptually, the pen will be considered ‘adjacent’ to all other chart cells; i.e. edges in the pen are treated as if they were elements in all of the regular chart cells.

The process of relating the indices occurring in signs of the grammar (words and phrase) to the variables supplied by the input semantics is called skolemization (after the Norwegian logician Thoralf Skolem, 1887–1963): for each relation of the input semantics, all lexical entries are retrieved that introduce the relation (the inverse of lexical look-up by orthography). To equate the indices occurring in the lexical entry (feature structures of types index, object, or event) with the fully-specified variables in the input semantics, each index is unified with what is called a skolem constant, essentially the variable name (like \( e_1 \) or \( x_2 \)) from the input logical form. For this purpose, the index type bears a feature SKOLEM, which we will use to house the skolem constant.

For expository purposes, assume the following input semantics:

\[
\begin{align*}
\{ & e_1 \\
& \text{the}_\text{q-rel}[\text{ARG0 } x_2] \\
& \text{dog}_\text{n-rel}[\text{ARG0 } x_2] \\
& \text{chase}_\text{v-rel}[\text{ARG0 } e_1, \text{ARG1 } x_2, \text{ARG2 } x_3] \\
& \text{the}_\text{q-rel}[\text{ARG0 } x_3] \\
& \text{cat}_\text{n-rel}[\text{ARG0 } x_3] \\
\} \\
\end{align*}
\]

which is a simplified version of Minimal Recursion Semantics (MRS). To us, an MRS is a bag (a multi-set, i.e. a set in which elements can occur more than once but stand in no meaningful order to each other) of elementary predications (EPs). Each EP has a predicate (like ‘dog\_n\_rel’ for what we take to be the meaning of the noun dog) and a set of roles, feature – value pairs with the role names drawn from an impoverished ARG\_n language and the values being logical variables. Similar to the LKB view you saw above, the INDEX value is printed at the top (the \( e_1 \) on the first line) followed by the RELS elements, one per line; in our condensed notation, however, the attributes INDEX and RELS are omitted. For convenience, variable names indicate types, where \( e_i \) denotes an event and \( x_i \) an object.

The feature structure of the lexical entry for dog in turn includes the following elements (not showing syntactic information):

\[
\begin{align*}
\text{SEM} & \begin{bmatrix}
\text{INDEX} & 1_{\text{object}} & \text{SKOLEM} *\text{string}\*
\end{bmatrix} \\
\text{RELS} & \begin{bmatrix}
\text{PRED} & \text{“dog\_n\_rel”} \\
\text{ARG0} & 1
\end{bmatrix}
\end{align*}
\]

Starting from an elementary predication from the input semantics that triggers lexical look-up of this lexical entry, skolemization will effectively specialize the value of the ARG0 role, such that the SKOLEM value becomes the string \( ”x2” \). As we create a (passive) edge for the skolemized lexical entry, we will say that its index is \( x_2 \).

Given this new notion of (semantic) indices for edges, we will organize passive edges according to the index that they ‘provide’ (i.e. the value of their SEM.INDEX.SKOLEM feature) and active edges according to the index that they seek to combine with next. Thus, in determining the index of an active edge, we will be looking at the value of SEM.INDEX.SKOLEM on the argument position referred to by the first element of the unanalyzed component on the edge (the one immediately ‘following the dot’); this will make sure that active edges are stored in the same chart cell as candidate passive edges for them to consume.

Semantic indices for chart access will either be strings (corresponding to variables from the input MRS) or nil, which we will use to represent an underspecified index. We have modified the chart abstract data type to support both indexing of edges by start and end positions (in parsing) or by semantic indices (in generation). The function chart-adjoin() will inspect its edge argument and store it appropriately; likewise, chart-cell() can now be called with either two numerical arguments or just one, in which case it will consider its first argument a semantic index. Using the special index value nil will add to the chart pen when storing edges into the chart, and return all chart edges when used for retrieval. Additionally,
the chart provides two new accessor functions, \texttt{passive-edges-at()} and \texttt{active-edges-at()}, which take a semantic index (or \texttt{nil}) as their sole argument and return all edges of the specified type from the cell denoted by that index (or from the pen).

Besides these changes to the chart structure, we have augmented the edge data type to (i) allow edges to keep track of their index, (ii) record in a component \texttt{string}, on each edge, what the surface form corresponding to this edge is, and (iii) record in a component \texttt{eps} which of the relations from the input MRS are being realized by this edge. When building new edges, we need to make sure to supply correct values for all three new edge components, and will use the latter to avoid combining two edges that ‘overlap’ in their semantics (which would amount to verbalizing a relation from the input MRS more than once).

(a) Finally, we are in a position to look at the definition of the main \texttt{generate()} function in ‘generate.lsp’. Coming to the end of this quarter, we have done most of the work for you. Go through each step in \texttt{generate()}, read our comments, and remind yourself of the corresponding pieces in the \texttt{parse()} function (which you can view in the file ‘active.lsp’). Fill in the missing parts.

(b) The core function combining edges, \texttt{fundamental-rule()}, will be shared between the parser and generator. The parser will centrally depend on \texttt{from} and \texttt{to} values, where the generator will use \texttt{index} exclusively. When combining an active and a passive edge, \texttt{fundamental-rule()} maintains all edge components in a way that is compatible with both parsing and generation. Given what we said about determining the index of a new edge, work out how to call the new function \texttt{dag-index()} to compute the appropriate value. \texttt{dag-index()} takes a \texttt{dag} and, optionally, a \texttt{path} as arguments and extracts the \texttt{SEM.INDEX.SKOLEM} value from \texttt{dag} or the sub-structure of \texttt{dag} pointed to by \texttt{path}. As for combining the additional \texttt{string} and \texttt{eps} components for the new edge, make sure to accumulate the respective values from both the active and passive edges (in that order) on the new edge.

(c) The new predicate \texttt{independentp()} at the bottom of ‘generate.lsp’ will make sure that two edges are semantically ‘independent’ of each other, i.e. do not overlap in the sets of elementary predications that they verbalize.

Once you have eliminated all occurrences of ‘???’ in ‘generate.lsp’, save everything and reload (eliminating load-time errors if necessary). To test our generator, we will use the same strategy as the LKB, viz. read off an MRS structure from a parsing result: First be sure the refined \texttt{fundamental-rule()} still works for parsing; confirm that invoking \texttt{parse()} on, say, \texttt{the dog barks} has the expected result(s). Next, use our new function \texttt{dag-to-mrs()} to convert the feature structure of a result edge returned by \texttt{parse()} into an MRS structure, be sure that what you see corresponds to your expectations, and invoke \texttt{generate()} on the MRS.

\begin{verbatim}
? (setf mrs (dag-to-mrs (edge-category (first (parse '(the dog barks)))))
? (generate mrs)
\end{verbatim}

From looking at the edges, you should be able to confirm that the expected string(s) were generated (and nothing else). Use the \texttt{browse()} function to inspect edges returned from the generator and also try the following:

\begin{verbatim}
? (browse (unpack-edge (first (generate mrs)))
\end{verbatim}

Experiment with some of the examples from the first part of the exercise and confirm that our generator returns results equivalent to the one built into the LKB.

\section*{Submitting Your Results}

Since our programs are of some interesting complexity at this point, please make sure to submit the contents of the \texttt{entire ‘week7’ directory}.

Submit your results in email to Dan and Stephan by noon on Thursday, May 29.