Homework #8: CMPT-413

Distributed on Mon, Apr 4, Due on Mon, Apr 11

Anoop Sarkar – anoop@cs.sfu.ca

(1) Implement the feature unification algorithm given in Figure 11.8 in Jurafsky and Martin (on page 423). Download the file `FeatureStruct-inc.pm` which contains almost all the functions you need – copy this file to `FeatureStruct.pm` and complete the definition of the function `unify` in this file. The feature structure $\left[\textbf{number: sg}\right]$ has a standard textual notation `<number>=sg` which uses the notation of *feature paths* (see page 398 in J&M). Reentrant features can also be represented. The feature structure:

$$\begin{bmatrix} \textbf{agreement:} & \boxed{1}\begin{bmatrix}\textbf{number: sg}\\ \textbf{person: 3}\end{bmatrix} \\ \textbf{subject:} & \left[\textbf{agreement: } \boxed{1}\right] \end{bmatrix}$$

is represented as:

```
<agreement number>=sg
<agreement person>=3
<subject agreement>=<agreement>
```

Note that the second equation in the textual notation sets up the index used in the graphical representation used to share the value of *agreement*.
In order to implement unification using the algorithm given in J&M, the feature structures have to be represented as directed acyclic graphs (*dag*s).The file `FeatureStruct.pm` already has a function to convert the textual notation shown above to dags. The function `parseText` converts from text to a dag representation in Perl, while `stringValue` converts the dag representation back to text. For example, the following code converts from text to a dag and then back again.

```
use FeatureStruct;
my $dagP = FeatureStruct::parseText("<number>=sg");
my $dagQ = FeatureStruct::parseText("<person>=3");
print FeatureStruct::stringValue($dagP);
print FeatureStruct::stringValue($dagQ);
```

`parseText` converts the feature structure text into a Perl data structure for dags. The data structure is a nested hashtable: each entry in the hashtable is either a reference to another hashtable or a string value The references are used to create the graph structure. For example, the input feature structure:

```
<number>=sg
<person>=3
```

is converted to the Perl dag (NULL values are not stored in the hashtable):

```
$dag->{'content'}->{'number'}->{'content'}='sg'
$dag->{'content'}->{'person'}->{'content'}='3'
```

Note that the additional edges called `content` and `pointer` that are required for the unification algorithm given in J&M is already part of the dag constructed by `parseText`. The function `realContents` takes a dag and provides the "real contents" as defined in the description of the unification algorithm in J&M.

a. Provide the definition of the function `unify` in `FeatureStruct.pm`. Your code for `unify` should return −1 if unification failed, else it should return the result of unification as a dag. You do not have to deal with disjunctive features, e.g. `<number>=sg|pl`. This dag should be printed out as text using `stringValue`, as shown below:

```
my $dagR = FeatureStruct::unify($dagP, $dagQ);
if ($dagR == -1) { print "failed\n"; }
else { print FeatureStruct::stringValue($dagR); }
```

b. Using `unify` provide the results as output from `stringValue` for the following unifications:

1. $\left[\textbf{number: sg}\right] \sqcup \left[\textbf{number: sg}\right]$
2. $\left[\textbf{number: sg}\right] \sqcup \left[\textbf{person: 3}\right]$
3. $\left[\begin{array}{ll}\textbf{agreement:} & \boxed{1}\left[\textbf{number: sg}\right] \\ \textbf{subject:} & \left[\textbf{agreement: } \boxed{1}\right]\end{array}\right] \sqcup \left[\textbf{subject: } \left[\textbf{agreement: } \left[\textbf{person: 3}\right]\right]\right]$

Note that `stringValue` prints out reentrant feature structures in a verbose manner. For example,

```
my $fstext=<<EOT;
<agreement number>=sg
<subject agreement>=<agreement>
EOT
print FeatureStruct::stringValue(FeatureStruct::parseText($fstext));
```

produces the following output where the reentrant feature is simply printed out until the value `sg`:

```
<subject agreement number>=sg
<agreement number>=sg
```

(2) The `WordNet` database is accessible online at `http://www.cogsci.princeton.edu/~wn/`. Follow the link to *Use WordNet Online* or go directly to:
`http://www.cogsci.princeton.edu/cgi-bin/webwn`

`WordNet` contains information about word *senses*, for example, the different senses of the word *plant* as a manufacturing plant or a flowering plant. For each sense, `WordNet` also has several class hierarchies based on various relations. Once such relation is that of *hypernyms* also known as *this is a kind of* . . . relation. It is analogous to a object-oriented class hierarchy over the meanings of English nouns and verbs and is useful in providing varying types of word class information.

For example, the word *pentagon* has 3 senses. The sense of *pentagon* as five-sided polygon has the following hypernyms. The word *line* has 29 senses. The sense of *line* as trace of moving point in geometry has the following hypernyms.

Sense3: pentagon
⇒polygon,polygonal-shape
⇒plane-figure,two-dimensional-figure
⇒figure
⇒shape,form
⇒attribute
⇒abstraction

Sense4: line
⇒shape,form
⇒attribute
⇒abstraction

The *lowest common ancestor* for these two senses is the hypernym shape, form. A *hypernym path* goes up the hypernym hierarchy from the first word to a common ancestor and then down to the second word. Note that a hypernym path from a node other than the lowest common ancestor will always be equal to or longer than the hypernym path provided by the lowest common anscestor. For the above example, the hypernym path is *pentagon* ⇒ polygon,polygonal-shape ⇒ plane-figure,two-dimensional-figure ⇒ figure ⇒ shape,form ⇒ *line*.

Find the lowest common ancestor across all senses of each of the following pairs of words: (a) *Canada* and *Vancouver*, and (b) *English* and *Tagalog*.