Word Parsing, Part 2

2014-05-19 BY GENE

In the previous episode, I rambled about the history and completion of my mechanical word parser. This time, we go section by section through the deceptively short synopsis…

As before, the code lives at https://github.com/ology/Lingua-Word-Parser and also at https://metacpan.org/release/Lingua-Word-Parser.

UPDATE: A Web GUI can now be found at https://github.com/ology/Word-Part

---

```
#!/usr/bin/env perl
use strict;
use warnings;
use Lingua::Word::Parser;
```

This is the standard preamble for perl followed by my word parser module.

Right off, we create an object with attributes for the lexicon file and the word to parse:

```
my $p = Lingua::Word::Parser->new(
  file => 'eg/lexicon.dat',
  word => 'abioticaly',
);
```

This lexicon is made of regular expressions (indicating prefix, affix, suffix or combining vowel) and the definition. The word can be anything that the dictionary knows about. (Yes it's a loaded deck, but that is the point – using a finite lexicon.) It looks like this:

```
a(?=\w) opposite
ab(?=\w) away
bi(?=\w) two
bio(?=\w) life
(?<=\w)ic belonging
(?<=\w)ly like
(?<=\w)o(?=\w) combining
(?<=\w)tic possessing
(?<=\w)y like
```

The first method of the code captures the known parts of the word:

```
my ($known) = $p->knowns;
```

Printing *Dumper($known)* shows more information than is (currently) used to process a word. Here are the first entries of a made up, test word "abiotically" :

```
{
  defn => "life",
  mask => "0111000000",
  part => "bio",
  span => [1,3]
},
{
  defn => "away",
  mask => "1100000000",
  part => "ab",
  span => [0,1]
}, ...
```

In particular, we use the set of masks for parsing the word into known parts. These are found by iterating over *each and every entry* of our simple word part lexicon. Oof!

The mask that is created shows a single known "chunk" with ones for the known characters and zeros for the unknowns.

Next up, we create all possible combinations of known parts that do not overlap. This is where the mechanical, combinatorial bits come into play.

```
my $combos = $p->power;
```

This produces a long list that begins with:

```
[
  "0000000011",
  "0000000100",
  "0000111000",
  "0001000000",
  "1100000000"
],
[
  "0000000001",
  "0000000100",
  "0000111000",
  "0001000000",
  "1100000000"
], ...
```

This is the set of non-overlapping bitstrings that when combined, make all or part of the original word.

The meat of the *power()* subroutine is the iteration over the powerset of masks, ignoring ones that contain overlapping known parts.

```
sub power {
    my $self = shift;

    # Get a new powerset generator.
    my $power = Data::PowerSet->new(sort keys %{ $self->{masks} });

    # Consider each member of the powerset.
    while (my $collection = $power->next) {

        # Compare each mask against the others.
        LOOP: for my $i (0 .. @$collection - 1) {

            # Set the comparison mask.
```

```
            my $compare = $collection->[$i];

            for my $j ($i + 1 .. @$collection - 1) {

                # Set the current mask.
                my $mask = $collection->[$j];
                ...
```

You can see the mechanical iteration. The crucial comparison of masks to determine overlap is done with an XOR and an OR expression. If they are not the same, there is an overlap!

Lastly, we score the combinations.

```
my $score = $p->score;
```

I spent a lot of time thinking about this, but in the end it was two ratios: “known chunks vs unknown chunks” and known characters vs unknown characters.

```
sub score {
    my $self = shift;

    # Visit each combination...
    my $i = 0;
    for my $c (@{ $self->{combos} }) {
        $i++;
        my $together = $self->or_together(@$c);
        ...
```

For each (non-overlapping) combination we OR them together, then the four measures are counted up…

In order to do that, we run-length encode an “un-digitized” string. For example, *011100* becomes *ukkkuu* which finally is converted to *u1k3u2*.

By tallying these numbers it is possible to get two ratios – a two dimensional familiarity score that effectively considers the relative sizes of contiguous spans of known vs unknowns.

When the word is reconstructed into something meaningful, that is the ratios and the word itself with delimited parts, we get this:

```
[
  5/8 + 10/40 => <a>bioticaly, a<bio>ticaly, abio<tic>aly, abiotic<a>ly, ab
  6/10 + 10/50 => <a>bioticaly, a<bi>oticaly, abi<o>ticaly, abio<tic>aly, a
]
```

It seems easy enough when you look at a word and the parts jump out at you like differently colored LEGOs. But convincing a computer to do it takes a bit of head scratching.

FILED UNDER: SOFTWARE
TAGGED WITH: COMBINATORICS, LINGUISTICS, MORPHOTACTICS, PARSING, PERL, SUBSEQUENCES, TERMINOLOGY

Epistemologist-at-large

^ Top