MetaMap Migration to Lexical Tools Java APIs – Derivational Issues

First of all, our approach is to make the interface code as close as "C-code" (even include known bugs) and we are able drop the difference from 419,502 (last week) to 732 (words). I think we should stop at this point and discuss with you on issues we found before we move on (instead of keep digging out bugs from C-code on dmv).

Issues (bugs) in "C-code" are described below. We will need to go through every single item together to make it right. Hopefully, we can do this right this time so we don't need to worry about it in the future ☺

I. Use the right Case, category, and inflection:
Derivation variants generation depends on three things:
a). the spelling of term (case sensitive)
b). category (part of speech, such as noun, verb, adj, adv, ..)
c). inflection (such as base)

So, the first step is to get the categories and inflections of the input term (lvg -f:Ln).

Issue 1. in C-code, it does not take inflections into consideration.
Example: zonated

> lvg –f:Ln (to get category and inflection from LEXICON)
zonated
zonated|zonated|1|1|Ln|1|
zonated|zonated|1|256|Ln|1|
zonated|zonated|1024|32|Ln|1|
zonated|zonated|1024|64|Ln|1|

Accordingly, the only derivation should be generated is from result 1: "zonated|1|1" because all the other 3 results have no derivational variant due to the inflections.

> lvg -f:d -kd:3 -cf:2 -if:3 (to get derivation by specifying term|cat|infl)
zonated|1|1
zonated|1|1|zonat|128|1|d|1|
zonated|1|1|zonate|128|1|d|1|
zonated|1|1|zonatedity|128|1|d|1|
zonated|1|1|zonatedness|128|1|d|1|
zonated|1|1|zonatedly|2|1|d|1|

However, in C-code, it ignores inflections (set inflection to all, include base) and generate follows:

zonated|1024|1 (<= where the inflection should be 32 or 64)
zonated|1024|1|zonateder|128|1|d|1|
zonated|1024|1|zonatedor|128|1|d|1|
zonated|1024|1|zonatedant|128|1|d|1|
zonated|1024|1|zonatedance|128|1|d|1|
zonated|1024|1|zonatedment|128|1|d|1|
zonated|1024|1|zonatedation|128|1|d|1|
zonated|1024|1|zonatedably|2|1|d|1|
zonated|1024|1|zonatedant|1|1|d|1|
zonated|1024|1|zonatedable|1|1|d|1|

Issue 2. Should derivational variants be case sensitive?
=> In C-code, there are algorithm error in handling this.

For example:

> lvg –f:Ln
zap
zap|ZAP|128|1|Ln|1|
zap|ZAP|128|512|Ln|1|
zap|zap|1024|1|Ln|1|
zap|zap|1024|1024|Ln|1|
zap|zap|1024|262144|Ln|1|

Results 1 and 2 should be removed because the case does not match.
So, the only derivational variants should be generated from result 3: "zap|1024|1"

> lvg -f:d -kd:3 -cf:2 -if:3

zap|1024|1
zap|1024|1|zaper|128|1|d|1|
zap|1024|1|zapor|128|1|d|1|
zap|1024|1|zapant|128|1|d|1|
zap|1024|1|zapance|128|1|d|1|
zap|1024|1|zapment|128|1|d|1|
zap|1024|1|zapation|128|1|d|1|
zap|1024|1|zapably|2|1|d|1|
zap|1024|1|zapant|1|1|d|1|
zap|1024|1|zapable|1|1|d|1|

However, in C-code, it takes line 1 "ZAP|128|1" and lowercase it into "zap|128|1" and generate:

zap|128|1
zap|128|1|zapism|128|1|d|1|
zap|128|1|zapist|128|1|d|1|
zap|128|1|zapy|1|1|d|1|
zap|128|1|zapal|1|1|d|1|
zap|128|1|zaped|1|1|d|1|
zap|128|1|zapic|1|1|d|1|
zap|128|1|zapary|1|1|d|1|
zap|128|1|zapous|1|1|d|1|
zap|128|1|zapable|1|1|d|1|

In addition, there are more case handling errors in C-code algorithm to handle case with all lower case, all upper case, mixed upper case and lower case, etc.. The worse thing is C-code is not consistent on the case handling.

II. Derivation variants generation based on two algorithms:

a) Fact: known facts by lexicon (uses database tables)
b) Rules: not known by lexicon (uses trie algorithm)

Issue 3. The known derivational facts in C-code is an older version..

- ⇨ We found out there are only about 582 facts in C-code while there are 4,559 facts in the 2010 Java version. This difference are the main part of the current remaining 732 different words (when Java code find some derivational variants C-code misses.

For example:
Eastwards|2|1 is a known fact to get the derivation of east|128 in Java code (and not in C-code)

Issue 4. The rules in Java-code have implemented two new enhanced features (heuristic rules) while C-code does not have these features. Also, these two rules are configurable in Java:

Issue: 4-a). min. term length:
In derivation generation, if the length of generated derivational variant is too small (default is 3 in Java; no such function in C, so the value is 0), the word is usually an acronym or does not have too much meaning. This rule is implemented to eliminate such terms.

For example:
C-code generate "zo" from "zoic", while Java code could filters this out.

C-code generate "r" from "red", while Java code could filter this out.
C-code generate "re" from "red", while Java code could filter this out.

C-code generate "r" from "rist", while Java code could filter this out.
…

Issue: 4-b). min. stem length:
The stem length is the length of the word minus the length of input suffix rule. This is used in trie algorithm to rule out some unreasonable cases. For example, RULE|ic$|adj|base|y$|noun|base, the length of input suffix (ic$) is 2. If the input term is "zoic", the length of stem ("zo") will be 2 (= 4-2). If the length of stem is too short, usually, the generated derivational variants are not good guess (because it is generated by rules) and should be filter out. The default value of this in Java version is 3. This value really depends on how aggressive you want for the derivational variants.

For example:

Shell> lvg -f:d -kd:3 -cf:2 -if:3 -p -m

zoic|1|1
zoic|1|1|zo|128|1|d|1|RULE|ic$|adj|base|$|noun|base|
zoic|1|1|zoy|128|1|d|1|RULE|ic$|adj|base|y$|noun|base|
zoic|1|1|zoia|128|1|d|1|RULE|ic$|adj|base|ia$|noun|base|
zoic|1|1|zoism|128|1|d|1|RULE|ic$|adj|base|ism$|noun|base|
zoic|1|1|zoicity|128|1|d|1|RULE|$|adj|base|ity$|noun|base|
zoic|1|1|zoicity|128|1|d|1|RULE|ic$|adj|base|icity$|noun|base|
zoic|1|1|zoicness|128|1|d|1|RULE|$|adj|base|ness$|noun|base|
zoic|1|1|zoicly|2|1|d|1|RULE|$|adj|base|ly$|adv|base|

This is the result from C-code (with min. stem length = 0).

If we set min. stem length = 2, the result will become:

zoic|1|1|zoicity|128|1|d|1|RULE|$|adj|base|ity$|noun|base|
zoic|1|1|zoicness|128|1|d|1|RULE|$|adj|base|ness$|noun|base|
zoic|1|1|zoicly|2|1|d|1|RULE|$|adj|base|ly$|adv|base|

because the input stem (zo) for rule ic$ is 2 (must be > then min. stem length).

If we set min. stem length = 3, there is no derivation is generated.

So, which one is the best for MetaMap application?

Also, I have one more question for you.
Issue-5) Does MetaMap get derivational variants for terms not in LEXICON?

We will be able to modify the interface once we have answers for all above issues.
IMHO, I won't be surprised to see the several thousands of difference (improvement :) from the C-code after this new implementation.

I did not know this E-mail is going to be this long when I started it (sorry).
Anyway, hopefully this helps and we can get the derivation right this time for MetaMap application.
Have a nice weekend!

- Chris