A Systematic Approach for Automatically Generating Derivational Variants in Lexical Tools Based on the SPECIALIST Lexicon

Chris J. Lu, Ph.D.[1,2], Lynn McCreedy, Ph.D.[1], Destinee Tormey[1] and Allen C. Browne [1]

[1]National Library of Medicine, Bethesda, MD
[2]Medical Science & Computing Inc., Rockville, MD

A systematic approach to generating derivational variants, including prefixes, suffixes, and zero derivations, from the SPECIALIST Lexicon. This new approach will enhance automatic generation of derivational variants in the SPECIALIST Lexical Tools (NLP Tools), improving both precision and recall rate.

1. Introduction

The demand for natural language processing (NLP) in medicine has grown significantly in recent years. This growth is expected to increase rapidly due to the continuing adoption of electronic medical records (EMRs). Medical language processing (MLP) seeks to analyze linguistic patterns found not only in electronic medical records, but also in published biomedical research, clinical trials reports, and other sources. Due to the great deal of lexical variation in natural language, managing this variability is an important key to successful MLP[1-2]. The National Library of Medicine (NLM) distributes the NLP SPECIALIST Lexicon and Lexical Tools as one of the Unified Medical Language System (UMLS) Knowledge Sources along with the Metathesaurus to provide the NLP/MLP community with rich NLP resources and an extensive NLP toolset since 1994[3]. One of these SPECIALIST tools, Lexical Variant Generation (LVG), is designed to handle lexical variations including derivational variant generation. This paper presents a novel systematic approach to the automatic generation of derivational variants using this tool in conjunction with the SPECIALIST Lexicon.

The SPECIALIST Lexicon

The Lexicon is a large syntactic lexicon of biomedical and general English, designed and developed to provide the lexical information needed for the SPECIALIST Natural Language Processing System[4] which includes SemRep, MetaMap, and the Lexical Tools. The lexicon entry for each lexical item (word or term) records the syntactic, morphological, and orthographic information needed by the SPECIALIST NLP System. This information includes a syntactic category, inflectional variation, spelling variation, abbreviations, acronyms, allowable complementation patterns, etc. Lexical records are built by linguists through a web-based lexicon building tool called LexBuild[5]. A software package, LexCheck[6], validates the syntax and contents of Lexical record(s). This package is integrated in LexBuild to ensure the quality of the Lexicon in real-time. It also provides Java APIs to convert lexical records (Lexicon) into three forms: text, XML, and Java objects, for NLP research using the SPECIALIST Lexicon. In addition to the unit record format, more than 14 LR-files in relational table format (expressing the same information) are generated by computer programs and distributed to maximize their usefulness for different types of NLP applications. Each table contains lexical

information retrieved from the SPECIALIST Lexicon, such as agreement and inflection (LRAGR), abbreviation and acronym (LRABR), spelling variant (LRSPL), etc. The Lexicon is also available for lookup and browsing through a web tool called LexAccess[7]. With this comprehensive computer-aided system, the Lexicon[8] has grown from its first release in 1994 with 66,059 records and 112,990 forms to 462,129 records and 836,066 forms in the 2012 release, providing many NLP projects a corpus with wider coverage and higher quality.

The SPECIALIST Lexical Tools

Lexical variation is a key factor determining precision and recall in NLP applications. The SPECIALIST Lexical Tools[9] include a tool called Lexical Variant Generation (LVG) to handle lexical variation. Furthermore, the Lexical Tools also provide other fundamental and commonly used NLP functions, such as normalization, Unicode to ASCII conversions, tokenization, stopword removal, etc. Each function is represented as a flow component (flow) in LVG. Currently, the Lexical Tools include seven tools, 62 flows, 37 options, and Java APIs in the 2012 release. The Lexical Tools provide a comprehensive toolset for lexical variant generation and other NLP tasks in MLP.

LVG uses the Lexicon as the basis for lexical variant generation. A set of computer programs retrieves lexical information from the Lexicon and automatically generates relational database tables for various lexical variations, such as inflectional variants, acronyms, spelling variants, etc. These tables are updated annually with each new Lexicon release. The derivational variants table, however, is manually maintained because there is no direct derivational information coded in the Lexicon.

2. Derivational Variant Generation in LVG

Derivational processes such as suffixation and prefixation create new words based on existing words. Words are derivational variants of each other if they are related by a derivational process. They need not be synonymous, and in fact, derivation often entails thoroughgoing meaning change. For example, the adjective "unkind", the adverb "kindly," and the noun "kind" are all derived from the adjective "kind", by the derivational processes of prefixation ("un"), suffixation ("ly") and zero derivation (category change without affixation, here, changing adjective to noun), respectively. Since we are interested in relatedness rather than history, we do not record the direction of derivation but consider each member of a derivational pair (dPair) a derivational variant of the other without regard to which came first. The information that such a dPair exists (not including which word is the root word) is coded in LVG for use in NLP applications.

Figure-1 shows the derivational network for the "kind" family. Each link and the associated two nodes in derivational network define a dPair. For example, kindness|noun and kind|adj are a dPair because they connect directly. This dPair is coded in LVG's derivational fact table as kindness|noun|kind|adj. Derivational pairs include base forms as well as syntactic category information, are bi-directional, and can be categorized into three types: prefix derivation (prefixD), suffix derivation (suffixD), and zero derivation (zeroD). Each dPair can only involve one derivational affix, or none, in the case of zero derivation. This is not to say that each pair of terms can only contain one derivational affix; just that only one affix will be pertinent to a given dPair.

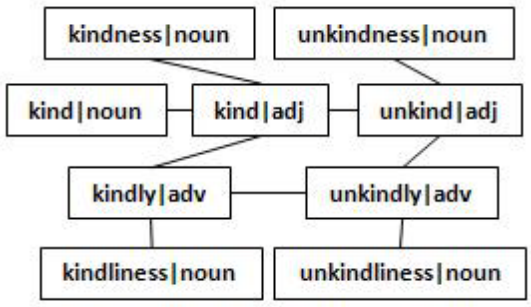

Figure-1. Derivational Network Example

In contrast, terms that are not connected directly do not comprise a dPair. For example, kindness|noun and kindly|adv are not a dPair because they connect through kind|adj. LVG handles both cases via two derivational generation flow components: direct (-f:d) and recursive (-f:R). The recursive derivational flow also provides the distance (number of dPairs

involved). For example, kindness|noun and kindly|adv have a distance of 2 because two dPairs are involved.

Use of the LVG derivational flow components allows users to find closely related terms that may differ in syntactic category, but are nonetheless usefully related. For example, if the source vocabulary includes hyperuricemic|adj, the derivational variant generation flow will map it to hyperuricemia|noun, which is a UMLS Metathesaurus[10] term. More information, such as concepts (C0740394) and synonyms, can be retrieved for further NLP analysis.

The LVG derivational flow component is based on a Rules and Facts paradigm designed to capture the morphological relations between terms. It is handled by a list of known dPairs (Facts) and a set of rules with exceptions (Rules). Derivational rules should have two characteristics: high frequency and high precision rate. In practice, Rules-generated derivational variants tend to have higher coverage (recall rate) with lower precision rate. A list of known exceptions for each rule is added to increase the precision. Both derivational Facts and Rules are manually maintained by linguists before the 2012 release. The maintenance task involves collecting, validating, and tagging dPairs. This process requires derivational analysis by linguists. The derivational analysis is complicated when more than one affix is involved. For example, multioptional|adj could be derived from optional|adj with the prefix "multi" or from multioption|noun with suffix "al". It could get even more complicated when more affixes are involved, such as "pseudo-hyper-para-thyroid-ism". In such cases, the usage of all related words must be checked and the order of derivation must be determined for accurate analysis and tagging. This process is time consuming and labor intensive. Over the years, derivational Facts and Rules in LVG have not grown proportionally with the growth of the Lexicon because of this difficulty and limited resources. This paper describes a systematic approach to the updating dPairs and generating the derivational table (Facts) in LVG from the Lexicon. Both precision and recall rate are improved. This study also provides analysis and refinement of derivational Rules development.

3. Approaches

This section describes a systematic approach to semi-automated data mining processes for generating dPairs of prefixD, suffixD, and zeroD for LVG by using information already contained in the Lexicon (The SPECIALIST Lexicon 2012 release).

Prefix Derivations

For prefix derivations, a prefix is placed at the beginning of a base word to form another word. This process results in prefixD pairs. A series of computer-aided processes has been developed to generate prefixD pairs as follows.

First, the Lexical Systems Group (LSG) collects 143 unique and commonly used prefixes for derivations. Both prefixes (e.g. "re") and combining forms (e.g. "multi") are included in this prefix list because both could generate prefixD pairs. This prefix list[11] is subject to annual update with the Lexical Tools release. Second, all base forms (citations and spelling variants) are retrieved from the Lexicon. Third, three types of raw prefixD pairs are generated if they match the prefix patterns shown below:

prefix: nonsignificant|significant
prefix and a dash: non-significant|significant
prefix and a space: non significant|significant

There are 115,139 raw prefixD pairs found in this step from the Lexicon, as shown in the last row of Table-1. This list is then sent to linguists for final tagging. Due to limited resources for this labor intensive process, LSG decided to tag only the most frequent and user requested prefixes in the Lexicon for the 2012 release. 65.67% (56,694) of the 86,333 tagged prefixD pairs are valid (as shown in the second to the last row in Table-1). Among valid prefixD pairs, 24.54% (13,914) involved category change, such as the prefixD pair fog|noun|antifog|adj, in which the category changes from noun to adjective. Also, 0.83% (472) of valid prefixD pairs involved abbreviations or acronyms (such as "MDR", acronym for "multidrug resistance" which occurs in the valid prefixD pair

MDR|noun|antiMDR|adj). Accordingly, no category filter, abbreviation filter or acronym filter was implemented in the prefixD generation program to preserve the high recall rate. The prefixD tagging results are recorded so that only data newly added to the Lexicon will be tagged for future releases.

Columns 1, 2, 3, and 4 in Table-1 show the frequency ranking (rows 2 ~ 6), prefix word, raw prefixD counts (percentage) and valid prefixD counts (percentage) of prefixD pairs found in the Lexicon, respectively. As shown in column 4, the maximum valid rate (80.31%) of the prefix "post" and the average valid rate (65.67%) are not high enough to implement prefixD rules to auto-generate prefixD pairs.

	Prefix	Raw prefixD	Valid prefixD
1	non	16,471 (14.31%)	12,598 (76.49%)
2	pre	9,651 (8.38%)	7,224 (74.85%)
3	post	9,490 (8.24%)	7,621 (80.31%)
4	anti	6,500 (5.65%)	5,051 (77.71%)
5	sub	4,262 (3.70%)	2,698 (63.30%)
...	...	...	...
Tag		86,333 (74.98%)	56,694 (65.67%)
Raw		115,139 (100%)	

Table-1. Statistical Data for the Most Frequent, Tagged, and Raw PrefixD Pairs

Suffix Derivation Facts and Nominalizations

In linguistics, a suffix is an affix which is placed after the stem of a word. We limit our scope on suffix derivation Facts (SD-Facts) to suffixes which create nominalizations because this information is encoded in the SPECIALIST Lexicon. Nominalization is a process that relates a verb or adjective to a synonymous noun with matching complementation. Nominalization derivation (nomD) is a type of suffixD. A series of computer-aided processes has been developed to retrieve nomD as follows.

First, nominalization information in the lexical records is retrieved from the Java object format of the SPECIALIST Lexicon. For example, state|verb|statement|noun is retrieved from nominalization=statement|noun|E0057700 in the lexical record for the verb "state" in the Lexicon, as shown below:

```
{base=state
entry=E0057695
        cat=verb
        variants=reg
        tran=fincomp(t)
        tran=np
        tran=whfincomp
        tran=whinfcomp:arbc
        cplxtran=np,infcomp:objr
        nominalization=statement|noun|E0057700
}
```

Please note that the nominalization is symmetric (bi-directional). Hence, the code nominalization_of=state|verb|E0057695 is in the lexical record of "statement". In our system, only one nomD pair (of these two symmetric nomD pairs) is added to remove the redundancy. 14,445 raw nomD pairs are found in the Lexicon. NomD pairs are over-generated and filter algorithms subsequently eliminate invalid nomD pairs as follows:

1). Pattern filter: the most common way to nominalize a verb is by adding a suffix. However, not every nominalization occurs that way. Thus, not every nominalization is a derivation. Nominalizations with verb particles are identified as invalid derivations. Four patterns of verb particle nominalizations are identified as invalid nomD pairs and associated examples are illustrated as follows:

Pattern-1: baseParticle|noun|base|verb
⇨ backup|noun|back|verb
Pattern-2: base-Particle|noun|base|verb
⇨ cut-through|noun|cut|verb
Pattern-3: inflParticle|noun|base|verb
⇨ grownup|nou|grow|verb
Pattern-4: infl-Particle|noun|base|verb
⇨ salting-in|noun|salt|verb

The "base" and "infl" represent base forms and inflectional variants of the base forms, respectively. Particles are classified as prepositions in the Lexicon. However, the preposition "per" is not included in the particle list because it filters out valid nomD pairs. For example, shopper|noun|shop|verb is a valid nomD pair and should not be removed.

2). Exception filter: other known invalid nomD pairs from nominalizations are filtered out as exceptions. These are identified by linguists from a computer-generated list comparing the first and last three characters between base forms of dPairs. Some of these exceptions are listed as follows:

```
face-saving|noun|save|verb
decision-making|noun|make|verb
lovemaking|noun|make|verb
...
```

As a result, 0.5% (72) nomD pairs are removed by filter programs and 99.5% (14,373) of nomD pairs (14,445) are valid dPairs from Lexicon. These program generated nomD pairs are used for SD-Facts table in LVG.

Suffix Derivation Rules (SD-Rules)

In addition to SD-Facts, LVG also uses SD-Rules to generate suffixD variants to cover suffixD that are not nomD. LSG derives 97 SD-Rules[12] from the most common English suffixes for derivations in LVG. For example, the suffix "ment" can be added to a verb to create a noun, which is then the suffix derivational variant of the word. Thus, adding "ment" to "retire" creates "retirement", expressible as the suffixD pair retire|verb|retirement|noun. SD-Rules can be applied to generate suffixD pairs in both directions. This SD-Rule is coded in the following format in LVG:

```
$|verb|ment$|noun
```

"$" means the end of the word. SD-Rules are stored and retrieved through a persistent Trie[13] mechanism for generating suffixD variants in the LVG rule based generation. Again, the SD-Rules over-generate suffixD pairs. Four heuristic algorithms are implemented in LVG to eliminate these non-realistic derivational variants and increase precision:

1). Exception filter: there are exceptions (invalid dPairs) for each SD-Rule. For example, depart|verb|department|noun is an invalid suffixD pair that is filtered out and added to the exception list for the SD-Rules listed above. Exceptions for each rule are maintained by linguists and implemented as part of Trie.

2). Min. length of a word: if the length of a term is too short (less than 3 as default), the word is usually an acronym or abbreviation; thus, SD-Rules should not be applied. For example, mo|verb generated from moment|noun is an invalid suffixD pair and is removed because the length of "mo" is too short (2).

3). Min. length of stem in the Trie: the stem length is the length of the word minus the length of its suffix. If the length of the stem is too short (less than 3 as default), usually the generated suffix derivational variants are invalid. For example, the stem size of "lament" is 2 (6-4) and thus the invalid suffixD pair lament|noun|la|verb is removed.

4). Domain filter: this filter allows users to eliminate invalid results in which the SD-Rules generate suffixD pairs that are not both in the Lexicon. For example, "colorment|noun", an SD-Rules generated derivational variant of "color|verb", is eliminated because it is not in the Lexicon.

The above 2-4 options are configurable in LVG to provide more flexibility for different NLP goals.

SD-Rules Validation

We developed a set of programs to validate SD-Rules using SD-Facts. First, a program is used to identify possible SD-Rules by stripping the same starting characters of each valid dPair in SD-Facts. For example, a SD-Rule of ion$|noun|e$|verb is identified by stripping "locat" from "location" and "locate" in the dPair location|noun|locate|verb. In this way, 496 possible SD-Rules are identified from SD-Facts. These identified SD-Rules must be further analyzed and decomposed by adding linguistic knowledge to form better SD-Rules because not all of these identified SD-Rules have high enough frequency and precision rates. For example, the SD-Rule ion$|noun|e$|verb is identified with 1,694 instances in the SD-Facts. This rule can be further categorized into seven linguistic SD-Rules, as shown in Table-2. The two most frequent SD-Rules of this example are used in LVG. Table-3 shows seven SD-Rules

from the five most frequent SD-Rules identified from the SD-Facts are used in LVG.

Linguistic SD-Rules	Example	No.
ation$\|noun\|ate$|verb	location|noun|locate|verb	1,547
sion$\|noun\|se$|verb	tension|noun|tense|verb	77
ution$\|noun\|ute$|verb	delution|noun|delute|verb	37
etion$\|noun\|ete$|verb	completion|noun|complete|verb	22
otion$\|noun\|ote$|verb	devotion|noun|devote|verb	6
ition$\|noun\|ite$|verb	ignition|noun|ignite|verb	4
cion$\|noun\|ce$|verb	coercion|noun|coerce|verb	1

Table-2. SD-Rules from ion$|noun|e$|verb

Identified Rules	SD-Rules in LVG	Counts
ness$\|noun\|$|adj	ness$\|noun\|$|adj	2,481
ion$\|noun\|e$|verb	ation$\|noun\|ate$|verb	1,547
	sion$\|noun\|se$|verb	77
	Others ...	70
ity$\|noun\|$|adj	ity$\|noun\|$|adj	881
	icity$\|noun\|ic$|adj	745
ility$\|noun\|le$|adj	ability$\|noun\|able$|adj	1,036
	Others ...	253
ation$\|noun\|e$|verb	ation$\|noun\|e$|verb	1,133

Table-3. Five Most Frequent SD-Rules Identified from SD-Facts

Zero Derivations

Zero derivation is a linguistic process that assigns an already existing word to a new syntactic category without any concomitant change in form. This process is also known as a functional shift or conversion. For example, flex|noun|flex|verb is a zeroD pair. As expected, the zeroD pair has the same base form ("flex") and different category (noun and verb). A series of computer-aided processes has been developed to generate zeroD pairs as follows.

First, the base forms and category information can be retrieved because they are coded in the Lexicon. All words from the Lexicon with the same base form but different categories are paired up as a raw zeroD pair list. Next, a filter algorithm is applied to eliminate two types of invalid zeroD pairs as follows: 1). abbreviations and acronyms are invalid derivations; 2). all words with a length less than two are invalid derivations. This information can be retrieved from the Lexicon in the Java object format for the filter algorithm. For example, the invalid zeroD pair AAIR|noun|AAIR|adj is removed because "AAIR" is coded as an acronym in the Lexicon.

At this point, the filtered zeroD pairs list includes all possible zeroD pairs. This list is then sent to linguists for final tagging to remove invalid zeroD pairs. For example, round|adj|round|prep is a invalid zeroD pair because their etymologies are unrelated. The tags of all zeroD pairs are recorded so that for future releases, only newly added Lexicon data will need to be tagged. The result shows that 10.52% (1,935) raw zeroD pairs (18,400) are automatically filtered out by filter programs and 80.14% (14,747) of raw zeroD pairs are valid. Given these results, no zeroD Rules are identified because of the relatively low precision rate of valid dPairs (80.14%).

Final Compile

All dPairs from prefixD, suffixD, and zeroD need to be validated by an affix validation program by checking the first and last three characters between base forms to assure only one affix is involved. An exception filter is used in this program to preserve valid dPairs. For example, long|adj|length|noun is valid (an exception) even though "long" and "length" have different first and last three characters. This exception filter also accounts for spelling variants. For example, "dysmature" is a spelling variant of "dismature". Therefore, the exception filter passes dysmaturity|noun|dismature|adj as a valid dPair. Finally, the three validated lists of dPairs (prefixD, suffixD, and zeroD) are combined and used as Facts in LVG derivational variants generation.

4. Conclusion & Future Work

Automatic derivational variant generation is a complicated task. LSG developed a systematic data mining approach to retrieve raw dPairs by patterns matched (prefixD and zeroD) and embedded codes retrieval (suffixD), applied various filter algorithms to eliminate invalid dPairs, and integrated the results with expert tagging processes to accomplish this task in the Lexical Tools 2012 release. With this approach, the coverage (recall rate) of derivational variant generation in LVG will grow proportionally

with Lexicon growth. The result has been a dramatic improvement from 4,559 to 89,950 dPairs in Facts used in LVG. Ideally, the precision in Facts should reach virtually 100%, assuming an error-free tagging process. These improvements in both precision and recall rates provide better results in NLP applications with the use of the SPECIALIST Lexical Tools.

For future releases, in addition to the annual update processes to generate dPairs from the latest Lexicon, three new tasks will be necessary:

1). Update the prefix list and complete tagging processes for all collected prefixes to increase coverage of prefixD pairs.

2). Develop a set of processes to retrieve more dPairs in suffixD Facts by suffix list (not limited to nomD) and thoroughly validate LVG SD-Rules and associated exceptions by all possible raw suffixD pairs in the Lexicon to ensure the quality of generated suffixD pairs.

3). Further investigate the possibility of including syntactic category and other linguistic knowledge for rules-based generated dPairs and more rules-based filters on zeroD and prefixD pairs.

References

1. Pacak MG, Norton LM, and Dunham GS, "Morphosemantic analysis of -ITIS forms in medical language", Methods InfMed 1980, vol. 19, pp. 99-105
2. Wolff S, "Automatic coding of medical vocabulary", In: Sager N, Friedman C, and Lyman MS, eds, Medical Information Processing – Computer Management of Narrative Data. Addison Wesley, Reading Mass, 1987, pp. 145-62
3. A T McCray, A R Aronson, A C Browne, T C Rindflesch, A Razi, and S Srinivasan, "UMLS Knowledge for biomedical language processing", Bull Med Libr Assoc. 1993 April, 81(2), pp. 184-194.
4. A.T. McCray, S. Srinivasan, A.C. Browne, "Lexical Methods for Managing Variation in Biomedical Terminologies", the Proceedings of the 18th Annual Symposium on Computer Applications in Medical Care, 1994, pp. 235-239
5. Lexical systems Group, LexBuild project: http://umlslex.nlm.nih.gov/lexBuild
6. Lexical systems Group, LexCheck project: http://umlslex.nlm.nih.gov/lexCheck
7. Lexical systems Group, LexAccess project: http://umlslex.nlm.nih.gov/lexAccess
8. Lexical systems Group, Lexicon project: http://umlslex.nlm.nih.gov/lexicon
9. Lexical systems Group, Lexical Tools project: http://umlslex.nlm.nih.gov/lvg
10. Unified Medical Language System project: http://umls.nlm.nih.gov
11. Lexical Tools – Derivational Prefix List: http://lexsrv3.nlm.nih.gov/LexSysGroup/Projects/lvg/2012/docs/designDoc/UDF/derivations/prefixList.html
12. Lexical Tools – Derivational Suffix Rules: http://lexsrv3.nlm.nih.gov/LexSysGroup/Projects/lvg/2012/docs/designDoc/UDF/derivations/suffixRules.html
13. Alfred V. Aho, Jeffrey D. Ullman, John E. Hopcroft, "Data Structure and Algorithms" Addison Wesley, 1983. pp. 163-169.

Chris J Lu is a systems architect in Lexical Systems Group (LSG) at the NLM. He received his PhD in Computer Integrated Manufacturing and Design (CIMAD) in the mechanical engineering department from University of Maryland. He is a member of AMIA. Contact him at lu@nlm.nih.gov.

Lynn McCreedy is a linguist and lexicographer for the UMLS SPECIALIST Lexicon. She received her PhD in linguistics from Georgetown University. Contact her at mccreedy@nlm.nih.gov.

Destinee Tormey is a linguist and lexicographer for the UMLS SPECIALIST Lexicon. She received an M.S. in Computational Linguistics from Georgetown University. Contact her at dln4@georgetown.edu.

Allen C. Browne is an information research specialist at the Lister Hill Center in the National Library of Medicine where he is the leader of the Lexical Systems Group. Contact him at browne@nlm.nih.gov.