A Systematic Approach for Medical Language Processing: Generating Derivational Variants

Chris J. Lu, Lynn McCreedy, Destinee Tormey, and Allen C. Browne, *US National Library of Medicine*

Medical language processing seeks to analyze linguistic patterns in electronic medical records, which requires managing lexical variations. A systematic approach to generating derivational variants, including prefixes, suffixes, and zero derivations, has improved precision and recall rates.

The demand for natural language processing (NLP) in medicine has grown significantly in recent years and is only expected to increase due to the continuing adoption of electronic medical records (EMRs). Medical language processing (MLP) seeks to analyze linguistic patterns found not only in EMRs but also in published biomedical research, clinical trial reports, and other sources. Natural language has a great deal of lexical variation, so managing this variability is an important key to MLP's success.[1,2]

Since 1994, the US National Library of Medicine (NLM) has been distributing its SPECIALIST Lexicon, a large syntactic lexicon of biomedical and general English, as one of the Unified Medical Language System (UMLS) knowledge sources. The lexicon (denoted "Specialist" Lexicon throughout), along with Lexical Tools and the Metathesaurus (a collection of multiple vocabularies, code sets, and standards), provide the NLP and MLP communities with rich NLP resources and an extensive NLP toolset.[3] One of these tools, Lexical Variant Generation (LVG), is designed to handle lexical variations (a key factor determining precision and recall in NLP applications) and generate derivational variants.

Here, we present a novel systematic approach to automatically generating derivational variants using LVG in conjunction with the Specialist Lexicon.

The Specialist Lexicon

The Lexicon was designed to provide the lexical information needed for the Specialist NLP system,[4] which includes SemRep (an application for automatic semantic interpretation), MetaMap (an application that maps biomedical text to the Metathesaurus concepts), and the Lexical Tools. Each lexicon entry records the syntactic, morphological, and orthographic information that the Specialist NLP system needs. This information includes a syntactic category, inflectional variation, spelling variation, abbreviations, acronyms, and allowable complementation patterns.

Building Lexical Records

NLM linguists build lexical records through a Web-based tool called LexBuild (http://umlslex.nlm.nih.gov/lexBuild). Integrated into this tool is the LexCheck software package (http://umlslex.nlm.nih.gov/lexCheck), which validates the records' syntax and contents in real time. LexCheck also provides Java APIs to convert the lexical records into three forms—text, XML, and Java objects—for NLP research using the Specialist Lexicon. In addition to these three formats, computer programs generate more than 14 LR (lexical records) files in a relational table format (expressing the same information). The programs then distribute these files to maximize their usefulness for different NLP applications. Each table contains lexical information retrieved from the Specialist Lexicon, such as lexical-record agreement and inflection, abbreviation and acronym, and spelling variations.

The Lexicon (http://umlslex.nlm.nih.gov/lexicon) comprises all of this information and makes it searchable through a Web tool called LexAccess (http://umlslex.nlm.nih.gov/lexAccess). With this comprehensive computer-aided system, the Lexicon has grown from 66,059 records and 112,990 forms with its first release in 1994 to 462,129 records and 836,066 forms in the 2012 release, providing many NLP projects a corpus with wider coverage and higher quality.

Lexical Variant Generation

The Specialist Lexical Tools provide a comprehensive toolset for lexical variant generation and other NLP tasks in MLP. In particular, the set includes the LVG tool (http://umlslex.nlm.nih.gov/lvg), which helps with lexical variation as well as other fundamental and commonly used NLP functions, such as normalization, Unicode-to-ASCII conversions, tokenization, and stopword removal. Each function is represented as a flow component (flow) in LVG. The 2012 release of the Lexical Tools set includes seven tools, 62 flows, 37 options, and Java APIs.

LVG uses the Lexicon to generate lexical variants. A set of computer programs retrieves lexical information from the Lexicon and automatically generates relational database tables for lexical variations. These tables are updated annually with each new Lexicon release. The derivational variants table, however, is manually maintained because there's no direct derivational information coded in the Lexicon.

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're related by a derivational process. They need not be synonymous, and, in fact, derivation often entails a complete change in meaning.

The Lexicon has grown to 462,129 records and 836,066 forms, providing many NLP projects a corpus with wider coverage and higher quality.

For example, the adjective "unkind," the adverb "kindly," and the noun "kind" are all derived from the adjective "kind" though the derivational processes of prefixation ("un"), suffixation ("ly"), and zero derivation (category change without affixation—in this case, changing an adjective to a noun), respectively. Because we're interested in relatedness rather than history, we don't 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. We code the information that such a dPair exists (not including which word is the root word) 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

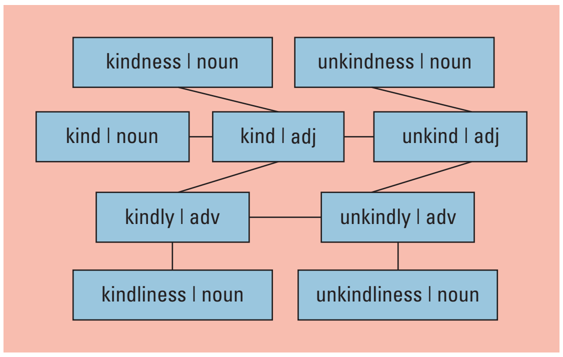

Figure 1. Derivational network example for the "kind" family. Each link and the associated two nodes in derivational network define a dPair.

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 bidirectional, 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 isn't 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.

In contrast, terms that aren't directly connected don't constitute a dPair. For example, kindness|noun and kindly|adv aren't 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 two because two dPairs are involved.

Using the LVG derivational flow components lets users find closely related terms that might 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 term (http://umls.nlm.nih.gov). More information, such as concepts (C0740394—the concept unique identifier for "hyperuricemia") and synonyms, can be retrieved from the Metathesaurus 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's handled by a list of known dPairs (Facts) and a set of rules with exceptions (Rules). Derivational rules should exhibit high frequency and have a high precision rate. In practice, Rules-generated derivational variants tend to have higher coverage (recall rates) with lower precision rates. A list of known exceptions for each rule is added to increase the precision.

Before the 2012 release, our linguists manually maintained derivational Facts and Rules. The maintenance task involved collecting, validating, and tagging dPairs, and it required derivational analysis, which 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's even more complicated when more affixes are involved, such as "pseudo-hyper-para-thyroid-ism." In such cases, we must check the usage of all related words and determine the order of derivation for accurate analysis and tagging. This process is time consuming and labor intensive, which is why derivational Facts and Rules in LVG haven't grown proportionally with the Lexicon.

A Systematic Approach

Here we present a systematic approach to semi-automated data-mining processes for generating dPairs of prefixD, suffixD, and zeroD for LVG using information already contained in the 2012 Lexicon.

Prefix Derivations

Placing a prefix at the beginning of a base word to form another word creates a prefixD pair. We developed a series of computer-aided processes to generate prefixD pairs. First, the Lexical Systems Group (LSG) collects 143 unique and commonly used prefixes for derivations. We include both prefixes (such as "re") and combining forms (such as "multi"), because both could generate prefixD pairs. The LSG updates this prefix list[5] annually with the Lexical Tools release.

Second, the LSG retrieves all base forms (citations and spelling variants) from the Lexicon.

Third, three types of raw prefixD pairs are generated if they match the following prefix patterns:

- prefix: nonsignificant|significant;
- prefix and a dash: non-significant|significant; or
- prefix and a space: non significant|significant.

We then sent the resulting 115,139 raw prefixD pairs to our linguists for final tagging.

Because of limited resources for this labor-intensive process, the LSG decided to tag only the most frequent and user-requested prefixes in the Lexicon for the 2012 release. Of the 86,333 tagged prefixD pairs, 65.67 percent (56,694) were valid. Among the valid prefixD pairs, 24.54 percent (13,914) involved a category change, such as the prefixD pair fog|noun|antifog|adj, in which the category changes from noun to adjective. Also, 0.83 percent (472) of valid prefixD pairs involved abbreviations or acronyms (such as "MDR," the 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 were recorded so that only data newly added to the Lexicon will be tagged for future releases.

Table 1 shows the frequency ranking, prefix word, raw prefixD counts (percentage), and valid prefixD counts (percentage) of prefixD pairs found in the Lexicon. The maximum valid rate (80.31 percent) of the prefix "post" and the average valid rate (65.67 percent) weren't high enough to implement prefixD rules to autogenerate prefixD pairs.

Table 1. Statistical data for the most frequent, tagged, and raw prefixD pairs.

Frequency ranking	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%)
Tagged pairs		86,333 (74.98%)	56,694 (65.67%)
Raw pairs		115,139 (100%)	

Suffix Derivation Facts and Nominalizations

In linguistics, a suffix is an affix placed after a word stem. We limit our scope on suffix derivation Facts (SD-Facts) to suffixes that 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.

We developed a series of computer-aided processes to retrieve nomD. 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 follows:

```
{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
}
```

Note that the nominalization is symmetric (bidirectional), so 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. We found 14,445 raw nomD pairs in the Lexicon. NomD pairs are overgenerated, and filter algorithms subsequently eliminate invalid nomD pairs.

Pattern filter. The most common way to nominalize a verb is to add a suffix, but not every nominalization occurs that way. Thus, not every nominalization is a derivation. Nominalizations with verb particles are identified as invalid derivations. We identify four patterns of verb particle nominalizations as invalid nomD pairs and presented associated examples in Table 2.

In Table 2, 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" isn't included in the particle list because it filters out valid nomD pairs. For example, shopper|noun|shop|verb is a valid nomD pair and shouldn't be removed.

Table 2. Verb particle nominalizations identified as invalid nominalization derivation (nomD) pairs.

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

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

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

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

Suffix Derivation Rules

In addition to SD-Facts, LVG also uses SD-Rules to generate suffixD variants to cover suffixDs that aren't nomD. LSG derives 97 SD-Rules[6] 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, where "$" means the end of the word. SD-Rules are stored and retrieved through a persistent Trie[7] 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 nonrealistic derivational variants and increase precision.

The first is the exception filter, which handles 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 earlier. Linguists maintain exceptions for each rule and implement them as part of Trie.

The second algorithm addresses minimum word length. If a term is too short (less than three characters is the default), the word is usually an acronym or abbreviation and SD-Rules shouldn't be applied. For example, mo|verb generated from moment|noun is an invalid suffixD pair and is removed because "mo" is too short.

The third algorithm addresses the minimum stem length in the Trie. The stem length is the length of the word minus the length of its suffix. If the stem is too short (less than three characters is the default), usually the generated suffix derivational variants are invalid. For example, the stem size of "lament" is two, so the invalid suffixD pair lament|noun|la|verb is removed.

The final algorithm is the domain filter. It lets users eliminate invalid results in which the SD-Rules generate suffixD pairs that aren't both in the Lexicon. For example, "colorment|noun," an SD-Rules generated derivational variant of "color|verb," is eliminated because it's not in the Lexicon.

These last three algorithms 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 identifies possible SD-Rules by stripping the same starting characters of each valid dPair in SD-Facts. For example, an 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, we can identify 496 possible SD-Rules from SD-Facts.

These identified possible SD-Rules must be further analyzed and decomposed by adding linguistic knowledge to form more finely-grained SD-Rules that have high precision and frequency

Table 3. Suffix Derivation (SD) Rules from ion$|noun|e$|verb ("$" indicates the end of the word).

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

so they can be used in LVG for automatic Rules-generated derivations. For example, the SD-Rule ion$|noun|e$|verb is identified with 1,694 instances in the SD-Facts. We can further analyze this rule into seven linguistic SD-Rules (see Table 3). The two most frequent SD-Rules of this example are used in LVG. Table 4 shows that LVG uses seven SD-Rules from the five most frequently identified SD-Rules from the 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). We developed a series of computer-aided processes to generate zeroD pairs.

First, the base forms and category information can be retrieved because they're 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: abbreviations and acronyms and all words with a length of less than two. 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. Our linguists then go through this list for final tagging to remove invalid zeroD pairs. For example, round|adj|round|prep is an invalid zeroD pair because their etymologies are unrelated. The tags of all zeroD pairs are recorded so that future releases will require only newly added Lexicon data to be tagged.

The result shows that programs automatically filtered out 10.52 percent (1,935) raw zeroD pairs (18,400), so 80.14 percent (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 percent).

Final Compile

An affix validation program validates all dPairs from prefixD, suffixD, and zeroD by checking the first and last three characters between base forms to ensure only one affix is involved. An exception filter used in this program preserves 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.

Table 4. The five most frequent SD-Rules identified from SD-Facts.

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

Finally, the three validated lists of dPairs (prefixD, suffixD, and zeroD) are combined and used as Facts in LVG derivational variants generation.

Using the systematic data-mining approach, various filter algorithms, and expert tagging processes for the Lexical Tools 2012 release resulted in a dramatic increase in dPairs Facts in the LVG—from 4,559 to 89,950. Ideally, the precision in Facts should reach virtually 100 percent, assuming an error-free tagging process. These improvements in both precision and recall rates provide better results in NLP applications when using 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. First, we'll update the prefix list and complete tagging processes for all collected prefixes to increase coverage of prefixD pairs.

Second, we'll 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.

Finally, we'll 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. **IT**

References

1. M.G. Pacak, L.M. Norton, and G.S. Dunham, "Morphosemantic Analysis of -ITIS Forms in Medical Language," *J. Methods of Information in Medicine*, vol. 19, no. 2, 1980, pp. 99–105.
2. S. Wolff, "Automatic Coding of Medical Vocabulary," *Medical Information Processing—Computer Management of Narrative Data*, Addison Wesley, 1987, pp. 145–162.
3. A.T. McCray et al., "UMLS Knowledge for Biomedical Language Processing," *Bull. Medical Library Assoc.*, vol. 81, no. 2, 1993, pp. 184–194.
4. A.T. McCray, S. Srinivasan, and A.C. Browne, "Lexical Methods for Managing Variation in Biomedical Terminologies," *Proc. 18th Ann. Symp. Computer Applications in Medical Care*, Am. Medical Informatics Assoc., 1994, pp. 235–239.
5. "Derivational Prefix List," Lexical Tools 2012, Nov. 2011; http://lexsrv3.nlm.nih.gov/LexSysGroup/Projects/lvg/2012/docs/designDoc/UDF/derivations/prefixList.html.
6. "Derivational Suffix Rules," Lexical Tools 2012, Oct. 2011; http://lexsrv3.nlm.nih.gov/LexSysGroup/Projects/lvg/2012/docs/designDoc/UDF/derivations/suffixRules.html.
7. A.V. Aho, J.D. Ullman, J.E. Hopcroft, *Data Structure and Algorithms*, Addison Wesley, 1983, pp. 163–169.

Chris J. Lu *is a systems architect in the Lexical Systems Group (LSG) at the US National Library of Medicine. His research interests include natural language processing, software development, and system integration. Lu received his PhD in computer integrated manufacturing and design (CIMAD) in the mechanical engineering department of the University of Maryland. He's a member of the American Medical Informatics Association (AMIA). Contact him at lu@nlm.nih.gov.*

Lynn McCreedy *is a linguist and lexicographer for the UMLS Specialist Lexicon at the US National Library of Medicine. Her research interests include discourse analysis and sociolinguistics. McCreedy 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 at the US National Library of Medicine. Her research interests include natural language processing, corpus linguistics, and computational linguistics. Tormey received her MS 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 US National Library of Medicine, where he's the leader of the Lexical Systems Group. His research interests include natural language processing, biomedical informatics, and sociolinguistics. Browne received his MS in linguistics from Georgetown University. Contact him at browne@nlm.nih.gov.*