An Algorithmic Approach to English Pluralization

Damian Conway

School of Computer Science and Software Engineering
Monash University
Clayton 3168, Australia

mailto:damian@csse.monash.edu.au
http://www.csse.monash.edu.au/~damian

Abstract

This paper discusses some of the issues involved in designing robust and comprehensive algorithms which convertsingular English nouns, verbs and adjectives to their appropriate plural forms. Four such algorithms are given: one for each part of speech which inflects in the plural, and a unified algorithm for all such parts of speech. A word comparison algorithm that can identify words that differ only in their grammatical number is also given. Finally, an overview is given of a full implementation of the various algorithms in the Perl [1] programming language.

The problem of English plurals

The English language is overburdened with idiosyncratic grammatical features, a legacy of its eclectic accretion over 1500 years [2,3]. One unfortunate consequence of this otherwise admirable richness is that automatically generating correct English is fraught with difficulty. Composing the simplest of sentences may require quite sophisticated semantic understanding to enable the correct syntax to be chosen. Even at the lexical level it can be a complex matter to correctly inflect the individual words of a sentence to reflect their number, person, mood, case, etc.

The use of English plurals in synthetic sentences is a case in point. In computing applications, for example, it is quite common to encounter error messages which jar because they do not correctly inflect for grammatical number:

Compilation aborted: 1 errors were found

Individually, such inelegances are easily overcome (or, more accurately, the inelegance may be transferred from the interface to the code):

```
print "Compilation aborted: $count ",
      ($count==1  ? "error was"
                  : "errors were"),
      " detected.\n";
```

Unfortunately, in attempting to generate more complex text, some less tractable problems arise, notably the diversity of plural forms available in English. Consider the difficulty faced by a text generation system (machine or human) in forming plural versions of the following:

Her criterion differs from mine.

Analysis of this aquarium's fish failed to determine its genus.

That phalanx suffered a trauma.

This paper presents an algorithmic approach that provides (nearly) automatic plural inflections for such examples.

Coping with English plurals in synthetic text

Existing techniques for dealing with plural inflections in generated text fall into a four categories: indifference, evasion, explication, and automation. The following sections briefly describe each of these.

Ignoring the problem

Ignoring issues of pluralization has a long and glorious history in certain synthetic text generation contexts. Typically, when this approach is used, the programmer simply assumes that the number required will always be non-singular and that any cases where a singular does appear will be written off by the user as a "computer glitch" or tolerated as a flaw in the interface. Hence the familiar *There were 1 errors* message.

One might argue that this approach is economically rational, in that the extra cost and complexity involved in identifying and coding around that one special case outweighs the benefit of correctly handling it. This, of course, is the perennial excuse for ugly and ungainly interfaces, and quite unassailable in the estimation of the utilitarian mind.

Avoiding the problem

English is sufficiently flexible that programmers, faced with the task of generating text of a changeable number, may easily enough recast their synthetic prose into "number-inclusive" forms. The simplest approach is to structure the text so that the grammatical number of the various parts of speech in a sentence is fixed, regardless of the actual number of items being referred to. Hence:

Number of errors: 1
Number of errors: 10

A common (if somewhat clumsy) alternative is to bet both ways and structure the sentence so that it will read correctly in either grammatical number:

1 error(s) found.

10 error(s) found.

Evasion techniques such as these solve the problem of "canned" synthetic text, but do so either by craving the readers' indulgence (of threadbare English) or their complicity (in ignoring the inappropriate sense of a schizophrenic construction). However, in general text generation, such terse and artificial structures may be inappropriate or simply unachievable.

A "manual" scheme

One variation on the "each-way bet" approach is for the programmer to explicitly provide both singular and plural forms and then have the system select the correct form according to the actual number required, For example, consider a subroutine:

```
sub select_pl($$)
{
    my ($word, $count) = @_;
    $word =~ s{\(([^)/]*)/([^)]*)\)}
              {$count==1 ? $1 : $2 #ge};
    return $word;
}
```

which allows the programmer to code synthetic text generation as follows:

```
print $count,
 select_pl(" error(/s) w(as/ere)",$count),
 " found\n";
```

This approach neatly solves the problem of correctly inflecting "canned" text for number, but is not easily adapted to handle the more general problems encountered when the text is not pre-determined.

Pluralizing algorithms

The simplest algorithm for generating arbitrary English plurals is simply to add *-s* to each word (*clam* → *clams*, *storey* → *storeys*, *bag* → *bags*, etc.). Of course, this approach fails miserably on many special cases (*class* → *classes*, *story* → *stories*, *box* → *boxes*), and on the hundreds of irregular plural English nouns (*criterion* → *criteria*, *stigma* → *stigmata*, *ox* → *oxen*). Nor does it cater for verbs (*classifies* → *classify*, *stores* → *store*, *bobs* → *bob*) or adjectives (*my* → *our*, *her* → *their*, *Bob's* → *Bobs'*).

More complex algorithms that cope with specific suffixes (*-ss* → *-sses*, *-y* → *-ies*, etc.) can be specified, but pure suffix-based approaches will still be prone to exceptions and meta-exceptions. For example: *-y* becomes *-ies*, except after a vowel (when it becomes *-ys*), except for *soliloquy* (which takes *-ies*).

A usable pluralization algorithm must therefore cope with three categories of plural formation: universal defaults, general suffix-based rules, and specific exceptional cases. The following section examines each of these categories in more detail.

Categories of English plurals

Universal rules

Although described here first, and encountered most frequently, the universal rules of plural inflection are the "last resort" in an algorithmic sense. That is, these rules only apply when all other more specific rules or special cases (see below) are inapplicable.

The rules themselves are well-known and need no elaboration. By default:

- Nouns are made plural by appending *-s*.
- Verbs are made plural by removing any trailing *-s* (and otherwise do not change).
- Adjectives and adverbs do not change when made plural.

Suffix categories

There are, however, an enormous number of exceptions to these defaults [4]. Most such exceptions are still regular (in the sense that they occur in predictable patterns), but are specific to a particular word suffix. For example, nouns that end in *-ss* universally become *-sses* in the plural (and vice versa for verbs). Likewise, nouns which end in a vowel followed by *-y* almost always become *-ies* in the plural.

Certain types of adjectives also inflect in this way. For example, possessive adjectives that end in *-'s* or *-'* in the singular are made plural by forming the plural of the root word and appending an apostrophe (unless the root's plural does not itself end in *-s*, in which case *-'s* is appended). Hence *cat's* becomes *cats'*, *axis'* becomes *axes'*, whilst *child's* becomes *children's*.

Other suffix categories arise because words of foreign origin (most commonly Ancient Greek or Latin) have retained a non-anglicized plural inflection. Hence *criterion* becomes *criteria*, *nucleus* becomes *nuclei*, and *matrix* becomes *matrices*. Dealing with such categories is complicated by the fact that many other imports have been wholly or partially anglicized. Hence although *criterion* always forms its plural with *-a*, *ganglion* may take either *-s* or *-a* (*ganglions* or *ganglia*), whilst *bastion* is always inflected with *-s*. Occasionally the anglicized and "classical" plural forms of a word may both be in common use, but with distinct meanings. Thus a copy-editor might remove *appendices*, whereas a surgeon would remove *appendixes*.

The correct inflection of words derived from Latin can be particularly complex, since the same suffix may form different Latinate plurals depending on the declension (or sometimes the part of speech) of the

Singular suffix	Anglicized plural	Classical plural	Example
-a	(none)	*-ae*	*alga* → *algae*
-a	*-as*	*-ae*	*nova* → *novas/novae*
-a	*-as*	*-ata*	*dogma* → *dogmas/dogmata*
-an	*-en*	(none)	*woman* → *women*
-ch	*-ches*	(none)	*church* → *churches*
-eau	*-eaus*	*-eaux*	*chateau* → *chateaus/chateaux*
-en	*-ens*	*-ina*	*foramen* → *foramens/foramina*
-ex	(none)	*-ices*	*codex* → *codices*
-ex	*-exes*	*-ices*	*index* → *indexes/indices*
-f(e)	*-ves*	(none)	*life* → *lives*
-ieu	*-ieus*	*-ieux*	*milieu* → *mileus/milieux*
-is	(none)	*-es*	*basis* → *bases*
-is	*-ises*	*-ides*	*iris* → *irises /irides*
-ix	*-ixes*	*-ices*	*matrix* → *matrixes/matrices*
-nx	*-nxes*	*-nges*	*phalanx* → *phalanxes /phalanges*
-o	*-oes*	(none)	*potato* → *potatoes*
-o	*-os*	(none)	*photo* → *photos*
-o	(none)	*-i*	*graffito* → *graffiti*
-o	*-os*	*-i*	*tempo* → *tempos/tempi*
-on	(none)	*-a*	*aphelion* → *aphelia*
-on	*-ons*	*-a*	*ganglion* → *ganglions/ganglia*
-oo-	*-ee-*	(none)	*foot* → *feet*
-oof	*-oofs*	*-ooves*	*hoof* → *hoofs/hooves*
-s	*-s*	(none)	*series* → *series*
-s	*-ses*	(none)	*atlas* → *altases*
-sh	*-shes*	(none)	*wish* → *wishes*
-um	(none)	*-a*	*bacterium* → *bacteria*
-um	*-ums*	*-a*	*medium* → *mediums/media*
-us	(none)	*-era*	*genus* → *genera*
-us	(none)	*-i*	*stimulus* → *stimuli*
-us	*-uses*	*-era*	*opus* → *opuses/opera*
-us	*-uses*	*-i*	*radius* → *radiuses/radii*
-us	*-uses*	*-ora*	*corpus* → *corpuses/corpora*
-us	*-uses*	*-us*	*status* → *statuses/status*
-x	*-xes*	(none)	*box* → *boxes*
-y	*-ies*	(none)	*ferry* → *ferries*

Table 1: Major English suffix categories.

original. Thus the plural of *stimulus* (second declension) is *stimuli*, and that of *genus* (third declension) is *genera*. *Status* (fourth declension) is traditionally unchanged in the plural, whilst *ignoramus* (a first person plural Latin verb) has been wholly anglicized and becomes *ignoramuses*.

The only practical way to deal with such complexities in an algorithm is to categorize words by both suffix .and inflection, and to allow for both anglicized and classical variants. Table 1 illustrates such categories

General and user-defined exceptions

Some categories of words contain only a single example, and are more appropriately treated as exceptions to more general rules. Table 2 lists the main offenders.

Singular form	Anglicized plural	Classical plural
beef	*beefs*	*beeves*
brother	*brothers*	*brethren*
child	(none)	*children*
cow	*cows*	*kine*
ephemeris	(none)	*ephemerides*
genie	*genies*	*genii*
money	*moneys*	*monies*
mongoose	*mongooses*	(none)
mythos	(none)	*mythoi*
octopus	*octopuses*	*octopodes*
ox	(none)	*oxen*
soliloquy	*soliloquies*	(none)
trilby	*trilbys*	(none)

Table 2: Irregular English plurals

This table is surprisingly comprehensive, though certainly not exhaustive. Indeed, specific dialects of English may define much larger sets of irregular plurals and may not recognize some of the entries in Table 2. Hence it is important that any algorithmic approach to pluralization be both extensible and adjustable, so that its output may be easily expanded or trimmed for a specific audience.

A pluralizing algorithm for English

This section first presents algorithms for forming plurals of English nouns, verbs, and adjectives. It then describes how these three algorithms may be merged into a single inflection procedure that is applicable to any part of speech. Finally, the limitations of this unified algorithm are discussed.

The algorithms are based on the rules of English inflection described in the *Oxford English Dictionary* [5] (OED), Fowler's *Modern English Usage* [6], and *A Practical English Grammar* [1]. Where these sources disagree, the OED is taken to be definitive.

A note about user-defined inflections

All four algorithms presented below allow for user-defined inflections that override the normal rules of English plural formation. Such user-defined inflections might be specified as an ordered table of *<singular form>* → *<plural form>* pairs (much like the various enumerated tables for irregular plurals listed in Appendix A). For example:

VAX → *VAXen*

To extend the power of this mechanism, each singular form can be specified as a (case-insensitive) regular expression, rather than a literal word to be matched. This allows the user to specify families of common inflections. For example, one might specify that all nouns ending in *-x* will be inflected to *-xen* (*oxen*, *boxen*, *suffixen*, etc.), regardless of the normal rules of English:

(.)x* → *$1xen*

Furthermore, if the user-defined table preserves a suitable ordering (perhaps "first-defined, last-tried"), then exceptions to such user-defined generic rules can also be specified. For example:

(.)x* → *$1xen*
fox → *foxes*

As a final generalization, the plural form allows two variants (an anglicized plural and a "classical" alternative), separated by some delimiter - say "|". In such cases, the plural selected would depend on whether classical or anglicized plurals had been requested. For example, the previous generic rule might be rewritten to cater for "classical" usages:

(.)x* → *$1xes | $1xen*
fox → *foxes*
ox → *oxen*

Note that, where only one plural form is specified, it is used in both "anglicized" and "classical" modes.

Nomenclature

In the algorithmic descriptions below, the following constructs are used:

suffix(<*suffix*>)

This predicate returns true if the word being inflected ends in **<*suffix*>**. Note that standard regular expression conventions are used after the "-" that introduces the suffix.

category(*<singular>*,*<plural>*)

This predicate returns true if the word being inflected belongs to the set of English words whose suffixes inflect from ***<singular>*** to ***<plural>*** when pluralized.

inflection(*<singular>*,*<plural>*)

This function returns the word being inflected, after replacing its current suffix (which must be ***<singular>***) with the suffix ***<plural>***.

stem(*<suffix>*)

This function removes the specified suffix (***<suffix>***) from the word being inflected and returns the remaining stem. If the word does not originally end in the specified suffix, a special "undefined" value is returned.

"the (user-)specified plural form"

This phrase is used whenever a word has been found to belong to an enumerated category. The "specified plural form" is the appropriate anglicized or classical plural form of the word, as it appears in the category table.

Algorithms for forming plural nouns, verbs and adjectives

Algorithm 1 takes the singular form of an English noun and returns its plural.

Algorithm 2 takes the singular form of a conjugated English verb and returns its plural form. English verb inflections are more regular than noun inflections and hence the verb inflection algorithm is considerably simpler.

Algorithm 3 takes the singular form of an English adjective (or article or genitive pronoun) and returns its plural form. Note that only a very few English adjectives inflect with number.

A unified algorithm

Having specified an algorithm for each particular part of speech, it is a relatively simple matter to combine them and construct a single algorithm that correctly handles any of these parts of speech (but see "Issues and Limitations" below). The general approach taken here is to treat a word being pluralized as if it were a noun, unless it can be unambiguously recognized as a verb or adjective. Hence the unified pluralization algorithm (Algorithm 4) first honours any user-defined inflections, then seeks to apply a subset of the steps from the verb- and adjective-specific algorithms presented above and, if they fail, finally applies the entire noun-specific algorithm to the word. Note that, since the complete noun algorithm handles all words, the untried steps of the verb and adjective algorithms will never need to be invoked.

Note that algorithm presented represents a particular compromise in the face of inherently ambiguous input. Other compromises (which might perhaps more heavily favour the verb sense of a word) may also be defined, by selecting different subsets of the three algorithms or by changing the order in which the subsets are used.

Issues and limitations

Homographs of heterogeneous case

The singular pronoun *it* presents a special problem because its plural form can vary, depending on its grammatical case. For example:

It ate it → *They ate them*

As a consequence of this ambiguity, the noun and unified algorithms cannot guarantee to inflect *it* correctly without additional context. This could be provided by an extra parameter (one which specifies the required case), or by simply defaulting to the nominative (*it* → *they*) and accepting a small number of incorrect inflections.

Of course, where the necessary context is already provided (for example, when forming the plural of a dative or ablative: *to it*, *from it*, *with it*, etc.), the noun algorithm detects this (in step 3) and correctly returns the accusative plural form: *to them*, *from them*, *with them*, etc.)

Homographs of heterogeneous person

In the conjugation of most English verbs, the 1st and 2nd person singular forms are identical (*I eat*, *you eat*; *I see*, *you see*), as are the corresponding plural forms (*we eat*, *you eat*; *we see*, *you see*).

However, if a verb were to take common singular forms but different plurals (for example, the atrophying British usage: *I will* → *you shall, you will* → *you will*), then the algorithms presented above would be unable to determine the correct inflection without additional context (such as an extra "person" parameter).

The author is not currently aware of any other verbs in English which present this problem, but is not willing to assume *ipso facto* that none exist.

Other homographs with heterogeneous plurals

One context in which intent (rather than content) sometimes determines plurality, is where two distinct meanings of a word require different plurals. For example:

I put the mice next to the cheese.
I put the mouses next to the keyboards.

Three basses were stolen from the band's trailer.
Three bass were stolen from the band's fishpond.

Several had thoughts of leaving.
Several had thought of leaving.

1. Check if the user has defined an inflection for the noun, and , if so, accept that...

```
if the word matches a user-defined noun,
      return the user-specified plural form
```

2. Handle words that do not inflect in the plural (such as *fish*, *travois*, *chassis*, nationalities ending in *-ese* etc. - see Tables A.2 and A.3)...

```
if suffix(-fish) or suffix(-ois) or suffix(-sheep)
or suffix(-deer) or suffix(-pox) or suffix(-[A-Z].*ese)
or category(-,-),
      return the original noun
```

3. Handle pronouns in the nominative, accusative, and dative (see Tables A.5), as well as prepositional phrases...

```
if the word is a pronoun,
      return the specified plural of the pronoun
if the word is of the form: "<preposition> <pronoun>",
      return "<preposition> <specified plural of pronoun>"
```

4. Handle standard irregular plurals (*mongooses*, *oxen*, etc. - see table A.1)...

```
if the word has an irregular plural,
      return the specified plural
```

5. Handle irregular inflections for common suffixes (*synopses*, *mice* and *men*, etc.)...

```
if suffix(-man),       return inflection(-man,-men)
if suffix(-[lm]ouse),  return inflection(-ouse,-ice)
if suffix(-tooth),     return inflection(-tooth,-teeth)
if suffix(-goose),     return inflection(-goose,-geese)
if suffix(-foot),      return inflection(-foot,-feet)
if suffix(-zoon),      return inflection(-zoon,-zoa)
if suffix(-[csx]is),   return inflection(-is,-es)
```

6. Handle fully assimilated classical inflections (*vertebrae*, *codices*, etc. - see tables A.10, A.14, A.19 and A.20, and tables A.11, A.15 and A.21 if in "classical mode)...

```
if category(-ex,-ices), return inflection(-ex,-ices)
if category(-um,-a),    return inflection(-um,-a)
if category(-on,-a),    return inflection(-on,-a)
if category(-a,-ae),    return inflection(-a,-ae)
```

7. Handle classical variants of modern inflections (*stigmata*, *soprani*, etc. - see tables A.11 to A.13, A.15, A.16, A.18, A.21 to A.25)...

```
if in classical mode,
      if suffix(-trix),       return inflection(-trix,-trices)
      if suffix(-eau),        return inflection(-eau,-eaux)
      if suffix(-ieu),        return inflection(-ieu,-ieux)
      if suffix(-..[iay]nx),  return inflection(-nx,-nges)
      if category(-en,-ina),  return inflection(-en,-ina)
      if category(-a,-ata),   return inflection(-a,-ata)
      if category(-is,-ides), return inflection(-is,-ides)
      if category(-us,-i),    return inflection(-us,-i)
      if category(-us,-us),   return the original noun
      if category(-o,-i),     return inflection(-o,-i)
      if category(-,-i),      return inflection(-,-i)
      if category(-,-im),     return inflection(-,-im)
```

8. The suffixes *-ch*, *-sh*, and *-ss* all take *-es* in the plural (*churches*, *classes*, etc)...

```
if suffix(-[cs]h), return inflection(-h,-hes)
if suffix(-ss),    return inflection(-ss,-sses)
```

9. Certain words ending in *-f* or *-fe* take *-ves* in the plural (*lives*, *wolves*, etc)...

```
if suffix(-[aeo]lf) or suffix(-[^d]eaf) or suffix(-arf),
      return inflection(-f,-ves)
if suffix(-[nlw]ife),
      return inflection(-fe,-ves)
```

10. Words ending in *-y* take *-ys* if preceded by a vowel (*storeys*, *stays*, etc.) or when a proper noun (*Marys*, *Tonys*, etc.), but *-ies* if preceded by a consonant (*stories*, *skies*, etc.)...

```
if suffix(-[aeiou]y), return inflection(-y,-ys)
if suffix(-[A-Z].*y), return inflection(-y,-ys)
if suffix(-y),        return inflection(-y,-ies)
```

11. Some words ending in *-o* take *-os* (*lassos*, *solos*, etc. - see tables A.17 and A.18); the rest take *-oes* (*potatoes*, *dominoes*, etc.) However, words in which the *-o* is preceded by a vowel always take *-os* (*folios*, *bamboos*)...

```
if category(-o,-os) or suffix(-[aeiou]o), return inflection(-o,-os)
if suffix(-o), return inflection(-o,-oes)
```

12. Handle plurals of compound words (*Postmasters General*, *Major Generals*, *mothers-in-law*, etc) by recursively applying the entire algorithm to the underlying noun. See Table A.26 for the military suffix *-general*, which inflects to *-generals*...

```
if category(-general,-generals), return inflection(-l,-ls)
if the word is of the form: "<word> general",
      return "<plural of word> general"
if the word is of the form: "<word> <preposition> <words>",
      return "<plural of word> <preposition> <words>"
```

13. Otherwise, assume that the plural just adds *-s* (*cats*, *programmes*, *trees*, etc.)...

```
otherwise, return inflection(-,-s)
```

Algorithm 1: Plural inflection of nouns

1. Check if the user has defined an inflection for the verb, and , if so, accept that...

```
if the word matches a user-defined verb,
      return the user-specified plural form
```

2. Check if the verb is being used as an auxiliary and has a known irregular inflection (*has seen*, *was going*, etc. See Table A.8 for irregular verbs)...

```
if the word has the form "<auxiliary> <words>"
and <auxiliary> belongs to the category of irregular verbs,
      return "<specified plural of auxiliary> <words>"
```

3. Handle simple irregular verbs (*has*, *is*, etc. - see Table A.8)...

```
if the word belongs to the category of irregular verbs,
      return the specified plural form
```

4. Verbs in the regular 3rd person singular lose their *-es*, *-ies*, or *-oes* suffix (*she catches* → *they catch*, *he tries* → *they try*, *it does* → *they do*, etc.)...

```
if suffix(-[cs]hes), return inflection(-hes,-h)
if suffix(-[sx]es),  return inflection(-es,-)
if suffix(-zzes),    return inflection(-es,-)
if suffix(-ies),     return inflection(-ies,-y)
if suffix(-oes),     return inflection(-oes,-o)
```

5. Other 3rd person singular verbs ending in *-s* (but not *-ss*) also lose their suffix...

```
if suffix(-[^s]s),   return inflection(-s,-)
```

6. Handle ambiguous simple verbs that might also be nouns (*thought*, *sink*, *fly*, etc. - see Table A.4)...

```
if the word is in the ambiguous category,
      return the specified plural form
```

7. All other cases are regular 1st or 2nd person verbs, which don't inflect...

```
otherwise, return the word uninflected
```

Algorithm 2: Plural inflection of verbs

1. Check if the user has defined an inflection for the adjective, and, if so, accept that...
```
if the word matches a user-defined adjective,
      return the user-specified plural form
```
2. Handle indefinite articles and demonstratives...
```
if the word is "a" or "an", return "some"
if the word is "this",      return "these"
if the word is "that",      return "those"
```
3. Handle possessive pronouns (*my* → *our*, *its* → *their*, etc - see Table A.7)...
```
if the word is a personal possessive,
      return the specified plural form
```
4. Handle genitives (*dog's* → *dogs'*, *child's* → *children's*, *Mary's* → *Marys'*, etc). The general rule is: remove the apostrophe and any trailing *-s*, form the plural of the resultant noun, and then append an apostrophe (or *-'s* if the pluralized noun doesn't end in *-s*)...
```
if suffix(-'s) or suffix(-'),
      if suffix(-'), let the noun <owner> be inflection(-',-)
      otherwise,     let the noun <owner> be inflection(-'s,-)
      let the noun <owners> be the noun plural of <owner>
      if <owners> ends in -s, return "<owners>'"
      otherwise,              return "<owners>'s"
```
5. In all other cases no inflection is required...
```
otherwise, return the word uninflected
```

Algorithm 3: Plural inflection of adjectives

1. Handle user-defined cases...
```
try step 1 of Algorithm 3
try step 1 of Algorithm 2
try step 1 of Algorithm 1
```
2. Handle known adjectives...
```
try steps 2 through 4 of Algorithm 3
```
3. Handle known verbs...
```
try steps 2 through 5 of Algorithm 2
```
4. Handle singular nouns ending in *-s* (*ethos*, *axis*, etc. - see Tables A.2, A.3, A.16, A.22, and A.23)...
```
if word is a noun ending in -s,
      try steps 2 through 13 of Algorithm 1
```
5. Handle 3rd person singular verbs (that is, any other words ending in *-s*)...
```
try steps 4 and 5 of Algorithm 2
```
6. Treat the word as a noun...
```
try steps 2 through 13 of Algorithm 1
```

Algorithm 4: Unified plural inflection of nouns, verbs, and adjectives

The algorithms presented above handle such words in two ways:

- If both meanings of the word are the same part of speech (for example, *bass* is a noun in both sentences above), then one meaning is chosen as the "usual" meaning, and only that meaning's plural is ever returned by any of the inflection subroutines.
- If each meaning of the word is a different part of speech (for example, *thought* is used as both a noun and a verb), then the noun's plural is returned by the noun and unified algorithms and the verb's plural is returned only by the verb algorithm.

Such contexts are (fortunately) uncommon, particularly examples involving two senses of a noun. An informal study of nearly 600 "difficult" plurals indicates that the unified algorithm can be relied upon to choose appropriately in about 98% of cases (although, of course, ichthyophilic guitarists may experience higher rates of confusion).

Finally, if the choice of a particular "usual inflection" is inappropriate for a particular application, it can always be changed by specifying an overriding user-defined inflection.

"Number-insensitive" comparisons

The need for "number-insensitive" comparisons

Another task which is complicated by the irregular inflections of many English plurals is that of indexing or cross-referencing text. Consider the following extracts from Ambrose Bierce's estimable dictionary [7]:

> *Child*
> *An accident to the occurrence of which all the forces and arrangements of nature are specially devised and accurately adapted.*
>
> *Genius*
> *Any degree of mental superiority that enables its possessor to live acceptably upon his admirers, and without blame be unbrokenly drunk.*
>
> *Self*
> *The most important person in the universe.*

Any reliable indexing algorithm for such terms will need to be able to identify text containing the various irregular plural forms of these words. Furthermore, since a small number of Bierce's definitions are for plural terms (*aborigines*, *footprints*, *kine*, *relations*, etc.), cross-referencing the collection requires checks in both directions (singular text to plural term, and plural text to singular term). Worse still, the need to cross-reference terms like *kine* (to the words *cow* and *cows*) means that words which are alternate plural forms of a common singular must also be identified.

An algorithm

This section presents an algorithm for equality test between two words. The algorithm returns which returns true if:

- the two words are identical, or
- one word is a plural form of the other, or
- the two words are distinct plural forms of some other word.

It should be noted, however, that two distinct singular words which happen to take the same plural form are not considered equal, nor are cases where one (singular) word's plural is the other (plural) word's singular. Hence *base* is not "number-insensitively" equal to *basis*, even though they both have the plural form *bases*. Likewise, *opus* does not compare equal to *operas* even though *opus* has the plural *opera* and *opera* has the plural *operas*.

Note that, because steps 2 to 3 do not specify which pluralizing algorithm is used, Algorithm 5 is generic and may be readily adapted to deal with only nouns, verbs, or adjectives, or with all three at once. Such adaptations merely involve selecting the appropriate algorithm (Algorithms 1 through 4 respectively) with which to generate the "appropriate plural" forms. Where the algorithm is adapted to a particular part of speech, one or both of steps 4 and 5 may be omitted entirely, if inappropriate.

A Perl implementation

This section briefly summarizes a freely available Perl implementation of the pluralization algorithms presented above (`Lingua::EN::Inflect`). The module and full supporting documentation are available from the Comprehensive Perl Archive Network (via `http://www.perl.com`), or directly from the author:

`http://www.csse.monash.edu.au/~damian/CPAN/Lingua-EN-Inflect.gz.tar`

The various subroutines of `Lingua::EN::Inflect` provide plural inflections for English words. Plural forms of most nouns, many verbs, and some adjectives are provided. Where appropriate, "classical" variants are also provided. The module also offers pronunciation-based selection of indefinite articles (*a* and *an*), but discussion of those facilities is beyond the scope of this paper.

Inflecting plurals - the `PL_...()` subroutines

`Lingua::EN::Inflect` provides four exportable subroutines (prefixed **`PL_...`**) which implement the noun-, verb-, adjective-, and unified pluralization algorithms described above. All of the **`PL_...()`** subroutines take the word to be inflected as their first

1. Check for simple equality...

```
if <word1> equals <word2>, return true
```

2. Check for number disparity using standard inflection...

```
using anglicized plurals...
      if the appropriate plural of <word1> equals <word2>,
             return true
      if the appropriate plural of <word2> equals <word1>,
             return true
```

3. Check for number disparity using "classical" inflection...

```
using classical plurals...
      if the appropriate plural of <word1> equals <word2>,
             return true
      if the appropriate plural of <word2> equals <word1>,
             return true
```

4. Handle two variant plurals for the same noun (*brothers* and *brethren*, for example) by checking if there exists a category **<c>** and a word **<w>**, such that **<word1>** and **<word2>** end in the distinct plural suffixes of category **<c>**, and word **<w>** can inflect to both **<word1>** and **<word2>**...

```
if the words are nouns,
      for each noun category <c>...
             let <ss> be the singular suffix for category <c>
             let <sa> be the anglicized plural suffix for <c>
             let <sc> be the classical plural suffix for <c>
             if <sa> differs from <sc>,
                    let <stem1> be stem(<sa>) of <word1>
                    if <word2> equals inflect(-,<sc>) of <stem1>,
                           return true
                    let <stem2> be stem(<sa>) of <word2>
                    if <word1> equals inflect(-,<sc>) of <stem2>,
                           return true
```

5. Handle distinct plural genitives (*cows'* and *kine's*, for example) by removing any *-'s*, *-s'*, or *-'* inflection and comparing the underlying nouns...

```
if the words are adjectives,
      let <word1a> be stem(-'s) or stem(-') of <word1>
      let <word2a> be stem(-'s) or stem(-') of <word2>
      let <word1b> be stem(-s') of <word1>
      let <word2b> be stem(-s') of <word2>
      for each defined <w1> in (<word1a>, <word1b>)...
             for each defined <w2> in (<word2a>, <word2b>)...
                    apply step 4 to <w1> and <w2>
                    if step 4 returns true,
                           return true
```

6. All other cases corresponding to an equality...

```
otherwise, return false
```

Algorithm 5: "Number-insensitive" comparison

argument and return the corresponding inflection. Note that all such subroutines expect the singular form of the word. The results of passing a plural form are undefined (and unlikely to be meaningful).

The **`PL_...()`** subroutines also take an optional second argument, which indicates the desired grammatical number of the word. If the "number" argument is supplied and is not **`1`** (or **`"one"`** or **`"a"`**, or some other adjective that implies the singular), the plural form of the word is returned. If the "number" argument does indicate singularity, the (uninflected) word itself is returned. If the number argument is omitted, the plural form is returned unconditionally.

The various subroutines are:

`PL_N($;$)`

> **`PL_N()`** takes a singular English noun or pronoun and returns its plural.

`PL_V($;$)`

> **`PL_V()`** takes the singular form of a conjugated verb (one which is already in the correct grammatical person and mood) and returns the corresponding plural conjugation.

`PL_ADJ($;$)`

> **`PL_ADJ()`** takes the singular form of certain types of adjectives and returns the corresponding plural form.

`PL($;$)`

> **`PL()`** takes a singular English noun, pronoun, verb, or adjective and returns its plural form. Where a word has more than one inflection depending on its part of speech, the (singular) noun sense is generally preferred to the (singular) verb sense. Of course, the inherent ambiguity of such cases suggests that, where the part of speech is known, **`PL_N()`**, **`PL_V()`**, and **`PL_ADJ()`** should be used in preference to **`PL()`**.

Note that all these subroutines ignore any whitespace surrounding the word being inflected, but preserve that whitespace when the result is returned. For example, **`PL(" cat ")`** returns the string **`" cats "`**.

Modern vs classical inflections

`Lingua::EN::Inflect` can differentiate between modern and classical plural variants via the exportable subroutine **`classical()`**. If **`classical()`** is called with no arguments, it unconditionally invokes classical mode. If it is called with an argument, it invokes classical mode only if that argument evaluates to true. If the argument is false, classical mode is switched off.

In classical mode, the non-anglicized plural form of a word (if one exists) is preferred. Hence, whereas *dogma* is normally inflected to *dogmas*, if classical mode is active it becomes *dogmata*.

User-defined inflections - the `def_...()` subroutines

`Lingua::EN::Inflect` provides three exportable subroutines which allow the programmer to override the module's pluralizing behaviour for specific cases:

`def_noun($$)`

The **`def_noun()`** subroutine takes a pair of string arguments: the singular and plural forms of the noun being specified. The singular form specifies a pattern to be interpolated (as **`m/^(?:$first_arg)$/i`**). Any noun matching this pattern is then replaced by the string in the second argument. The second argument specifies a string which is interpolated after the match succeeds, and is then used as the plural form. The second argument string may also specify a second variant of the plural form, to be used when "classical" plurals have been requested. The beginning of the second variant is marked by a '**`|`**' character:

```
def_noun  'cow'      =>  'cows|kine';
def_noun  '(.+i)o'   =>  '$1os|$1i';
```

If no classical variant is given, the same plural form is used in both normal and "classical" modes. If the second argument is **`undef`** instead of a string, then the current user definition for the first argument is removed, and the standard (algorithmic) plural inflection is reinstated.

`def_verb($$$$$$)`

The **`def_verb()`** subroutine takes three pairs of string arguments (that is, six arguments in total), specifying the singular and plural forms of the three grammatical persons of verb. As with **`def_noun()`**, the singular forms are specifications of run-time-interpolated patterns, while the plural forms are specifications of (up to two) run-time-interpolated strings:

```
def_verb 'am'       => 'are',
         'ar(e|t)'  => 'are",
         'is'       => 'are';
```

`def_adj($$)`

The **`def_adj()`** subroutine takes a pair of string arguments, which specify the singular and plural forms of the adjective being defined. As with **`def_noun()`** and **`def_verb()`**, the singular forms are specifications of run-time-interpolated patterns, whilst the plural forms are specifications of (up to two) run-time-interpolated strings:

```
def_adj  'dat' => 'dose';
def_adj  'red' => 'red|gules';
```

Numbered plurals - the `NO()` subroutine

The **`PL_...()`** subroutines only return the inflected word, not the count that was used to decide its inflection. Thus, in order to output the plural form *I saw 3 ducks*, it is necessary to use:

```
print "I saw $N ", PL_N($what,$N), "\n";
```

Since the usual purpose of producing a plural is to make it agree with an explicit preceding count, `Lingua::EN::Inflect` provides an exportable subroutine (**`NO($;$)`**) which, given a word and an optional count, returns the count followed by the correctly inflected word. Hence the previous example can be rewritten:

```
print "I saw ", NO($what,$N), "\n";
```

In addition, if the count is zero (or some other expression which implies zero, such as **`"zero"`**, **`"nil"`**, etc.), the count is replaced by the string **`"no"`**. Hence if **`$N`** had the value zero the previous example would print the somewhat more elegant:

I saw no ducks

rather than:

I saw 0 ducks

Reducing the number of counts required - the `NUM()` subroutine

In some contexts, the need to supply an explicit count to the various **`PL_...()`** subroutines makes for tiresome repetition. For example:

```
print PL_ADJ("This",$errors),
      PL_N(" error",$errors),
      PL_V(" was",$errors), " fatal.\n";
```

`Lingua::EN::Inflect` therefore provides an exportable subroutine (**`NUM($;$)`**) which may be used to set a persistent "default number" value. If such a value is set, it is subsequently used whenever an optional second "number" argument of a **`PL_...()`** subroutine is omitted. The default value thus set can subsequently be removed by calling **`NUM()`** with no arguments:

```
NUM($errors);     # SET DEFAULT NUMBER
print PL_ADJ("This"), PL_N(" error"),
      PL_V(" was"), "fatal.\n";
NUM();            # CLEAR DEFAULT NUMBER
```

By default, **`NUM()`** returns its first argument, so that it may also be "inlined" in contexts like:

```
print NUM($errors), PL_N(" error"),
      PL_V(" was"), " detected.\n";
print PL_ADJ("This"), PL_N(" error"),
      PL_V(" was"), "fatal.\n"
            if $severity > 1;
```

Interpolating inflections in strings - The `inflect()` subroutine

By far the commonest use of the inflection subroutines is to produce message strings for various purposes. Unfortunately, as the above examples demonstrate, the need to separate each **`PL_...()`** subroutine call often detracts from the readability of the resulting code.

To ameliorate this problem, `Lingua::EN::Inflect` provides an exportable string-interpolating subroutine (**`inflect($)`**), which recognizes calls to the various inflection subroutines within a string and interpolates them appropriately. Using **`inflect()`** plurals can be interpolated directly into a string as follows:

```
NUM($errors);
print inflect
      "NO(error) PL_V(was) detected\n";
print inflect
      "The PL_N(error) PL_V(was)" fatal\n"
            if $severity > 1;
```

Comparing "number-insensitively" - The `PL_..._eq()` subroutines

`Lingua::EN::Inflect` also implements the number-insensitive comparison algorithm described above, providing the exportable subroutines **`PL_eq($$)`**, **`PL_N_eq($$)`**, **`PL_V_eq($$)`**, and **`PL_ADJ_eq($$)`**. Each of these subroutines takes two strings, and compares them using the corresponding plural-inflection subroutine (**`PL()`**, **`PL_N()`**, **`PL_V()`**, and **`PL_ADJ()`** respectively).

The actual value returned by the various **`PL_..._eq()`** subroutines encodes which of the three equality rules succeeded: **`"eq"`** is returned if the strings were identical, **`"s:p"`** if the strings were singular and plural respectively, **`"p:s"`** for plural and singular, and **`"p:p"`** for two distinct plurals. Inequality is indicated by returning an empty string.

Conclusion

Capturing the English plural inflection in reliable algorithms proves to be a feasible, if challenging, task. The robustness of such algorithms depends heavily on encoding general rules (categories of inflection), rather than attempting to enumerate many hundreds of exceptions to the universal defaults.

It is possible to cater for differences in major usage patterns (for example, modern and classical inflections) and for local differences in dialect (via user-defined inflections). It is also possible to make use of the pluralization algorithms to efficiently detect pairs of words which differ only in grammatical number.

A free implementation of these algorithms is available, and provides additional features such as conditional pluralization (depending on a numerical parameter), setting of default number values, and interpolation of the various subroutines into strings.

References

[1] Wall, L., Christiansen, T., & Schwartz, R.L., *Programming Perl, 2nd Edition*, O'Reilly & Associates, 1996.

[2] McCrum, R., Cran, W., & MacNeil, R., *The Story of English*, Penguin Books, New York, 1986.

[3] Bryson, B., *The Mother Tongue: English and how it got that way*, William Morrow, New York, 1990.

[4] Thomson, A.J., & Martinet, A.V., *A Practical English Grammar, Fourth Edition*, Oxford University Press, Oxford, 1986.

[5] *The Oxford English Dictionary, Second Edition,* Oxford University Press, Oxford, 1989.

[6] Fowler, H.W., *Modern English Usage, Second Edition*, Oxford University Press, Oxford, 1965.

[7] Bierce, A. *The Devil's Dictionary*, Doubleday, New York, 1911.

Appendix A - Plural categories

Table A.1: Irregular nouns

Singular form	Anglicized plural	Classical plural
beef	*beefs*	*beeves*
brother	*brothers*	*brethren*
child	(none)	*children*
cow	*cows*	*kine*
ephemeris	(none)	*ephemerides*
genie	*genies*	*genii*
money	*moneys*	*monies*
mongoose	*mongooses*	(none)
mythos	(none)	*mythoi*
octopus	*octopuses*	*octopodes*
ox	(none)	*oxen*
soliloquy	*soliloquies*	(none)
trilby	*trilbys*	(none)

Table A.2: Uninflected nouns

bison	*flounder*	*pliers*
bream	*gallows*	*proceedings*
breeches	*graffiti*	*rabies*
britches	*headquarters*	*salmon*
carp	*herpes*	*scissors*
chassis	*hijinks*	*sea-bass*
clippers	*homework*	*series*
cod	*innings*	*shears*
contretemps	*jackanapes*	*species*
corps	*mackerel*	*swine*
debris	*measles*	*trout*
diabetes	*mews*	*tuna*
djinn	*mumps*	*whiting*
eland	*news*	*wildebeest*
elk	*pincers*	

Table A.3: Singular nouns ending in *-s*

acropolis	*chaos*	*lens*
aegis	*cosmos*	*mantis*
alias	*dais*	*marquis*
arthritis	*digitalis*	*metropolis*
asbestos	*encephalitis*	*neuritis*
atlas	*epidermis*	*pathos*
bathos	*ethos*	*pelvis*
bias	*gas*	*polis*
bronchitis	*glottis*	*rhinoceros*
caddis	*hepatitis*	*sassafras*
cannabis	*hubris*	*tonsillitis*
canvas	*ibis*	*trellis*

Table A.4: Ambiguous words (nouns or verbs)

act	*fight*	*run*
bend	*fire*	*saw*
bent	*like*	*sink*
blame	*look*	*sleep*
copy	*make*	*thought*
cut	*might*	*view*
drink	*reach*	*will*

Table A.5: Personal pronouns (nominative, accusative, and reflexive)

1st Person	2nd Person	3rd Person
I → *we*	*you* → *you* *thou* → *you*	*she* → *they* *he* → *they* *it* → *they* *they* → *they*
me → *us*	*you* → *you* *thee* → *you*	*her* → *them* *him* → *them* *it* → *them* *them* → *them*
myself → *ourselves*	*yourself* → *yourself* *thyself* → *yourself*	*herself* → *themselves* *himself* → *themselves* *itself* → *themselves* *themself* → *themselves* *oneself* → *oneselves*

Table A.6: Possessive pronouns

1st Person	2nd Person	3rd Person
mine → *ours*	*yours* → *yours* *thine* → *yours*	*hers* → *theirs* *his* → *theirs* *its* → *theirs* *theirs* → *theirs*

Table A.7: Personal possessive adjectives

1st Person	2nd Person	3rd Person
my → *our*	*your* → *your* *thy* → *your*	*her* → *their* *his* → *their* *its* → *their* *their* → *their*

Table A.8: Common irregular verbs

1st Person	2nd Person	3rd Person
am → *are*	*are* → *are*	*is* → *are*
was → *were*	*were* → *were*	*was* → *were*
have → *have*	*have* → *have*	*has* → *have*

Table A.9: Uninflected verbs

ate	*had*	*sank*
could	*made*	*shall*
did	*must*	*should*
fought	*ought*	*sought*
gave	*put*	*spent*

Table A.10: *-a* to *-ae*

alumna	*alga*	*vertebra*

Table A.11: *-a* to *-as* (anglicized) or *-ae* (classical)

abscissa	*formula*	*medusa*
amoeba	*hydra*	*nebula*
antenna	*hyperbola*	*nova*
aurora	*lacuna*	*parabola*

Table A.12: *-a* to *-as* (anglicized) or *-ata* (classical)

anathema	*enema*	*oedema*
bema	*enigma*	*sarcoma*
carcinoma	*gumma*	*schema*
charisma	*lemma*	*soma*
diploma	*lymphoma*	*stigma*
dogma	*magma*	*stoma*
drama	*melisma*	*trauma*
edema	*miasma*	

Table A.13: *-en* to *-ens* (anglicized) or *-ina* (classical)

stamen	*foramen*	*lumen*

Table A.14: *-ex* to *-ices*

codex	*murex*	*silex*

Table A.15: *-ex* to *-exes* (anglicized) or *-ices* (classical)

apex	*latex*	*vertex*
cortex	*pontifex*	*vortex*
index	*simplex*	

Table A.16: *-is* to *-ises* (anglicized) or *-ides* (classical)

iris	*clitoris*	

Table A.17: *-o* to *-os*

albino	*fiasco*	*manifesto*
archipelago	*ghetto*	*medico*
armadillo	*guano*	*octavo*
canto	*inferno*	*photo*
commando	*jumbo*	*pro*
crescendo	*lingo*	*quarto*
ditto	*lumbago*	*rhino*
dynamo	*magneto*	*stylo*
embryo		

Table A.18: *-o* to *-os* (anglicized) or *-i* (classical)

alto	*contralto*	*soprano*
basso	*solo*	*tempo*

Table A.19: *-on* to *-a*

aphelion	*hyperbaton*	*perihelion*
asyndeton	*noumenon*	*phenomenon*
criterion	*organon*	*prolegomenon*

Table A.20: *-um* to *-a*

agendum	*datum*	*extremum*
bacterium	*desideratum*	*stratum*
candelabrum	*erratum*	*ovum*

Table A.21: *-um* to *-ums* (anglicized) or *-a* (classical)

aquarium	*interregnum*	*quantum*
compendium	*lustrum*	*rostrum*
consortium	*maximum*	*spectrum*
cranium	*medium*	*speculum*
curriculum	*memorandum*	*stadium*
dictum	*millenium*	*trapezium*
emporium	*minimum*	*ultimatum*
enconium	*momentum*	*vacuum*
gymnasium	*optimum*	*velum*
honorarium	*phylum*	

Table A.22: *-us* to *-uses* (anglicized) or *-i* (classical)

focus	*nimbus*	*succubus*
fungus	*nucleolus*	*torus*
genius	*radius*	*umbilicus*
incubus	*stylus*	*uterus*

Table A.23: *-us* to *-uses* (anglicized) or *-us* (classical)

apparatus	*impetus*	*prospectus*
cantus	*nexus*	*sinus*
coitus	*plexus*	*status*
hiatus		

Table A.24: *-* to *-i*

afreet	*afrit*	*efreet*

Table A.25: *-* to *-im*

cherub	*goy*	*seraph*

Table A.26: *-general* to *-generals*

Adjutant	*Lieutenant*	*Quartermaster*
Brigadier	*Major*	