A quick reminder about noun plurals in English

1. Fully regular plurals
The default rule for making the plural of an English noun is to add an "s" to the end of its singular.
example: dog --> dogs

2. Semi-regular plurals (euphonic rules) ["euphonic" = easy to hear/pronounce]

Rule 1: If the singular noun ends in "s", "x", "z", "ch" or "sh", add an "es" instead of "s"

examples:
guess --> guesses
box --> boxes
buzz --> buzzes
catch --> catches
dish --> dishes

with the following exception: if the "ch" ending is pronounced "k", add "s" instead of "es"
example: stomach --> stomachs
as well as some fully irregular plurals.
example: ox --> oxen

Rule 2: If the word ends in a consonant followed by "y", change the "y" to "ies"

example: baby --> babies

Note that the presence of a consonant before the "y" is important.
counter-examples: boy --> boys, buy --> buys

3. Irregular plurals
* Collective (or non-count) nouns have no plural form
example: hair --> hair, mud --> mud
... but this rule maybe tricky to apply correctly!
example: His hair is mainly black, but as I saw on his head at least one hair that is grey, there are probably more grey hairs there.

* Invariant (or invariable) nouns do not change when inflected to the plural
example: Deer have antlers that grow, fall off and re-grow annually.

* Other irregular plurals
  - For most nouns that end in "f" or "fe", change the ending to "ves"
  examples: knife --> knives, wife --> wives, half --> halves
  counter-examples: if --> ifs
  - For many words that end in "is", change the "is" to "es"
  examples: hypothesis --> hypotheses, crisis --> crises
  counter-examples: vis (energy) --> vires, vis (a Burmese unit of measure for weight) --> visses
  - For some nouns that end in "o", add "es" instead of "s".
  example: tomato --> tomatoes, potato --> potatoes
  counter-example: faro --> no plural form (uncountable)
  etc.
* Fully irregular plurals exhibiting more complicated modifications
  examples: mouse --> mice, man --> men, foot --> feet, tooth --> teeth, ...

Computational Morphology for Nouns in English

The goal is to represent associations between strings representing:
- surface forms, i.e. words as they appear in texts;
and
- canonical representations, i.e. formal representations of the morphological analysis of these words

Examples of surface forms:
cats, book, flies, ...

Example of canonical representations:
cat+N+p, book+N+s, fly+N+p, ...

The typical format of a canonical representations is:
Lemma+GrammaticalCategory+MorphoSyntacticFeature$_1$+MorphoSyntacticFeature$_2$+...
where:
- Lemma (or Root) is the canonical form of an inflected word; i.e. the form usually found in dictionaries, e.g. the singular form for nouns, or the infinitive for verbs;
- GrammaticalCategory (or Part-of-Speech) is the tag used to represent the grammatical category of the word, e.g. N for a noun, Adj for an adjective, or V for a verb;
- MorphoSyntacticFeature$_k$ (k=1, 2, 3, ...) are the tags used to represent the morphosyntactic features (e.g. the number, the gender, the tense, the person, etc.) that are relevant to identify a specific inflection of a word;
and
- "+" is a (conventional) separating character.

For example:
- associating the canonical representation "cat+N+p" to the surface form "cats" expresses in a formal way that "cats" is the inflection of the noun "cat" corresponding to its plural form ("p" being the tag for the value "plural" of the morphosyntactic feature "number");
- associating the surface form "turns" to the canonical representation "turn+V+Ind+Pres+3+s" expresses in a formal way that the surface form corresponding to the inflection of the verb ("V") "to turn" at the 3rd person ("3") singular ("s") of the present ("Pres") indicative ("Ind") is "turns".

In other words, implementing Computational Morphology for English nouns is finding an efficient way of representing the (potentially very large) set of string associations of the form:

(cat+N+s, cat)
(cat+N+p, cats)
(book+N+s, book)
(book+N+p, books)
(fly+N+s, fly)
(fly+N+p, flies)
(fox+N+s, fox)
(fox+N+p, foxes)
(deer+N+s, deer)
(deer+N+p, deer)
(mouse+N+s, mouse)
(mouse+N+p, mice)
(ox+N+s, ox)
(ox+N+p, oxen)
etc.

where the left-hand side of the association represents a canonical representation, and the right-hand side the associated surface form.

By "efficient way", we mean a method that:
- allows to describe all the targeted associations without having to write them explicitly one-by-one;
- provides a computational mechanism with low algorithmic complexity able to produce the surface form(s) associated with a given canonical representation ("generation"), or the canonical representation(s) associated with a given surface form ("analysis").

Using transducers to implement Computational Morphology for English nouns

1. Implementing (the default rule for) regular noun plurals with transducers

As already mentioned, the default rule for making the plural of an English noun is:
add an "s" to the end of its singular

A first possibility to translate this rule in the form of a transducer is:
T1: ([^\+]+)((\+N\+p).x.(s))
where:
"(" and ")" are regular expression metacharacters allowing to delimit regular expressions (hereafter simply called regex and regexes) to prevent potential interpretation ambiguities;
"[^$c_1c_2…c_k$]" is a regex matching any (single) character other than the characters c_1, c_2…c_k;
"+" is the "Kleene plus" regex operator defined as: (e)+ is a regex that matches any non empty sequence of strings matching the regex e;
"\" is the "escape" metacharacter allowing to use metacharacter as matching characters in regexes (e.g. \+ is a regex matching the character "+" and not the Kleene plus operator); and
".x." represents the cross-product operator, defined as: a string association (s_1, s_2) is recognized by the transducer e_1 .x. e_2, where e_1 and e_2 are regular expressions, iff s_1 is recognized by e_1, and s_2 is recognized by e_2.

Question(s):

- Why is it recommendable to use the regex "([^\+]+)" as a prefix in the T1 transducer?
- How would the transducer T1 behave when processing the canonical representation "man+N+s+m", where the tag "m" represent the masculine gender?

To increase the readability of the manipulated transducers:
- the Kleene plus operator will be written "**+**" (in bold font), thus allowing to write the "+" character (very frequent in canonical representations) without the escape character "\"; and
- the ".x." cross-product operator will be written **x** (where the bold font differentiates it from the simple character "x").

With these conventions, the transducer T1 can be written as:

T1: ([^+]**+**)((+N+p)**x**(s))

This transducer will recognize associations such as:
(cat+N+p, cats)
(book+N+p, books),

but will also recognize incorrect associations such as:
(fly+N+p, flys)
(fox+N+p, foxs)
(deer+N+p, deers)
(mouse+N+p, mouses)
(ox+N+p, oxs)

and, in addition, will not recognize any association involving a singular noun, i.e. associated with a canonical representation of the form "…+N+s".

To correctly process singular nouns, the transducer T1 can be modified as follows:

T1: ([^+]**+**)((+N+p)**x**(s)) |
([^+]**+**)((+N+s)**x**(ε))

where:

"|" represents the "alternation" operator for transducers, defined as: a string association (s, s') is recognized by the transducer T1 | T2, iff it is recognized by T1 or by T2; and

"ε" represents the regex matching an empty string.

Question(s):

- How will the transducer T1 behave when processing the canonical representation limited to "+N+s" or "+N+p"?
- How will the transducer T1 behave when processing the surface form limited to "s"?

With the above modifications, the following associations corresponding to singular and/or regular plural forms are correctly processed:

(cat+N+s, cat)
(cat+N+p, cats)
(book+N+s, book)
(book+N+p, books)
(fly+N+s, fly)
(fox+N+s, fox)
(deer+N+s, deer)
(mouse+N+s, mouse)
(ox+N+s, ox)

but the associations corresponding to irregular plurals still need to be correctly taken into account.

2. Implementing semi-regular noun plurals with transducers

To implement semi-regular noun plurals, a handy approach is to use the notion of "**trace**", i.e. a special character (conventionally written X hereafter) to be included in the processed surface forms. The purpose of such a trace character is to preserve, in the surface form, the information about the location of the boundary between the lemma and the plural suffix "s", so that further processing of the systematic exceptions corresponding to euphonic rules can be implemented.

In this perspective, the transducer T1 is first modified as follows:

T1: ([^+]**+**)((+N+p)**x**(Xs)) |
([^+]**+**)((+N+s)**x**(X))

so as to generate the following associations:

(cat+N+s, catX)
(cat+N+p, catXs)
(book+N+s, bookX)
(book+N+p, bookXs)
(fly+N+s, flyX)
(fly+N+p, flyXs)
(fox+N+s, foxX)
(fox+N+p, foxXs)
(deer+N+s, deerX)
(deer+N+p, deerXs)
(mouse+N+s, mouseX)
(mouse+N+p, mouseXs)
(ox+N+s, oxX)
(ox+N+p, oxXs)

the left-hand side of which will serve as a basis for the implementation of the rules related to the semi-regular plurals.

Then, to implement such rules, a second transducer T2 will be created, and combined with the transducer T1 through the transducer "composition" operator **o**. In other words, the transducer T2 will be used to build the transducer T1 **o** T2, which produces string associations of the form (s, s') such that there exists a string s" such that: (s, s") is an association recognized by T1, and (s", s') is an association recognized by T2.

In this perspective, to implement the rule:

> if the singular noun ends in "s", "x", "z", "ch" or "sh", add an "es" instead of "s"

the following transducer T2 can be considered:

T2: ([^X]*)(s|x|z|ch|sh)((Xs)**x**(es))

where:
"*" is the "Kleene star" regex operator defined as: (e)* is a regex that matches, either the empty string, or any non empty sequence of strings that match the regex e; and

the metacharacter | represents the regex "alternation", i.e. $(e_1|e_2|\ldots|e_k)$ is a regex that recognizes any string that is recognized by any of the regexes e_1, e_2, …, e_k.

Question(s):

- Why has the regex "([^+]**+**)" used in T1 been replaced by the regex "([^X]*)" in the first part of the transducer alternation in T2?
- How will the transducer T2 behave when processing the left-hand side string "sXs"?
- How would the transducer T2 behave if the regexes used in the first regex alternation would not be exclusive (i.e. if several of these regexes could simultaneously match, as, for example, in the case of the regex "(([^X]*)(h|ch)")?

The transducer T2 recognizes the following associations:

(foxXs, foxes)
(oxXs, oxXs, oxes)
(guessXs, guesses)
(boxXs, boxes)
(buzzXs, buzzes)
(catchXs, catches)
(dishXs, dishes)
(stomachXs, stomaches)

However, it fails to recognize any association for all the valid left-hand side strings (i.e. strings corresponding to a right-hand side string recognized by T1) that do not match the regex "(([^X]*)(s|x|z|ch|sh)(Xs))", and thus needs to be extended to correctly cover these left-hand side strings as well:

T2: ([^X]*)(s|x|z|ch|sh)((Xs)**x**(es)) |
([^X]*****)([^sxzh]|([^cs]h))((Xs)**x**(s))

Question(s):

- Why does the second part of the transducer alternation contain the regex "([^sxzh]|([^cs]h))"?

Similarly, to implement the rule:

If the word ends in a consonant followed by "y", change the "y" to "ies"

the following transducer T2 can be considered:

T2: ([^X]*)([bcdfghjklmnpqrstvwxz])((yXs)**x**(ies))

where:

"[$c_1c_2…c_k$]" is a regex matching any of the characters c_1, $c_2…c_k$.

This transducer recognizes the following associations:
(babyXs, babies)
(candyXs, candies)

but, as for the previous rule, it fails to recognize any association for all the valid left-hand side strings (i.e. strings corresponding to a right-hand side string recognized by T1) that do not match the regex "(([^X]*)([bcdfghjklmnpqrstvwxz])(yXs))", and also needs to be extended to correctly cover these left-hand side strings:

T2: ([^X]*)([bcdfghjklmnpqrstvwxz])((yXs)**x**(ies)) |
([^X]*)(([^bcdfghjklmnpqrstvwxz]y)|[^y])((Xs)**x**(s))

Question(s):

- Why does the second part of the transducer alternation contain the regex "(([^bcdfghjklmnpqrstvwxz]y)|[^y])"?

As a result, one may think that the transducer T2 that simultaneously implements both of the rules is of the form:

T2: ([^X]*)(s|x|z|ch|sh)((Xs)**x**(es)) |
([^X]*****)([^sxzh]|([^cs]h))((Xs)**x**(s)) |
([^X]*)([bcdfghjklmnpqrstvwxz])((yXs)**x**(ies)) |
([^X]*)(([^bcdfghjklmnpqrstvwxz]y)|[^y])((Xs)**x**(s))

However, to guaranty that none of the string associations recognized by the transducer T2 can be produced several times, one must verify that the 4 parts of the above transducer alternation are exclusive. This is verified by construction for all the part pairs but the one consisting of the second and the fourth one.

Indeed, for these, we have that:

([^X]*****)([^sxzh]|([^cs]h))((Xs)**x**(s)) |
([^X]*)(([^bcdfghjklmnpqrstvwxz]y)|[^y])((Xs)**x**(s))

Is equivalent to:

([^X]*****)([^sxzh]|([^cs]h)|([^bcdfghjklmnpqrstvwxz]y)|[^y])((Xs)**x**(s))

and one must therefore verify that all the 4 parts of the regex alternation "([^sxzh]|([^cs]h)|([^bcdfghjklmnpqrstvwxz]y)|[^y])" are exclusive… which is not the case.

Question(s): Why?

In this case, the correct formulation of the regex alternation is:

"([^sxzhy]|([^cs]h)|([^bcdfghjklmnpqrstvwxz]y)"

and the resulting new version of the transducer T2 is therefore:

T2: ([^X]*)(s|x|z|ch|sh)((Xs)**x**(es)) |
([^X]*)([bcdfghjklmnpqrstvwxz])((yXs)**x**(ies)) |
([^X]*) ([^sxzhy]|([^cs]h)|([^bcdfghjklmnpqrstvwxz]y)((Xs)**x**(s))

Note that in the resulting transducer alternation, the third part represents the removal of the trace X in all the plural cases that are not covered by a rule governing some semi-regular plurals.

Finally, the T2 transducer must be further extended to be able to remove the trace X in the case of singular nouns (i.e. in left-hand sides of the form "([^X]**+**)X"). The final version of the transducer T2 is therefore:

T2: ([^X]*)(s|x|z|ch|sh)((Xs)**x**(es)) |
([^X]*)([bcdfghjklmnpqrstvwxz])((yXs)**x**(ies)) |
([^X]*)([^sxzhy]|([^cs]h)|([^bcdfghjklmnpqrstvwxz]y)((Xs)**x**(s)) |
([^X]*)((X)**x**())

where "()" conventionally represents the regex matching the empty character.

3. Implementing irregular noun plurals with transducers

By construction, the transducer T1 o T2 recognized all the string associations corresponding to singular nouns or regular and semi-regular noun plurals (at least those covered by the two implemented rules). However, it still produces the following incorrect string associations corresponding, either to exceptions to the rules governing the semi-regular plurals, or to irregular plurals:

(stomach+N+p, stomaches)
(ox+N+p, oxes)
(hair+N+p, hairs)
(mud+N+p, muds)
(deer+N+p, deers)
(knife+N+p, knifes)
(wife+N+p, wifes)
(half+N+p, halfs)
(hypothesis+N+p, hypothesises)
(crisis+N+p, crisises)
(tomato+N+p, tomatos)
(potato+N+p, potatos)
(mouse+N+p, mouses)
(man+N+p, mans)
(foot+N+p, foots)
(tooth+N+p, tooths)

To correctly process these exceptions, a handy approach is to create a third transducer T3, and combine it with the transducer T1 **o** T2, so as to associate the incorrect surface forms produced by T1 **o** T2 with the correct ones. The required transducer T3 is very easy to write, as it simply corresponds to a transducer alternation listing all the required string associations:

T3: (stomaches)**x**(stomachs) |
(oxes)**x**(oxen) |
(hairs)**x**(hair) |
(muds)**x**(mud) |
(deers)**x**(deer) |
(knifes)**x**(knives) |
(wifes)**x**(wives) |
(halfs)**x**(halves) |
(hypothesises)**x**(hypotheses)|
(crisises)**x**(crises) |
(tomatos)**x**(tomatoes) |
(potatos)**x**(potatoes) |
(mouses)**x**(mice) |
(mans)**x**(men) |
(foots)**x**(feet) |
(tooths)**x**(teeth)

However, this simple version for the transducer T3 is not fully sufficient, as:

- It does not process correctly all the correct associations produced by the transducer T1 **o** T2;
- It systematically associates a unique surface form with a given canonical representation, even for cases where two surfaces forms are acceptables (e.g. for “hair+N+p”, which can be associated, either with “hair”, or with “hairs”, or for “tomato+N+p”, which can be associated, either with “tomatos”, or with “tomatoes”)

To solve the first issue, it is necessary to extend the transducer alternation defining the transducer T3 with a specific transducer dealing with the default cases. To do so, a handy approach is to use a regex associated with a “negative lookahead”. Formally, this corresponds to use a regex of the form $e_1(?!e_2)$, where e_1 and e_2 are any regular expressions, defined as: “$e_1(?!e_2)$” recognizes any string that is recognized by e_1 and is not followed by a string recognized by e_2. With such a regex, the extended transducer T3 can then be defined as:

T3: (stomaches)**x**(stomachs) |
(oxes)**x**(oxen) |
(hairs)**x**(hair) |
(muds)**x**(mud) |
(deers)**x**(deer) |
(knifes)**x**(knives) |
(wifes)**x**(wives) |
(halfs)**x**(halves) |
(hypothesises)**x**(hypotheses)|
(crisises)**x**(crises) |
(tomatos)**x**(tomatoes) |
(potatos)**x**(potatoes) |
(mouses)**x**(mice) |
(mans)**x**(men) |
(foots)**x**(feet) |
(tooths)**x**(teeth) |
((^)(?!((stomaches)|(oxes)|(hairs)|…|(mans)|(foots)|(tooths))($))(.*)($))

where:

“^” is a regex matching an empty character at the beginning of the string to be processed;
“$” is a regex matching an empty character at the end of the string to be processed; and
“.” Is a regex matching any (single) character,

with the convention that a transducer reduced to a simple regex (i.e. the definition of which does not contain any transducer operator) recognizes all the string associations (s, s), where s is any string recognized by the regex.

Finally, to produce multiple surface forms associated with a given canonical representation (i.e. to solve the second issue), one can simply remove the corresponding “incorrect” surface form from the regex alternation “((stomaches)|(oxes)|(hairs)|…|(mans)|(foots)|(tooths))”.

For example, to associate the two surface forms “hair” and “hairs” to the canonical representation “hair+N+p”, i.e. to produce the two string associations:

(hair+N+p, hair)
(hair+N+p, hairs)

one just has to remove the “(hairs)” regex from the mentioned regex alternation.

In summary, applying Computational Morphology to implement the plurals of English nouns consists in writing the required 3 transducers T1, T2, and T3, and then using the left (resp. right) projection (or reduction) of the transducer T1 **o** T2 **o** T3 to generate the surface form(s) (resp. canonical representation(s)) associated to a given canonical representation (resp. surface form).