Here is a case study on how to code up a stemming algorithm in Snowball. First, the definition of the Porter stemmer, as it appeared in *Program*, Vol 14 no. 3 pp 130-137, July 1980.

THE ALGORITHM

A *consonant* in a word is a letter other than A, E, I, O or U, and other than Y preceded by a consonant. (The fact that the term ‘consonant’ is defined to some extent in terms of itself does not make it ambiguous.) So in TOY the consonants are T and Y, and in SYZYGY they are S, Z and G. If a letter is not a consonant it is a *vowel*.

A consonant will be denoted by c, a vowel by v. A list ccc... of length greater than 0 will be denoted by C, and a list vvv... of length greater than 0 will be denoted by V. Any word, or part of a word, therefore has one of the four forms:
CVCV ... C
CVCV ... V
VCVC ... C
VCVC ... V
These may all be represented by the single form
    [C]VCVC ... [V]
where the square brackets denote arbitrary presence of their contents. Using $(VC)^m$ to denote VC repeated m times, this may again be written as
    $[C](VC)^m[V]$.
m will be called the *measure* of any word or word part when represented in this form. The case m = 0 covers the null word. Here are some examples:
    m=0 TR,   EE,   TREE,   Y,   BY.
    m=1 TROUBLE,   OATS,   TREES,   IVY.
    m=2 TROUBLES,   PRIVATE,   OATEN,   ORRERY.
The *rules* for removing a suffix will be given in the form
    (condition) S1 -> S2
This means that if a word ends with the suffix S1, and the stem before S1 satisfies the given condition, S1 is replaced by S2. The condition is usually given in terms of m, e.g.
    (m > 1) EMENT ->
Here S1 is ‘EMENT’ and S2 is null. This would map REPLACEMENT to REPLAC, since REPLAC is a word part for which m = 2.

The ‘condition’ part may also contain the following:
    *S   - the stem ends with S (and similarly for the other letters).
    v - the stem contains a vowel.
    *d   - the stem ends with a double consonant (e.g. -TT, -SS).

*o - the stem ends cvc, where the second c is not W, X or Y (e.g. -WIL, -HOP).

And the condition part may also contain expressions with *and*, *or* and *not*, so that

(m>1 and (*S or *T))

tests for a stem with m>1 ending in S or T, while

(*d and not (*L or *S or *Z))

tests for a stem ending with a double consonant other than L, S or Z. Elaborate conditions like this are required only rarely.

In a set of rules written beneath each other, only one is obeyed, and this will be the one with the longest matching S1 for the given word. For example, with

SSES	-> SS
IES	-> I
SS	-> SS
S	->

(here the conditions are all null) CARESSES maps to CARESS since SSES is the longest match for S1. Equally CARESS maps to CARESS (S1=‘SS’) and CARES to CARE (S1=‘S’).

In the rules below, examples of their application, successful or otherwise, are given on the right in lower case. The algorithm now follows:

Step 1a

SSES	-> SS	caresses	-> caress
IES	-> I	ponies	-> poni
		ties	-> ti
SS	-> SS	caress	-> caress
S	->	cats	-> cat

Step 1b

(m>0) EED	-> EE	feed	-> feed
		agreed	-> agree
(*v*) ED	->	plastered	-> plaster
		bled	-> bled
(*v*) ING	->	motoring	-> motor
		sing	-> sing

If the second or third of the rules in Step 1b is successful, the following is done:

AT	-> ATE	conflat(ed)	-> conflate
BL	-> BLE	troubl(ed)	-> trouble
IZ	-> IZE	siz(ed)	-> size
(*d and not (*L or *S or *Z))	-> single letter	hopp(ing)	-> hop
		tann(ed)	-> tan
		fall(ing)	-> fall
		hiss(ing)	-> hiss

		fizz(ed)	-> fizz
(m=1 and *o)	-> E	fail(ing)	-> fail
		fil(ing)	-> file

The rule to map to a single letter causes the removal of one of the double letter pair. The -E is put back on -AT, -BL and -IZ, so that the suffixes -ATE, -BLE and -IZE can be recognised later. This E may be removed in step 4.

Step 1c

(*v*) Y	-> I	happy	-> happi
		sky	-> sky

Step 1 deals with plurals and past participles. The subsequent steps are much more straightforward.

Step 2

(m>0) ATIONAL	-> ATE	relational	-> relate
(m>0) TIONAL	-> TION	conditional	-> condition
		rational	-> rational
(m>0) ENCI	-> ENCE	valenci	-> valence
(m>0) ANCI	-> ANCE	hesitanci	-> hesitance
(m>0) IZER	-> IZE	digitizer	-> digitize
(m>0) ABLI	-> ABLE	conformabli	-> conformable
(m>0) ALLI	-> AL	radicalli	-> radical
(m>0) ENTLI	-> ENT	differentli	-> different
(m>0) ELI	-> E	vileli	-> vile
(m>0) OUSLI	-> OUS	analogousli	-> analogous
(m>0) IZATION	-> IZE	vietnamization	-> vietnamize
(m>0) ATION	-> ATE	predication	-> predicate
(m>0) ATOR	-> ATE	operator	-> operate
(m>0) ALISM	-> AL	feudalism	-> feudal
(m>0) IVENESS	-> IVE	decisiveness	-> decisive
(m>0) FULNESS	-> FUL	hopefulness	-> hopeful
(m>0) OUSNESS	-> OUS	callousness	-> callous
(m>0) ALITI	-> AL	formaliti	-> formal
(m>0) IVITI	-> IVE	sensitiviti	-> sensitive
(m>0) BILITI	-> BLE	sensibiliti	-> sensible

The test for the string S1 can be made fast by doing a program switch on the penultimate letter of the word being tested. This gives a fairly even breakdown of the possible values of the string S1. It will be seen in fact that the S1-strings in step 2 are presented here in the alphabetical order of their penultimate letter. Similar techniques may be applied in the other steps.

Step 3

(m>0) ICATE	-> IC	triplicate	-> triplic

(m>0) ATIVE	->		formative	-> form
(m>0) ALIZE	-> AL		formalize	-> formal
(m>0) ICITI	-> IC		electriciti	-> electric
(m>0) ICAL	-> IC		electrical	-> electric
(m>0) FUL	->		hopeful	-> hope
(m>0) NESS	->		goodness	-> good

Step 4

(m>1) AL	->	revival	-> reviv
(m>1) ANCE	->	allowance	-> allow
(m>1) ENCE	->	inference	-> infer
(m>1) ER	->	airliner	-> airlin
(m>1) IC	->	gyroscopic	-> gyroscop
(m>1) ABLE	->	adjustable	-> adjust
(m>1) IBLE	->	defensible	-> defens
(m>1) ANT	->	irritant	-> irrit
(m>1) EMENT	->	replacement	-> replac
(m>1) MENT	->	adjustment	-> adjust
(m>1) ENT	->	dependent	-> depend
(m>1 and (*S or *T)) ION	->	adoption	-> adopt
(m>1) OU	->	homologou	-> homolog
(m>1) ISM	->	communism	-> commun
(m>1) ATE	->	activate	-> activ
(m>1) ITI	->	angulariti	-> angular
(m>1) OUS	->	homologous	-> homolog
(m>1) IVE	->	effective	-> effect
(m>1) IZE	->	bowdlerize	-> bowdler

The suffixes are now removed. All that remains is a little tidying up.

Step 5a

(m>1) E	->	probate	-> probat
		rate	-> rate
(m=1 and not *o) E	->	cease	-> ceas

Step 5b

(m > 1 and *d and *L)	-> single letter	controll	-> control
		roll	-> roll