Finite State Automata for morphological analysis
Exercise #4
Due: February 3, 2000

Problem 0: FSA Tutorial
If you haven't used the Finite State Automata utilities much before, try working through the tutorial for them that Gosse Bouma has prepared. You can find it on the web at `http://www.let.rug.nl/~gosse/tt/fsa.html`. This is just a warm-up exercise, so you don't have to write up the answers. But, if you have any trouble using the FSA tools, please let me know.

Problem 1: Esperanto
Based on the examples in Problem 3 of Exercise 3, construct a finite state automaton for recognizing Esperanto word structure. Remember, since Esperanto morphology is completely regular, you don't need to worry about morphophonemic rules. But, you will need to construct letter trees for classes of morphemes and FSAs for assembling morphemes into words.

To get started, first create a file called `esperanto.pl` with the following contents:

```
%%% esperanto.pl - Regular expressions for Esperanto word structure
:- multifile macro/2.

% A macro for constructing a letter tree from a list of words
macro(words(List0), set(List1)) :-
        words(List0,List1).

words([],[]).
words([H0|T0], [word(H0)|T]) :-
        words(T0,T).

% Letter trees for classes of roots
macro(noun_root, words([knab,patr])).
macro(verb_root, words([pov,est])).

% A noun is a noun root plus an optional "in" plus "o"
macro(noun, [noun_root,word(in)^,o]).

% A verb is a verb root plus "as" or "is" or "i"
macro(verb, [verb_root,{word(as),word(is),i}]).

% A word is either a noun, a verb, or a particle
macro(word, set([noun, verb, words([la,'c^u'])])).
```

This file defines a number of short-cuts that will make it easier to write a regular expression describing Esperanto words. As you develop your analysis, you will want to add additional macro definitions to this file.

To use these macros to build an FSA which recognizes Esperanto words which match the regular expression `word`, use the command:

```
hagen% fsa -aux esperanto.pl -r word esperanto.fsa
```

Now you can use `fsa` to test the resulting FSA. To check whether your FSA recognizes a particular word, use the `-accepts` option. You can also ask the program to generate all the words recognized by the FSA using the `-produce` option:

```
hagen% fsa -accepts esperanto.fsa patrino
yes
hagen% fsa -accepts esperanto.fsa patra
no
hagen% fsa -produce esperanto.fsa
la
cˆu
esti
povi
estas
estis
knabo
patro
povas
povis
knabino
patrino
```

For complete details about how to use the `fsa` command, see the documentation at `http://odur.let.rug.nl/~vannoord/Fsa/Manual/`.

Once you have things working they way you like it, please send me the recognizer produced by the `fsa` program, along with any macro files you used and a script showing the commands that you used to produce the recognizer.

Problem 2: Diminutives

Construct a transducer which will pair up Dutch nouns with their diminutive forms. Assume the underlying form is something like `boek+je`. Your transducer should map the underlying form into the correct surface form, in this case `boekje`. At a minimum your transducer should be able to handle pairs like the following:

aap+je	aapje	ding+je	dingetje
huis+je	huisje	bel+je	belletje
hand+je	handje	pan+je	pannetje
ui+je	uitje	duim+je	duimpje
kamer+je	kamertje	koning+je	koninkje
tafel+je	tafeltje	leerling+je	leerlingetje
vrouw+je	vrouwtje	auto+je	autootje

However, feel free to try trickier pairs as well.

The easiest way to construct a transducer like this is to use the `replace` macro (defined in `~malouf/nlp/replace.pl`). This macro takes three arguments: a transducer, a left context recognizer, and a right context recognizer. For example, the regular expression `replace(d:x,c,e)` produces a transducer that changes a `d` into an `x` just in case it appears between a `c` and an `e`. (So, it would map the string `abcdefedcba` into the string `abcxefedcba`). More complex mappings can be built up from a sequence of `replace` rules using regular expression operations like composition, union, and intersection.

Here's a start at a transducer for the diminutives. It has two rules, one which doubles a long vowel at the end of a word just before a morpheme boundary, and one which deletes morpheme boundary markers from the underlying forms:

```
%%% diminutive.pl - Regular expressions for Dutch diminutives
:- multifile macro/2.

% Define classes of letters
macro(vowel, {a, e, i, o, u}).
macro(consonant, {a..z-vowel}).

% Double a long vowel at the end of a word
macro(rule1, replace({[a, []:a], [e, []:e], [o, []:o], [u, []:u]},
consonant, escape(+))).

% Delete all morpheme boundary markers (since there is no left or right
% context, this rule applies everywhere)
macro(rule2, replace(escape(+):[])).

% Combine both rules into a single transducer
macro(dim, rule1 o rule2).
```

The two rules are then composed, so that the output of `rule1` is the input of `rule2`. To complete the transducer, compose addition rules so that the output of the transducer is the correct surface form.

Since the transducer you write doesn't include a list of valid nouns in Dutch (you can assume that the underlying form will be a legitimate noun), you can't sensibly test the transducer by producing all the pairs of forms that it licenses, as you did to test the Esperanto word recognizer. Instead, to test your transducer, create a file with a list of

underlying forms, one per line (and be careful that that aren't any extra spaces at the ends of lines), and use that as input to the `fsa -ta` ('transduce all') command:

```
hagen% fsa -aux replace.pl -aux diminutive.pl -r dim >diminutive.fsa
hagen% fsa -ta diminutive.fsa <test.txt
aapje
huisje
handje
uije
kamerje
tafelje
vrouwje
dingje
belje
kamje
panje
duimje
koningje
leerlingje
autooje
```

The output will be a list of surface forms computed from each underlying form. To find bugs, compare the forms the transducer licenses to the actual correct surface forms (obviously, this one still needs some work).

As with Problem 1, when you have a working transducer, please send me the output of the `fsa` program, along with any macro files you used and a script showing the steps you took to construct it.