OSTBAYERISCHE TECHNISCHE HOCHSCHULE REGENSBURG

INFORMATIK UND MATHEMATIK

Natural Language Processing

Thema: Exercise sheet 2: Finite state transducers to model morphology and pronunciation

Von: Timo Baumann, timo.baumann@oth-regensburg.de

Datum: 04.04.2024

Abgabe bis: 10.04.2024 24:00

The dot language[1] is useful to describe finite state automata (transducers, and many more models) which can then be compiled to images via GraphViz (https://www.graphviz.org) locally, or using WebGraphviz (http://webgraphviz.com/) in the browser.

Please use the setting rankdir="LR", to yield graphs that are layouted left-to-right.

In-class exercises

C.1 FSAs and regular expressions

Work on the following tasks in small groups (3–4 students; **NOT** your submission team members!) for 15 minutes. Take care (everyone!) to understand and be able to follow along what your fellow students are discussing and doing.

1. Define a finite state automaton that accepts the language $\{a^n b^2 c^m | n, m \geq 0\}$.
2. Construct the left-linear grammar that defines this language. How about the right-linear grammar?
3. How would you implement (produce pseudo-code!) an FSA for this regular expression? How universally applicable is your approach?

[1] https://graphviz.org/doc/info/lang.html

H 2 Take-home exercises

Please hand in the exercise solutions via the Moodle course, one submission per submission team of 2–3 students.

It's useful to submit in a format that can integrate the Graphviz-produced figures (e. g. PDF, HTML or Markdown). Do not send a `.docx` file.

For the second exercise, please send your remarks and answers to the questions as well as the resulting files. Do not send your sequitur installation, I don't have unlimited storage.

H 2.1 Finite state transducers for morphological analysis

Note: In Graphviz' dot language, you describe transducers exactly like FSAs. Use Mealy's representation (like in the lecture) in which the edge inscriptions consist of `in/out` (or `in:out`) where `in` is the symbol to be read and `out` is the symbol to be written and the slash (or colon) is used to separate the two.

1. Construct an FST that is able to analyze/generate the following morphology:

surface form	lexical analysis
abgab	abgeben+Verb+Pers13sg+Impf
abgeben	abgeben+Verb+Inf
abgegeben	abgeben+Verb+Part2
abgegebene	abgegeben+Adj+Nom+Pl
weggab	weggeben+Verb+Pers13sg+Impf
weggeben	weggeben+Verb+Inf
weggegeben	weggeben+Verb+Part2
weggegebene	weggegeben+Adj+Nom+Pl

   Note that your automaton may only consume/produce up to 1 symbol from the input/output tapes when transitioning from one state to the next (but things like "+Verb" are one symbol, not multiple).
2. Apply the matching algorithm (as described in the lecture) to "abgegebenen" and note all intermediate steps.
3. Consolidate the automaton by allowing edge inscriptions that combine multiple symbols to avoid long sequences of states (i.e., allow inscriptions from $\Sigma^* : \Gamma^*$).
4. Add additional prefixes to your automaton: `durch-`, `auf-`, `bekannt-`, `zurück-` and `zu-` and enable the analysis of the forms of geben/gegeben without a prefix.

H 2.2 Training of an FST-based pronunciation model

In this exercise, you'll use *Sequitur G2P*: `https://github.com/sequitur-g2p/sequitur-g2p/`. You will also need a functioning PERL and PYTHON installation in order to be able to transform (via some detours) the FSTs that are generated by Sequitur into the `dot` format which you can turn into images.

We'll collect issues and hints for installing these tools in the Moodle forum.

Our pronunciation lexica use SAMPA[2] to describe the German speech sounds.

1. Install Sequitur and follow the step-by-step tutorial. Where the tutorial uses `train.lex`, please use `minimal.lex` from Moodle. (Your results will be called `model-1`, `model-2`, `model-3`, …)
2. Use the models to transcript the words `lag`, `sahne`, und `nase`. Which models transcribe which words correct or wrong?
3. Search for an (alternative/misspelled) writing for `nase` that is transcribed correctly by as many models (which ones?) as possible
4. Use `fsa.py` on each of your models in order to transform the Sequitur-internal representation of the FST into an XML representation. Then, use `fsa2dot.pl`[3] in order to convert the XML to `dot`. Produce a graphical representation from `dot` using Graphviz.
5. Compare the FSTs of your models. What happens in the FST when you train for an additional iteration?
6. Finally, train a model based on a larger pronunciation dictionary of German for several iterations (`Cocolab_DE.lex`; the resulting models unfortunately quickly grow too large to be visualized). Let Sequitur transcribe your first name, your last name and a street name of your choice. How well do the models work?

[2] `https://www.phon.ucl.ac.uk/home/sampa/german.htm`

[3] You may have to install the Perl modules `XML::Simple`, e. g. on Ubuntu with `apt install libxml-simple-perl`, or from CPAN with `cpan XML::Simple`.