

Introduction to the Artificial Intelligence methods

Finite State methods in Natural Language Processing in Prolog

Federal State Educational Institution of Higher Professional Education «Novosibirsk State University»
(National Research University)

21 октября 2019 г.

Finite State Automata

Finite State Automata

Introduction
to the
Artificial
Intelligence
methods

Finite State
Automata

Finite State
Parsers and
Transducers

Finite State
Methods in
NLP

- 1 Finite State Recognizers and Generators
 - 1 Basics and simple machines
 - 2 Finite State Automata
- 2 Deterministic and Non-deterministic Automata
- 3 Implementing FSA in Prolog
 - 1 Defining FSA in Prolog
 - 2 A Recognizer and Generator in Prolog with no jumps
 - 3 A Recognizer and Generator in Prolog with jumping arcs
- 4 Finite State Methods in Computational Linguistics and NLP

Finite State Automata

Finite State Recognizers and Generators

Introduction
to the
Artificial
Intelligence
methods

Finite State
Automata

Finite State
Parsers and
Transducers

Finite State
Methods in
NLP

- Finite State Automata are used in morphology, phonology, text to speech, and data mining.
- They are simple and well understood mathematically.
- They are easy to implement and implementations are usually very efficient.
- With all simplicity FSA are restricted in what they are able to do.
- There is not a finite state solution for every NLP problem.

Finite State Automata

Finite State Recognizers and Generators

Introduction
to the
Artificial
Intelligence
methods

Finite State
Automata

Finite State
Parsers and
Transducers

Finite State
Methods in
NLP

- A **finite state generator** is a simple computing machine that outputs a sequence of symbols.
- FSG has a finite number of different states (obviously it has).
- It starts in some start state and then tries to reach a final state by making transitions from one state to another. Every time it makes such a transition it **emits** a symbol.
- It cannot stop until it reaches a final state.
- FSG only know the state it is currently in, and cannot look ahead at the states that come and also doesn't have any memory of the states it has been.

Finite State Automata

Finite State Recognizers and Generators

Introduction
to the
Artificial
Intelligence
methods

Finite State
Automata

Finite State
Parsers and
Transducers

Finite State
Methods in
NLP

- Finite state generators can be thought of as directed graphs.
- In fact finite state generators are usually drawn as directed graphs.

Finite State Automata

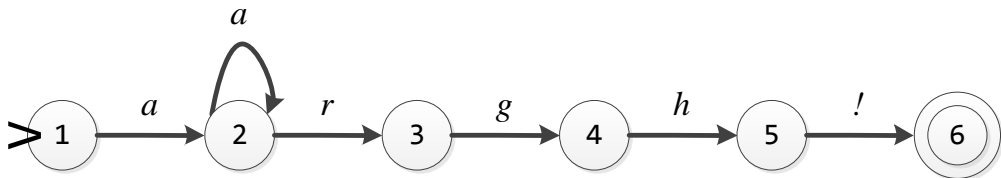
Finite State Recognizers and Generators

Introduction
to the
Artificial
Intelligence
methods

Finite State
Automata

Finite State
Parsers and
Transducers

Finite State
Methods in
NLP



Finite State Automata

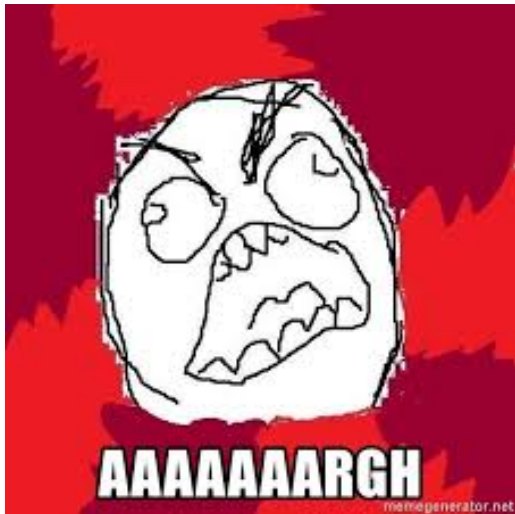
Finite State Recognizers and Generators

Introduction
to the
Artificial
Intelligence
methods

Finite State
Automata

Finite State
Parsers and
Transducers

Finite State
Methods in
NLP



Finite State Automata

Finite State Recognizers and Generators

Introduction
to the
Artificial
Intelligence
methods

Finite State
Automata

Finite State
Parsers and
Transducers

Finite State
Methods in
NLP

- Finite state recognizers are simple computing machines that read a sequence of symbols from an input tape.
- In fact, finite state generators and finite state recognizers are exactly the same kind of machine.
- An FSA recognizes (or accepts) a string of symbols s_1, s_2, s_3, \dots
- Starting in an initial state it can read in the symbols one after the other while making transitions from one state to another such that the transition reading in the last symbol takes the machine into a final state.
- An FSA fails to recognize a string if:
 - It cannot reach a final state or
 - it can reach a final state, but when it does there are still unread symbols left over

Finite State Automata

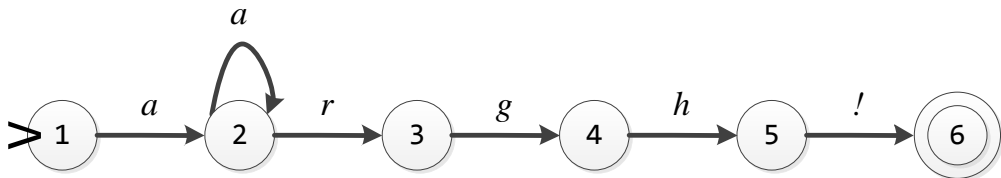
Finite State Recognizers and Generators

Introduction
to the
Artificial
Intelligence
methods

Finite State
Automata

Finite State
Parsers and
Transducers

Finite State
Methods in
NLP



Finite State Automata

Finite State Recognizers and Generators

Introduction
to the
Artificial
Intelligence
methods

Finite State
Automata

Finite State
Parsers and
Transducers

Finite State
Methods in
NLP

- A formal language is a set of strings.
- The language recognized by an FSM is the set of all strings it recognizes when used in recognition mode.
- The language generated by an FSM is the set of all strings it can generate when used in generation mode.
- The language accepted and the language generated by an FSM are exactly the same.

Finite State Automata

Some Examples

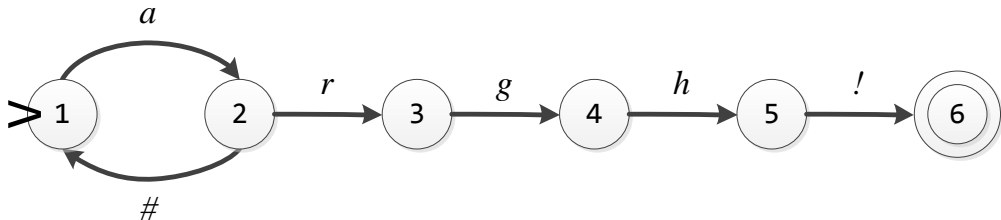
Introduction
to the
Artificial
Intelligence
methods

Finite State
Automata

Finite State
Parsers and
Transducers

Finite State
Methods in
NLP

Automaton with a jump arc.



Finite State Automata

Some Examples

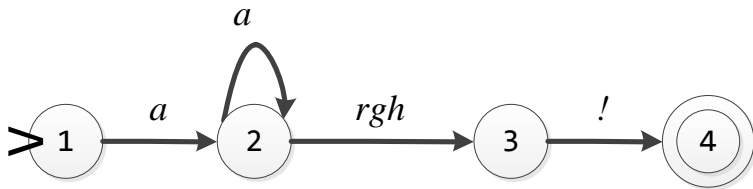
Introduction
to the
Artificial
Intelligence
methods

Finite State
Automata

Finite State
Parsers and
Transducers

Finite State
Methods in
NLP

Another alphabet.



Finite State Automata

Some Examples

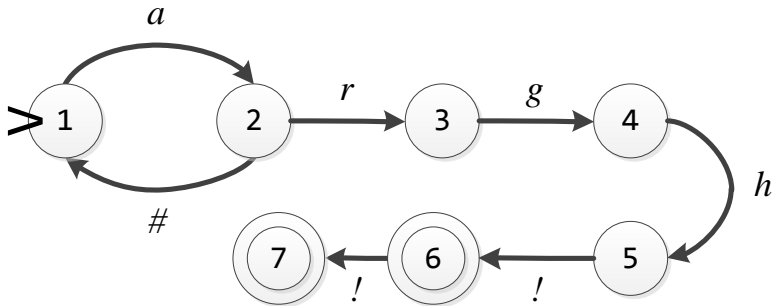
Introduction
to the
Artificial
Intelligence
methods

Finite State
Automata

Finite State
Parsers and
Transducers

Finite State
Methods in
NLP

Automaton with multiple final states.



Finite State Automata

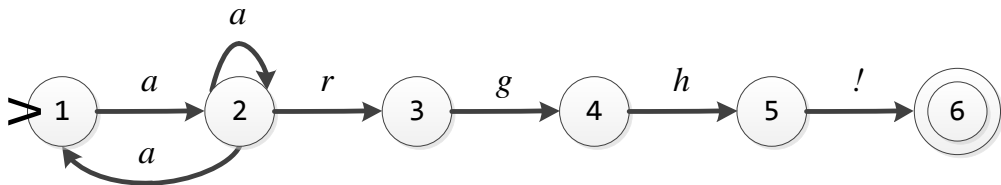
Deterministic and Non-deterministic Automata

Introduction
to the
Artificial
Intelligence
methods

Finite State
Automata

Finite State
Parsers and
Transducers

Finite State
Methods in
NLP



Finite State Automata

Deterministic and Non-deterministic Automata

Introduction
to the
Artificial
Intelligence
methods

Finite State
Automata

Finite State
Parsers and
Transducers

Finite State
Methods in
NLP

- This automaton has two arcs labelled with the same symbol going out of one state – this FSA is **non-deterministic**.
- Non-determinism doesn't add anything to what FSAs can do.
- When using an automaton for generation, there is no big difference between deterministic and non-deterministic machines.
- There is, however, a difference when we want to use the machine for recognition: non-determinism brings with it the need to perform **search**.

Finite State Automata

Implementing FSA in Prolog

Introduction
to the
Artificial
Intelligence
methods

Finite State
Automata

Finite State
Parsers and
Transducers

Finite State
Methods in
NLP

- We are going to treat FSAs as passive data structures that are manipulated by other programs.
- Separating declarative and procedural information is often a good idea.
- We will use three predicates to represent FSAs:
 - 1 start/1
 - 2 end/1
 - 3 arc/3

Finite State Automata

Implementing FSA in Prolog

Introduction
to the
Artificial
Intelligence
methods

Finite State
Automata

Finite State
Parsers and
Transducers

Finite State
Methods in
NLP

Simple screaming machine

```
start(1).
```

```
end(6).
```

```
arc(1,2,a).
```

```
arc(2,3,r).
```

```
arc(2,2,a).
```

```
arc(3,4,g).
```

```
arc(4,5,h).
```

```
arc(5,6,!).
```

Finite State Automata

Implementing FSA in Prolog

Introduction
to the
Artificial
Intelligence
methods

Finite State
Automata

Finite State
Parsers and
Transducers

Finite State
Methods in
NLP

Non-deterministic screaming machine

```
start(1).
```

```
end(6).
```

```
arc(1,2,a).
```

```
arc(2,3,r).
```

```
arc(2,1,a).
```

```
arc(2,2,a).
```

```
arc(3,4,g).
```

```
arc(4,5,h).
```

```
arc(5,6,!).
```

Finite State Automata

Implementing FSA in Prolog

Introduction
to the
Artificial
Intelligence
methods

Finite State
Automata

Finite State
Parsers and
Transducers

Finite State
Methods in
NLP

Screaming machine with a jumping arc

```
start(1).
```

```
end(6).
```

```
arc(1,2,a).
```

```
arc(2,3,r).
```

```
arc(2,1,'#').
```

```
arc(3,4,g).
```

```
arc(4,5,h).
```

```
arc(5,6,!).
```

Finite State Automata

Implementing FSA in Prolog

Introduction
to the
Artificial
Intelligence
methods

Finite State
Automata

Finite State
Parsers and
Transducers

Finite State
Methods in
NLP

Recognizer and Generator without jump arcs

```
recognize(Node, []) :- end(Node).  
recognize(Node, String) :- arc(Node, Next, Label),  
                             traverse(Label, String, NewStr),  
                             recognize(Next, NewStr).  
  
traverse(Label, [Label | Symbols], Symbols).  
  
run(Symbols) :- start(Node), recognize(Node, Symbols).
```

Recognizer and Generator with jump arcs

```
recognize1(Node,[]) :- end(Node).  
recognize1(Node,String) :- arc(Node,Next,Label),  
                             traverse1(Label,String,NewStr),  
                             recognize1(Next,NewStr).  
  
traverse1('#',String,String).  
traverse1(Label,[Label|Symbols],Symbols).  
run1(Symbols) :- start(Node),recognize1(Node,Symbols).
```

Finite State Automata

Finite State Methods in Computational Linguistics and NLP

Introduction
to the
Artificial
Intelligence
methods

Finite State
Automata

Finite State
Parsers and
Transducers

Finite State
Methods in
NLP

- Finite state machines are very simple, but there are limitations to what they can do.
- FSA can only recognize and generate **Regular Languages**.
- Many linguistic phenomena can only be described by languages which cannot be generated by FSAs.
- There are, however, linguistic applications where the expressive power of finite state methods seems to be sufficient.
- Finite state methods have been shown to be particularly useful in the areas of phonological and morphological processing, and have also been applied to syntactic analysis.

Finite State Parsers and Transducers

Finite State Parsers and Transducers

Introduction
to the
Artificial
Intelligence
methods

Finite State
Automata

Finite State
Parsers and
Transducers

Finite State
Methods in
NLP

- 1 Building Structure while Recognizing
 - 1 Finite State Parsers
 - 2 Separating out the Lexicon
- 2 Finite State Transducers
 - 1 What are Finite State Transducers?
 - 2 FSTs in Prolog
- 3 Morphological Analysis with Finite State Transducers

Finite State Parsers and Transducers

Building Structure while Recognizing

Introduction
to the
Artificial
Intelligence
methods

Finite State
Automata

Finite State
Parsers and
Transducers

Finite State
Methods in
NLP

- In addition to knowing that something is accepted by a certain FSA, we would like to have an explanation of why it was accepted.
- **Finite State Parsers** give us that kind of explanation by returning the sequence of transitions that was made.
- The parser output should tell us about the transitions that had to be made in the FSA when the input was recognized.
- There is a standard technique in Prolog for turning a recognizer into a parser: add one or more extra arguments to keep track of the structure that was found.

Finite State Parsers and Transducers

Building Structure while Recognizing

Introduction
to the
Artificial
Intelligence
methods

Finite State
Automata

Finite State
Parsers and
Transducers

Finite State
Methods in
NLP

In the base clause, when the input is read and the FSA is in a final state, all we have to do is record that final state.

```
recognize(Node, []) :- end(Node).
```

Finite State Parsers and Transducers

Building Structure while Recognizing

Introduction
to the
Artificial
Intelligence
methods

Finite State
Automata

Finite State
Parsers and
Transducers

Finite State
Methods in
NLP

In the base clause, when the input is read and the FSA is in a final state, all we have to do is record that final state.

```
recognize(Node, []) :- end(Node).
```

```
parse(Node, [Node], []) :- end(Node).
```

Finite State Parsers and Transducers

Building Structure while Recognizing

Introduction
to the
Artificial
Intelligence
methods

Finite State
Automata

Finite State
Parsers and
Transducers

Finite State
Methods in
NLP

The recursive clause looked as follows:

```
recognize(Node,String) :- arc(Node,Next,Label),  
                           traverse(Label,String,NewStr),  
                           recognize(Next,NewStr).
```

Finite State Parsers and Transducers

Building Structure while Recognizing

Introduction
to the
Artificial
Intelligence
methods

Finite State
Automata

Finite State
Parsers and
Transducers

Finite State
Methods in
NLP

The recursive clause looked as follows:

```
recognize(Node,String) :- arc(Node,Next,Label),  
                           traverse(Label,String,NewStr),  
                           recognize(Next,NewStr).
```

```
parse(Node,[Node,Label|Path],String) :- arc(Node,Next,Label),  
                                         traverse(Label,String,NewStr),  
                                         parse(Next,Path,NewStr).
```

Finite State Parsers and Transducers

Building Structure while Recognizing

Introduction
to the
Artificial
Intelligence
methods

Finite State
Automata

Finite State
Parsers and
Transducers

Finite State
Methods in
NLP

Finally the whole program looks like this:

```
parse(Node, [Node], []) :- end(Node).  
parse(Node, [Node, Label | Path], String) :- arc(Node, Next, Label),  
                                             traverse(Label, String, NewStr),  
                                             parse(Next, Path, NewStr).  
runParser(Symbols, Structure) :- start(Node),  
                                 parse(Node, Structure, Symbols),  
                                 atomics_to_string(Structure, ' ', Out),  
                                 writeln(Out).
```

Finite State Parsers and Transducers

Building Structure while Recognizing

Introduction
to the
Artificial
Intelligence
methods

Finite State
Automata

Finite State
Parsers and
Transducers

Finite State
Methods in
NLP

- The parser records the state the FSA is in and the symbol it is reading on the transition it is taking from this state.
- The rest of the path will be specified in the recursive call of `parse/3` and collected in the variable `Path`.
- Calling `runParser(_,_)` we will get all possible screams along with a full information about how exactly these words were generated.

Finite State Parsers and Transducers

Building Structure while Recognizing

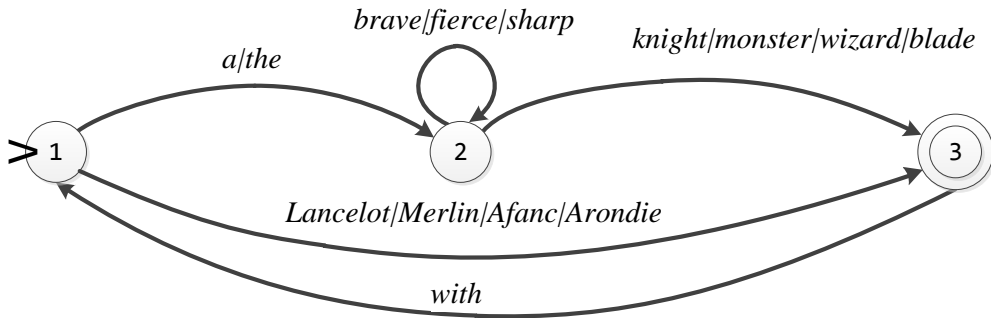
Introduction
to the
Artificial
Intelligence
methods

Finite State
Automata

Finite State
Parsers and
Transducers

Finite State
Methods in
NLP

Let us have a look at the FSA below:



Finite State Parsers and Transducers

Building Structure while Recognizing

Introduction
to the
Artificial
Intelligence
methods

Finite State
Automata

Finite State
Parsers and
Transducers

Finite State
Methods in
NLP

If you decided to construct it with Prolog you would probably end up with something like this:

| | |
|---------------------------------|--------------------------------|
| <code>start(1).</code> | <code>arc(1,3,merlin).</code> |
| <code>end(3).</code> | <code>arc(1,3,afanc).</code> |
| <code>arc(1,2,a).</code> | <code>arc(1,3,arondie).</code> |
| <code>arc(1,2,the).</code> | <code>arc(2,3,knight).</code> |
| <code>arc(2,2,brave).</code> | <code>arc(2,3,monster).</code> |
| <code>arc(2,2,fierce).</code> | <code>arc(2,3,wizard).</code> |
| <code>arc(2,2,sharp).</code> | <code>arc(2,3,blade).</code> |
| <code>arc(1,3,lancelot).</code> | <code>arc(3,1,with).</code> |

Finite State Parsers and Transducers

Building Structure while Recognizing

Introduction
to the
Artificial
Intelligence
methods

Finite State
Automata

Finite State
Parsers and
Transducers

Finite State
Methods in
NLP

- If we used the parser of the previous section on this automaton to parse the input [a,knight,with,a,blade], it would return (1 a 2 knight 3 with 1 a 2 blade 3).
- But in a way, it would be even nicer, if we got a more abstract explanation saying, e.g., that [a,knight,with,a,blade] is a noun phrase because it consists of a determiner followed by an noun which is followed by an another noun phrase.
- (1 det 2 common noun 3 prep 1 det 2 common noun 3).
- In fact, it would be a lot nicer, if you could specify transitions in the FSA based on **categories** like determiner, common noun, and so on and additionally give a separate lexicon which specifies what words belong to a category.

Finite State Parsers and Transducers

Building Structure while Recognizing

Introduction
to the
Artificial
Intelligence
methods

Like this, for example:

| | | |
|----------------|------------------|-------------------|
| start(1). | lex(a, det). | lex(wizard,cn). |
| end(3). | lex(the,det). | lex(blade,cn). |
| arc(1,2,det). | lex(fast,adj). | lex(lancelot,pn). |
| arc(2,2,adj). | lex(fierce,adj). | lex(merlin,pn). |
| arc(2,3,cn). | lex(sharp,adj). | lex(aranc,pn). |
| arc(1,3,pn). | lex(knight,cn). | lex(arondie,pn). |
| arc(3,1,prep). | lex(monster,cn). | lex(with,prep). |

Finite State
Automata

Finite State
Parsers and
Transducers

Finite State
Methods in
NLP

Finite State Parsers and Transducers

Building Structure while Recognizing

Introduction
to the
Artificial
Intelligence
methods

Finite State
Automata

Finite State
Parsers and
Transducers

Finite State
Methods in
NLP

It's not very difficult to change our recognizer to work with FSA specifications that, like the above, define their transitions in terms of categories instead of symbols and then use a lexicon to map those categories to symbols or the other way round. The only thing that changes is the definition of the `traverse` predicate. We don't simply compare the label of the arc with the next symbol of the input anymore, but have to access the lexicon to check whether the next symbol of the input is a word of the category specified by the label of the arc.

Finite State Parsers and Transducers

Finite State Transducers

Introduction
to the
Artificial
Intelligence
methods

Finite State
Automata

Finite State
Parsers and
Transducers

Finite State
Methods in
NLP

- A **Finite State Transducer** essentially is a **FSA** that works on two (or more) tapes. They read from one of the tapes and write onto the other.
- Transducers can be used in other modes than the translation mode as well: in the generation mode transducers write on both tapes and in the recognition mode they read from both tapes. Furthermore, the direction of translation can be turned around.

Finite State Parsers and Transducers

Finite State Transducers

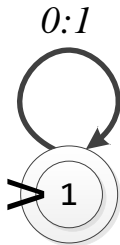
Introduction
to the
Artificial
Intelligence
methods

Finite State
Automata

Finite State
Parsers and
Transducers

Finite State
Methods in
NLP

This is a transducer that translates 0s into 1s (or vice versa):



Finite State Parsers and Transducers

Finite State Transducers

Introduction
to the
Artificial
Intelligence
methods

Finite State
Automata

Finite State
Parsers and
Transducers

Finite State
Methods in
NLP

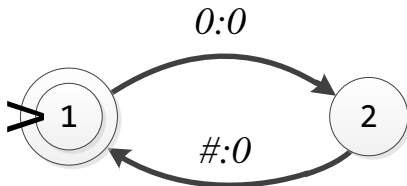
The above transducer behaves as follows in the different modes.

- **Generation mode:** It writes a string of zeros on one tape and a string ones on the other tape. Both strings have the same length.
- **Recognition mode:** It accepts when the word on the first tape consists of exactly as many zeros as the word on the second tape consists of 1s.
- **Translation mode (left to right):** It reads 0s from the first tape and writes an 1 for every 0 that it reads onto the second tape.
- **Translation mode (right to left):** It reads 1s from the second tape and writes a 0 for every 1 that it reads onto the first tape.

Finite State Parsers and Transducers

Finite State Transducers

Transitions in transducers can make jumps going from one state to another without doing anything on either one or on both of the tapes. So, transitions of the form $0:\#$ or $\#:0$ or $\#:\#$ are possible.



What does this transducer do?

- Generation mode: It writes twice as many 0s onto the second tape as onto the first one.
- Recognition mode: It accepts when the second tape has twice as many 0s as the first one.
- Translation mode (left to right): It reads 0s from the first tape and writes twice as many onto the second tape.
- Translation mode (right to left): It reads 0s from the second tape and writes half as many onto the first one.

Finite State Parsers and Transducers

Finite State Transducers

Introduction
to the
Artificial
Intelligence
methods

Finite State
Automata

Finite State
Parsers and
Transducers

Finite State
Methods in
NLP

In implementing FST in Prolog, we represent it as a static data structure that other programs manipulate. Note that to be able to write 0:1 as the label of the arc, we have to define colon as an infix operator as is done by the operator definition.

```
:- op(250,xfx,:).
```

```
start(1).
```

```
end(1).
```

```
arc(1,1,0:1).
```

Finite State Parsers and Transducers

Finite State Transducers

Introduction
to the
Artificial
Intelligence
methods

Finite State
Automata

Finite State
Parsers and
Transducers

Finite State
Methods in
NLP

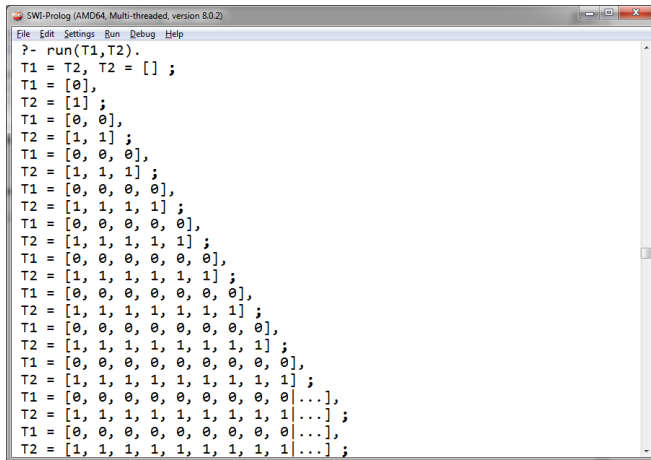
Now, we need a program that can manipulate these data structures and carry out the transduction.

```
traverse(S1:S2, [S1|Tape1], Tape1, [S2|Tape2], Tape2).  
transduce(Node, [], []) :- end(Node).  
transduce(N, T1, T2) :- arc(N, Nx, St),  
                        traverse(St, T1, NT1, T2, NT2),  
                        transduce(Nx, NT1, NT2).  
  
run(T1, T2) :- start(N), transduce(N, T1, T2).
```

Finite State Parsers and Transducers

Finite State Transducers

Executing program with the first, second or both parameters undefined we can run the transducer in various modes.



```
SWI-Prolog (AMD64, Multi-threaded, version 8.0.2)
File Edit Settings Run Debug Help
?- run(T1,T2).
T1 = T2, T2 = [] ;
T1 = [0],
T2 = [1] ;
T1 = [0, 0],
T2 = [1, 1] ;
T1 = [0, 0, 0],
T2 = [1, 1, 1] ;
T1 = [0, 0, 0, 0],
T2 = [1, 1, 1, 1] ;
T1 = [0, 0, 0, 0, 0],
T2 = [1, 1, 1, 1, 1] ;
T1 = [0, 0, 0, 0, 0, 0],
T2 = [1, 1, 1, 1, 1, 1] ;
T1 = [0, 0, 0, 0, 0, 0, 0],
T2 = [1, 1, 1, 1, 1, 1, 1] ;
T1 = [0, 0, 0, 0, 0, 0, 0, 0],
T2 = [1, 1, 1, 1, 1, 1, 1, 1] ;
T1 = [0, 0, 0, 0, 0, 0, 0, 0, 0],
T2 = [1, 1, 1, 1, 1, 1, 1, 1, 1] ;
T1 = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
T2 = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1] ;
T1 = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...],
T2 = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...] ;
T1 = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...],
T2 = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...] ;
```

Finite State Parsers and Transducers

Finite State Transducers

Introduction
to the
Artificial
Intelligence
methods

Finite State
Automata

Finite State
Parsers and
Transducers

Finite State
Methods in
NLP

To be able to use the second transducer, the zero doubler we need a program that can handle transitions involving jumps. The only thing that changes is the way that the tapes are treated when making a transition. This is taken care of by the `traverse1/4` predicate and this is all we have to adapt.

```
traverse1('#':S2,Tape1,Tape1,[S2|Tape2],Tape2).
```

```
traverse1(S1:S2,[S1|Tape1],Tape1,[S2|Tape2],Tape2).
```

Finite State Parsers and Transducers

Finite State Transducers

And this is how it works.

[illegible]

Finite State Parsers and Transducers

Morphological Analysis with Finite State Transducers

Introduction
to the
Artificial
Intelligence
methods

Finite State
Automata

Finite State
Parsers and
Transducers

Finite State
Methods in
NLP

- Morphology is about the internal structure of words.
- It asks of what are the building blocks – the morphemes – that a word is constructed from and what is the meaning of these blocks. And what are the rules by which these building blocks can be combined to form words?
- For instance the word "*share*" consists of one morpheme *share*, while "*shared*" consists of two morphemes, namely the **stem** *share* and the past tense *d*. The word "*blessed*" also consists of two morphemes, but now the past tense is *ed*.
- Morphology is an area of computational linguistics where finite state technology has been found to be particularly useful, because for many languages the rules after which morphemes can be combined to build words can be captured by finite state automata.
- It is possible to write finite state transducers that map the surface form of a word to a description of the morphemes that constitute that word or vice versa.

Finite State Parsers and Transducers

Morphological Analysis with Finite State Transducers

Introduction
to the
Artificial
Intelligence
methods

Finite State
Automata

Finite State
Parsers and
Transducers

Finite State
Methods in
NLP

- Such transducers could, for instance, map *share+d* and *bless+ed* to *share+PAST* and *bless+PAST* respectively.
- As an example we will look at past forms of verbs in English.
- The default rule is to add *ed* or *d* to a normal form.
- But there also are the irregular verbs, such as *run*, *teach*, *think* etc.

Finite State Parsers and Transducers

Morphological Analysis with Finite State Transducers

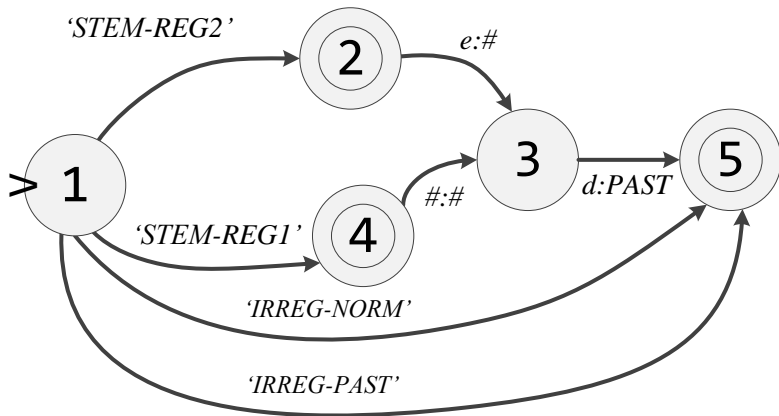
Introduction
to the
Artificial
Intelligence
methods

Finite State
Automata

Finite State
Parsers and
Transducers

Finite State
Methods in
NLP

We want a transducer that would translate *shared* to *share-PAST*, *blessed* to *bless-PAST*, *thought* to *think-PAST* and so on.



Finite State Parsers and Transducers

Morphological Analysis with Finite State Transducers

Introduction
to the
Artificial
Intelligence
methods

Finite State
Automata

Finite State
Parsers and
Transducers

Finite State
Methods in
NLP

As before, we split our lexicon into categories.

```
lex(bless:bless, 'STEM-REG2').  
lex(share:share, 'STEM-REG1').  
lex(run:run, 'IRREG-NORM').  
lex(think:think, 'IRREG-NORM').  
lex(ran:'run-PAST', 'IRREG-PAST').  
lex(thought:'think-PAST', 'IRREG-PAST').  
...
```

Finite State Parsers and Transducers

Morphological Analysis with Finite State Transducers

Introduction
to the
Artificial
Intelligence
methods

Finite State
Automata

Finite State
Parsers and
Transducers

Finite State
Methods in
NLP

- The above transducer assumes that the words are already split up into morphemes (such as stem and the past tense suffix).
- If it was implemented in Prolog it would assume input of the form `[share,d]` on the first tape.
- In order to be able to input plain strings we should use another transducer that would split the strings for us, and then use these two transducers in a cascade.
- That is, we let the morphological transducer run on the output of the splitting transducer, and compose them into a single transducer that performs both tasks.

Finite State Methods in NLP

Finite State Methods in NLP

Introduction
to the
Artificial
Intelligence
methods

Finite State
Automata

Finite State
Parsers and
Transducers

Finite State
Methods in
NLP

- 1 Morphology
- 2 Morph parsing
- 3 Regular relation and languages

Finite State Methods in NLP

Morphology

Introduction
to the
Artificial
Intelligence
methods

Finite State
Automata

Finite State
Parsers and
Transducers

Finite State
Methods in
NLP

- Morphology is interested in what are the smallest units in word that bear some meaning and how can they be combined to form words.
- The smallest unit in a word that bear some meaning are called **morphemes**.
- Morphemes that contribute the main meaning of a word are called **stems**, while the other morphemes are known as **affixes**.
- How can morphemes be combined to form words that are legal in some language ?

Finite State Methods in NLP

Morph parsing

Introduction
to the
Artificial
Intelligence
methods

Finite State
Automata

Finite State
Parsers and
Transducers

Finite State
Methods in
NLP

The goal of **morphological parsing** is to find out what morphemes a given word is built from.

| | | |
|--------|------|---------|
| car | car | Noun SG |
| cars | car | Noun PL |
| foot | foot | N SG |
| feet | foot | Noun PL |
| bosses | boss | N PL |

Finite State Methods in NLP

Morph parsing

Introduction
to the
Artificial
Intelligence
methods

Finite State
Automata

Finite State
Parsers and
Transducers

Finite State
Methods in
NLP

Morphological parsing yields information that, for instance helps to know the agreement features of words. Grammar checkers need to know agreement information to detect mistakes. Morphological information also helps spell checkers to decide whether something is a possible word or not, and in information retrieval it is used to search not only one form of a word, but also all other possible forms.

Finite State Methods in NLP

Morph parsing

Introduction
to the
Artificial
Intelligence
methods

Finite State
Automata

Finite State
Parsers and
Transducers

Finite State
Methods in
NLP

To get morphological information about a word, given it's surface form, we need to proceed in some steps.

- Split a word up into it's possible components and indicate morpheme boundaries. For example, we are going to make *car+s* out of *cars*. There may be multiple ways of splitting up a word.
- Use a lexicon of stems to look up the categories of the stems and the meaning of the affixes. So *boss+s* will be mapped to *boss Noun PL*, and *feet* to *foot Noun PL*.

Finite State Methods in NLP

Morph parsing

Introduction
to the
Artificial
Intelligence
methods

Finite State
Automata

Finite State
Parsers and
Transducers

Finite State
Methods in
NLP

| Surface form | Intermediate form | Morphological Structure |
|--------------|-------------------|-------------------------|
| car | car | car Noun SG |
| cars | car+s | car Noun PL |
| foot | foot | foot N SG |
| feet | feet | foot Noun PL |
| bosses | boss+s | boss N PL |

Finite State Methods in NLP

Morph parsing

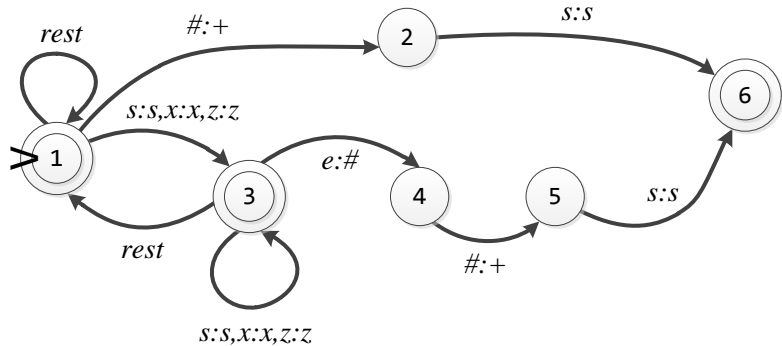
Introduction
to the
Artificial
Intelligence
methods

Finite State
Automata

Finite State
Parsers and
Transducers

Finite State
Methods in
NLP

From the surface form to the intermediate form.



From the Intermediate Form to the Morphological Structure.

The input that this transducer has to accept is of one of the following:

- 1 Regular noun stem (*car, boss, etc.*). Map all symbols of the stem to themselves and then output **N** and **SG**.
- 2 Regular noun stem + s (*car+s, boss+s, etc.*). Map all symbols of the stem to themselves, and then output **N** and replace s with **PL**.
- 3 Singular irregular noun stem (*foot, woman, etc.*). Do the same as in the first case.
- 4 Plural irregular noun stem (*feet*). Map the irregular plural noun stem to the corresponding singular stem, and then add **N** and **PL**.

Finite State Methods in NLP

Morph parsing

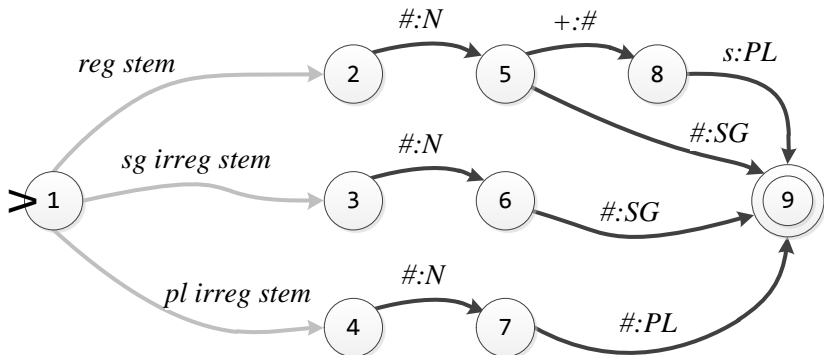
Introduction
to the
Artificial
Intelligence
methods

Finite State
Automata

Finite State
Parsers and
Transducers

Finite State
Methods in
NLP

From the Intermediate Form to the Morphological Structure.



Finite State Methods in NLP

Morph parsing

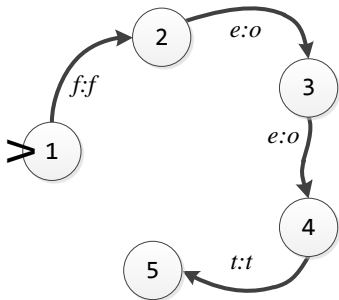
Introduction
to the
Artificial
Intelligence
methods

Finite State
Automata

Finite State
Parsers and
Transducers

Finite State
Methods in
NLP

- What still needs to be specified is how exactly the parts between state 1 and states 2,3, and 4 respectively look like.
- We need to recognize noun stems and decide whether they are regular or not.
- It is done by encoding a lexicon.



Finite State Methods in NLP

Morph parsing

Introduction
to the
Artificial
Intelligence
methods

Finite State
Automata

Finite State
Parsers and
Transducers

Finite State
Methods in
NLP

- If we let the two transducers run in a cascade, we can do a morphological parse of (some) English noun phrases.
- We can also use this transducer for generating a surface form from an underlying form.
- There are algorithms for combining several cascaded transducers or multiple transducers that are supposed to be applied in parallel into a single transducer. However, these algorithms only work, if the individual transducers obey some restrictions so that some care has to be taken when specifying them.

Finite State Methods in NLP

Regular relations and languages

Introduction
to the
Artificial
Intelligence
methods

Finite State
Automata

Finite State
Parsers and
Transducers

Finite State
Methods in
NLP

- As we have seen before, we don't always have to specify one big transducer that can deal with all spelling rules, but that it is enough to specify one small transducer per rule, because there are ways of combining these individual transducers into one big transducer.
- Still you might think that specifying the e-insertion transducer was actually quite tricky.
- But there is way of formulating spelling rules that makes it possible to automatically translate them into transducers.

- The languages that FSA can recognize are called **regular**.
- Every Regular Language can be specified with a **Regular Expression**.
 - 1 Let us have an **alphabet** $\Sigma = (c_1, c_2, \dots, c_k)$. For every i c_i is a regular expression.
 - 2 ε is a regular expression (an empty word).
 - 3 If α_1 and α_2 are regular expressions then $(\alpha_1|\alpha_2)$ and $(\alpha_1)(\alpha_2)$ are regular expressions.
 - 4 If α is a regular expression then $(\alpha)^*$ is a regular expression.
- There is a systematic way of translating regular expressions into finite state automata.

Finite State Methods in NLP

Regular relations and languages

Introduction
to the
Artificial
Intelligence
methods

Finite State
Automata

Finite State
Parsers and
Transducers

Finite State
Methods in
NLP

- Finite state transducers recognize tuples of strings.
- A set of tuples of strings that can be recognized by an FST is called a **Regular Relation**.
- Language spelling rules can be specified as regular relations.
- There are algorithms for translating such rule systems (at least, as long as they obey certain restrictions) into transducers.