Natuurlijke Taalverwerking II 2006/2007

Gosse Bouma

www.let.rug.nl/~gosse/ntv2

Overzicht

- Relatie tussen Reg Ex en Automaten
- (Non-)determinisme
- Transducers
- Operaties op Transducers

Regular Expressions

[A,B]	A followed by B
{A,B}	A or B
[A,B^]	An A optionally followed by a B
A*	zero or more occurrences of A
A+	one or more occurrences of A
?	Any symbol
'0'..'9'	Symbol in the range of '0' .. '9'
$ A	A string containing A

More Regular Expressions

~ A	A string not matching A
A - B	A string matching A but not B
A & B	A string matching A and B

Examples

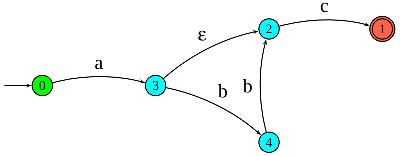
`[? *, i,s,h]`	A string with suffix **ish**
`$ [q,u]`	A string containing **qu**
`’0’..’9’`	a digit
`’0’..’9’ - ’2’`	all digits except 2
`~ ’0’ ...’9’`	not a digit (i.e. includes a, 10, ϵ)
`$ a & $ b`	strings containing an a and a b

From Reg Ex to FSA

- Every Reg Ex corresponds with a FS automaton
- Every Reg Ex operator defines an operation on FS automata

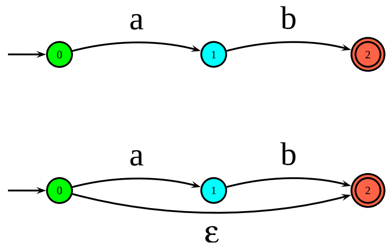
Epsilon

- Epsilon transition’s (jumps) allow transition from one state to another without reading any input symbol



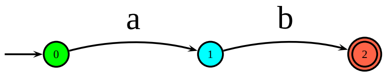
Optional

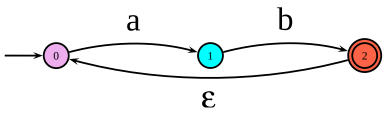
`[a,b]^`



Kleene Closure

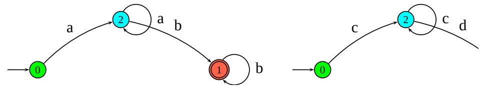
[a,b] *

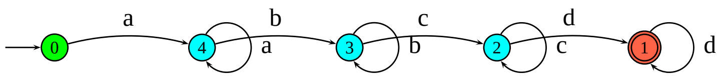




Concatenation

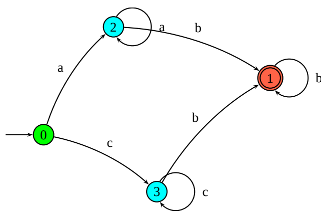
[[a+,b+], [c+,d+]]



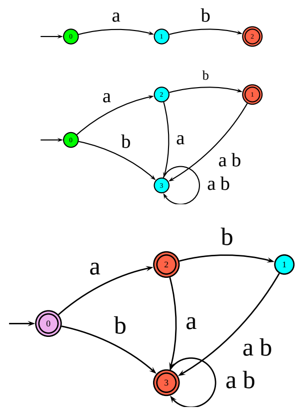


Union

{[a+,b+], [c+,b+]}



Complement

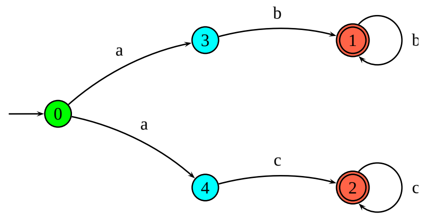


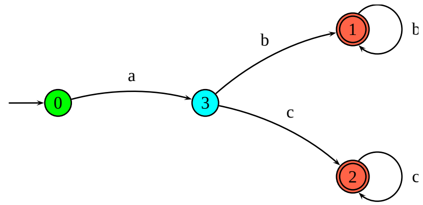
- Input automaton must be deterministic

Deterministic Recognizer

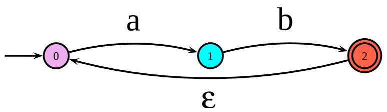
- A FS recognizer is deterministic iff
  - it has a single start state,
  - it has no epsilon transitions,
  - for each state and each symbol there is at most one applicable transition.
- For every M there is a deterministic (efficient) automaton M' such that $L(M) = L(M')$.

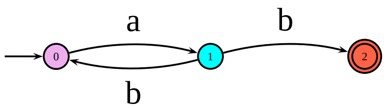
Removing Non-deterministic Transitions





Removing Epsilons





Converting NFA to DFA

www.cs.may.ie/~jpower/Courses/parsing/

We use a Deterministic Finite-State Automaton (DFA) which is a special case of a NFA with the additional requirements that:

- There are no transitions involving ϵ,
- No state has two outgoing transitions based on the same symbol .

Subset Construction Algorithm

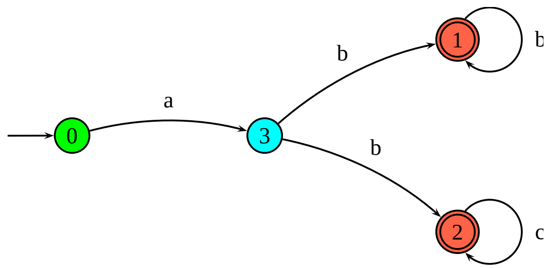
- The ϵ-closure function takes a state and returns the set of states reachable from it based on (one or more) ϵ-transitions.
- The function move takes a state and a character, and returns the set of states reachable by one transition on this character.

$$move(\{A,B\},a) = move(A,a) \cup move(B,a)$$

The Subset Construction Algorithm II

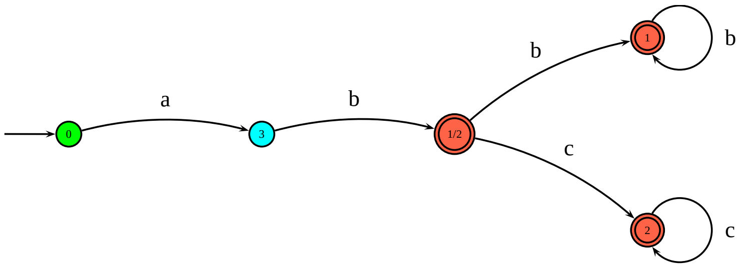
1. DFA start state = ϵ-closure(NFA start state).
2. For each new DFA-state S and possible input symbol a:
   - Add the transition (S,a,ϵ-closure(move(S,a)))
3. Apply step 2 to newly added states.
4. DFA finish states = states containing a NFA finish state.

Example



$$\{(0,a,3),(3,b,1),(3,b,2),(1,b,1),(2,c,2)\}$$
$$\Downarrow$$
$$\{(0,a,3),(3,b,\{1,2\}),(\{1,2\},b,1),$$
$$(\{1,2\},c,2),(1,b,1),(2,c,2)\}$$

DFA



Intermezzo: RegEx without Kleene *

- Automata for languages definable without Kleene * or + have interesting properties (Yli Jyrä, EACL 2003)
- Can you define the language a* without using Kleene *, +, or $

Intermezzo: RegEx without Kleene *

- Can you define the language a* without using Kleene *, +, or $
- `~[{[],~[]},? -a,{[],~[]}]`

Recognizers vs Transducers

- A finite state recognizer is an automaton which accepts strings (yes/no decisions):
  - ⋆ recognize Zip Codes, Proper Names, Syllables, ...
- A finite state transducer is an automaton which maps one string onto another string:
  - ⋆ Map Letters onto Phonemes, Inflected words onto Base Forms, Words onto Part of Speech Tags, ...

Stemming

- Translate a word into its base form,
- For information retrieval:
  - ⋆ Given a query, find relevant documents
  - ⋆ A query with republican, can lead to a document with republicans.

Stemming

`Georgia`	`georgia`
`Republicans`	`republican`
`are`	`be`
`getting`	`get`
`strong`	`strong`
`encouragement`	`encouragement`
`to`	`to`
`enter`	`enter`
`a`	`a`
`candidate`	`candidate`

Part of Speech Tagging

- Translate a sequence of words into a sequence of Part of Speech Tags
- Useful as a first step towards full parsing or to support searching for linguistic patterns,

Part of Speech Tagging

`AT1`	`a`
`JJ`	`relative`
`NN1c`	`handful`
`IO`	`of`
`DAz`	`such`
`NN2`	`reports`
`VBDZ`	`was`
`VVNv`	`received`

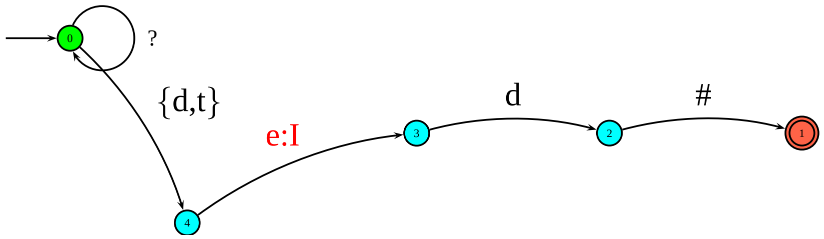
Grapheme to Phoneme Conversion

- Translate a sequence of letters into a sequence of phonemes
- Required for Text to Speech applications
- Each letter or sequence of letters is translated into a phoneme

a	b	b	r	e	v	i	a	t	e	d
$\Downarrow$	$\Downarrow$	$\Downarrow$	$\Downarrow$	$\Downarrow$	$\Downarrow$	$\Downarrow$	$\Downarrow$	$\Downarrow$	$\Downarrow$	$\Downarrow$
@	b	ϵ	r	i	v	I	1	t	I	d

Encoding a Rule

- e → I / {t,d} _ d #



- abbreviated# → abbreviatId#

Regex Notation for Transducers

- [a:b, c*] translates, among others, accc in bccc.
- : is the 'pair'-operator: it translates a symbol A in a symbol B.

Regex Notation for Transducers

- [a:b, c*] is short for [a:b, (c:c)*]
- By default, a regular expression without ':' is read as the identity-transducer: every symbol in the input is mapped onto itself.

Dutch Dimunitives

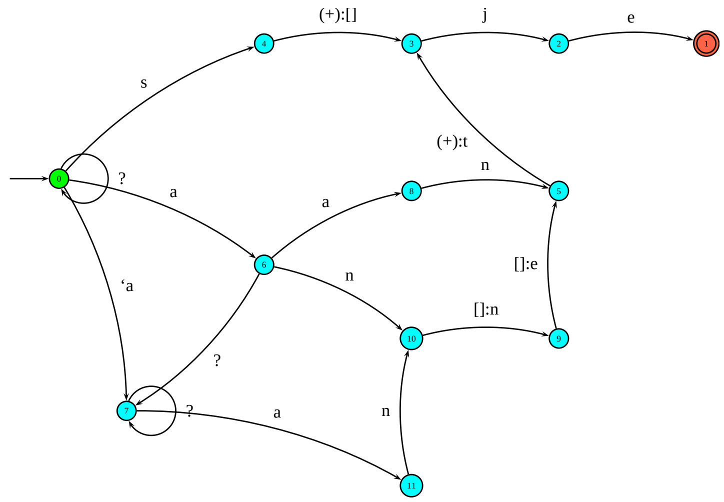
huis+je → huisje
haan+je → haantje
man+je → mannetje

input	h	u	i	s	+	j	e	
output	h	u	i	s	ϵ	j	e	
input	h	a	a	n	+	j	e	
output	h	a	a	n	t	j	e	
input	m	a	n	ϵ	ϵ	+	j	e
output	m	a	n	n	e	t	j	e

Dutch Dimunitives

```
[? *,{[s,+ :[]],
      [a,a,n,+ :t],
      [~a,a,n,[]:n,[]:e,+ :t]
     },
 j,e
]
```

Dutch Dimunitives

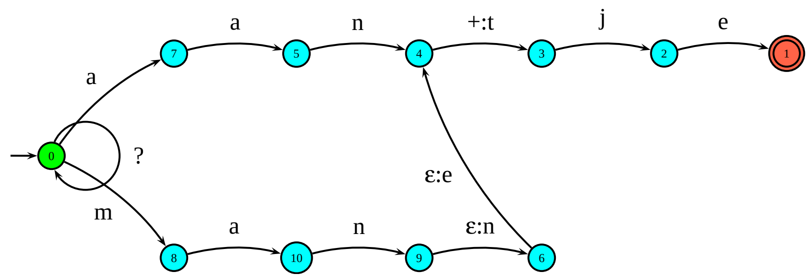


(Non-)determinism

- An transducer is deterministic if for every state and inputsymbol, at most a single transduction to a new state is possible.
- Non-deterministic transducers can sometimes be made deterministic, but not always.
- Non-deterministic recognizers can always be made deterministic.

Non-Determinism: Example

maan+je → maantje
man+je → mannetje



Two Sources of Non-determinism

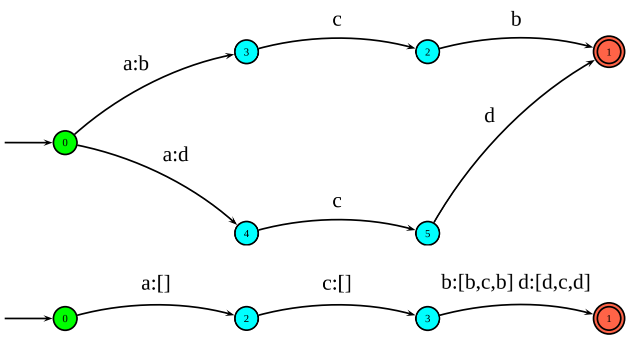
- Unbounded Look-ahead
  - ⋆ acccb → bcccb acccd → dcccd
  - ⋆ `{[a:b, c*, b], [a:d, c*, d]}`
- Multiple outputs
  - ⋆ bloem+je → bloempje
  - ⋆ bloem+je → bloemetje
  - ⋆ `[?*, o, e, m, {+:p, +:[e,t,]}, j, e]`

Deterministic Transducers

- Deterministic transducers are more efficient than non-deterministic transducers (because no choice-points/backtracking/search is required).
- But deterministic transducers can be much larger than corresponding non-deterministic transducer.
- (`t_determinize` option in FSA).

Making a Transducer Deterministic

acb → bcb
acd → dcd



From English to Dutch Numbers

- Automatic translation of (spoken) English into Dutch requires translation of number words,
- twentyone → eenentwintig,
- twentyone → 21 → eenentwintig

From Number Words to Numbers

```
macro(one, {one:1, two:2, ...,
                         nine:9 } ).
macro(twenty, {twenty:2,thirty:3,...,
                         ninety:9 } ).
macro(eng2num,{ one,ten:[1,0],
                eleven:[1,1],...,
                nineteen:[1,9],
                [twenty,one]      } ).
```

From English to Dutch Numbers

- Transducer T1 for translating English Number Words into Numbers,
- Transducer T2 for translating Numbers into Dutch Number Words
- The output of T1 is used as input by T2.

Composition

- The composition of transducers T1 and T2 is a new transducer T3, which is equivalent to passing the input through T1, taking the output of T1 as input for T2, and taking the output of T2 as output.
- T1 o T2 denotes the composition of T1 and T2.

Number Translation by Composition

```
macro(eng2num,
  {{one,ten:[1,0],..}).
macro(num2dut,
  {1:een,2:twee, ....}).
macro(eng2dut,
  eng2num o num2dut).
```

Input/Output reversal

- The inverse of a transducer T is a transducer which takes as input the output of T, and produces as output the input of T.
- In FSA: `inverse(T)`.
- Translating Dutch into English:

```
macro(dutch2eng,
  inverse(num2dut) o inverse(eng2num) ).
macro(dutch2eng,
  inverse(eng2dut) ).
```

Finite State POS Tagging

- Assign Part of Speech tags to words,
- but many words have more than one POS:
  - The/det report/n was/aux written/v
  - The/det police/n has/aux to/aux report/v all/det problems/n

Finite State POS Tagging

- A Solution:
  - A non-deterministic T which assigns a word all possible POS tags,
  - Recognizers R which filter the output of T,
  - Compose T and (the identity transducer for) R.

Finite State POS Tagging

```
macro(lexicon,
  { all:det,has:aux,police:n,problems:n,
    report:{v,n},the:det,to:v,was:aux,
    written:v}* ).
macro(no_det_v,
 ~ $ [ det, v ] ).
macro(tagger,
  lexicon o no_det_v ).
```