Finite-State Technology

Torbjörn Lager,
Göteborg University

Part 1

In this talk:
- Regular languages and relations
- Finite-state automata and transducers
- Regular expressions
- Finite-state component technologies
- Applications of finite-state technology
- The Xerox Finite-State Tool (XFST)

Languages
- Sets of strings
- Example:
  \{"ac" "abc" "abbc" "abbbc" ...\}

Finite-State Automata
- Directed graphs consisting of states, and arcs labelled with symbols. For example:

\[
\begin{array}{c}
\text{a} \\
\text{b} \\
\text{c} \\
\text{o}
\end{array}
\]

Regular expressions
- Example:
  \[a b^* c]\]
- Note:
  - Atomic symbols (e.g. 'a') denote languages (e.g. \{"a"\})
  - Operations (here: concatenation and kleene-star) are operations over languages

Regular expressions, regular languages and automata
- Regular expressions denote languages
- Compile into regular languages
- Generate accept finite-state automata
Regular expressions, regular languages and automata

\[ [a \ b^* c] \]

denotes

\{<"ac" "abc" "abbc" ...>\}

generates

accepts

Finite-State Transducers

- Directed graphs consisting of states, and arcs labelled with pairs of symbols. For example:

```
0 -----> 1 -----> 2
|    ^    |
|    \    |
|     \   |
|      \  |
|       \|
|        a
|        ^
|        x
|        c
```

Regular relations

- Sets of pairs of strings
- Example:
  \{<"ac" "ac">, <"abc" "axc">, <"abbc" "axxc"> ...\}

A regular relation has
- an upper language (e.g. \{"ac" "abc" "abbc" ...\})
- a lower language (e.g. \{"ac" "axc" "axxc" ...\})

Regular expressions

- Example:
  \[ [a \ [b \ x \ x]^* c] \]

Note:
- Here, \(a\) denotes a regular relation \{<"a" "a">\}
- Here, concatenation and iteration are operations over regular relations

Regular expressions, regular relations, and transducers

Regular expressions

- denote
- compile into

Regular relations

- generate
- accept

Finite-state transducers

- generates
- accepts

Regular expressions, regular relations, and transducers

- denoted
- compiled into

\{<"ac" "ac">, <"abc" "axc">, <"abbc" "axxc"> ...\}

- generates
- accepts
A FSA is a kind of FST

Regular expression operators
- Concatenation: $A \cdot B$
- Union: $A \cup B$
- Iteration (Kleene-star): $A^*$
- Difference: $A - B$
- Intersection: $A \cap B$
- Grouping of expressions: $[A]$

Examples
- $a b$ denotes \{"ab"\}
- $a [b|c]$ denotes \{"ab", "ac"\}
- $a* [b|c]*$ denotes \{"", "a", "ab"", ...\}
- $[a|b] \& [b|c]$ denotes \{"b"\}
- $[a|b] - b$ denotes \{"a"\}
- $[a|b] - [b|a]$ denotes \{

Special symbols
- ? The any symbol
- ?* The universal language
- [] The empty-string language
- 0 (or "") The empty string (epsilon)
- .# Beginning/end of string
- % The escape character

A warning
- Be aware of the difference between uni-character symbols and multi-character symbols!
- Examples:
  - $[a b | a b]$ denotes \{"ab"\}
  - $[a b | a b]$ denotes \{"ab", "ab"\}
  - $[a b c] \& $a$ denotes \{"abc"\}
  - $[abc] \& $a$ denotes \{

Regular expression operators
- Optionality: $(A)$
- Kleene-plus: $A^+$
- Complement: $\neg A$
- Containment: $\leq A$
- Restriction: $A \Rightarrow B \_ C$
Examples

- a (b) c  \{“ac” “abc”\}
- a b+ c  \{“abc” “abbc” “abbbc” ...
- ~[a b c]  \{”” ... “a” ... “ab” ... “abca” ...
- $[a|b]$  \{“a” ... “abba” ... “abcd” ...
- b => a c  \{“” ... “a” ... “ccc” ... “abc” ...

Regular expression operators

- Crossproduct  A .x. B
- Composition  A .o. B

Examples

- [a|b] .x. [c|d]  \{<“a” : “c”> <“a” : “d”> <“b” : “c”> <“b” : “d”>\}
- [a .x. b] .o. [b .x. c]  \{<“a” : “c”>\}

Regular expression operators

- Replacement  A -> B
- Cond. replacement  A -> B || C _ D
- Parallel replacement  A1 -> B1, A2 -> B2
- Directed replacement  A @-> B

Replacement example

- [a | a b ] -> x
  denotes ...
  (...) <“ab” : “xb”> ... <“ab” : “x”> ... <“x” : “x”> ... <“b” : “b”> ...
  Note that pairs such as <“a” : “a”> or <“ab” : “ab”> are NOT in the denotation of [[a | a b ] -> x].

Directed replacement example

- [a | a b ] @-> x
  denotes ...
  (...) <“ab” : “x”> ... <“x” : “x”> ... <“b” : “b”> ...
  Note that pairs such as <“a” : “a”>, <“ab” : “ab”> or <“ab” : “xb”> are NOT in the denotation of [[a | a b ] @-> x].
Defining extended operators

- \( A \Rightarrow B \bowtie C = \sim \sim [ \sim R B A \sim R ] | \sim R [ A \sim R C \sim R ] ] ] \)
- \( A \rightarrow B = [\sim [A \sim R ] [A . x. B] \sim R [A \sim R ] ] ] \)

Component technologies in FST

- Word lists and lexica
- Tokenisers
- Morphological analysers
- Part-of-speech taggers
- Parsers

Applications of FST

- Named-entity recognition
- Information extraction
- Corpus linguistics
- Spelling- and grammar checking
- Speech processing applications

The Xerox Finite-State Tool

- Compiles extended regular expressions into finite-state machines (automata and transducers)
- Allows the user to display, examine and modify the machines

Regular expressions, regular relations, and transducers

XFST commands

- read regex
- print words
- print lower-words
- print upper-words
- apply down
- apply up

\([a [b . x. x] * c]\)

\(\{ <"ac":"ac"> <"abc":"axc"> ... \}\)

generated

accepts

denotes

compiles into

\([a [b . x. x] * c]\)

\(\{ <"ac":"ac"> <"abc":"axc"> ... \}\)

generated

accepts

denotes

compiles into