Finite-State Programming

Some Examples
## Finite-state “programming”

<table>
<thead>
<tr>
<th>Function</th>
<th>Function on (set of) strings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source code</td>
<td>Regular expression</td>
</tr>
<tr>
<td>Object code</td>
<td>Finite state machine</td>
</tr>
<tr>
<td>Compiler</td>
<td>Regexp compiler</td>
</tr>
<tr>
<td>Optimization of object code</td>
<td>Determinization, minimization, pruning</td>
</tr>
</tbody>
</table>
Finite-state “programming”

<table>
<thead>
<tr>
<th>Function composition</th>
<th>(Weighted) composition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Higher-order function</td>
<td>Operator</td>
</tr>
<tr>
<td>Function inversion</td>
<td>Function inversion</td>
</tr>
<tr>
<td>(available in Prolog)</td>
<td>Ops + small regexps</td>
</tr>
</tbody>
</table>

Structured programming
## Finite-state “programming”

<table>
<thead>
<tr>
<th>Parallelism</th>
<th>Apply to set of strings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nondeterminism</td>
<td>Nondeterminism</td>
</tr>
<tr>
<td>Stochasticity</td>
<td>Prob.-weighted arcs</td>
</tr>
</tbody>
</table>
Some Xerox Extensions

$ \subseteq \text{containment} \\
\Rightarrow \text{restriction} \\
\rightarrow \Rightarrow \text{replacement}

Make it easier to describe complex languages and relations without extending the formal power of finite-state systems.
Containment

\$[ab^*c]\$

“Must contain a substring that matches \textit{ab}^*\textit{c}.”

Accepts \texttt{xxxacyy}
Rejects \texttt{bcba}

\texttt{*\* [ab^*c] \*\*}

Equivalent expression

Warning: \texttt{?} in regexps means “any character at all.” But \texttt{¿} in machines means “any character not explicitly mentioned anywhere in the machine.”
Restriction

\[ a \rightarrow b \_ c \]

“Any \( a \) must be preceded by \( b \) and followed by \( c \).”

Accepts \( \text{bacbbacde} \)
Rejects \( \text{baca} \)

\[ \sim [\sim [\text{[?* b]} \ a \ ?*]] \ & \sim [\text{[?* a} \sim [\text{c \ ?*}]] \]

Equivalent expression
Replacement

\[ a \ b \rightarrow b \ a \]

“Replace ‘ab’ by ‘ba’.”

Transduces \underline{abcdbaba} to \underline{bacdbbbaa}

\[
\left[ \sim\$[a \ b] \ [ [a \ b] \ .x. \ [b \ a]] \right]^* \sim\$[a \ b]
\]

Equivalent expression

slide courtesy of L. Karttunen (modified)
Replacement is Nondeterministic

\[ a \ b \rightarrow b \ a \mid x \]

"Replace ‘ab’ by ‘ba’ or ‘x’, nondeterministically."

Transduces \underline{abcdbaba} to \{\underline{bacdbbaa}, \underline{bacdbxa}, \underline{xcdbbbaa}, \underline{xcdbxaxa}\}
Replacement is Nondeterministic

\[
[ \text{a b} \rightarrow \text{b a} | \text{x} ] \cdot \circ \cdot [ \text{x} \rightarrow \_ \text{c} ]
\]

“Replace ‘ab’ by ‘ba’ or ‘x’, nondeterministically.”

Transduces \underline{abcdbaba} to \{\underline{bacdbbaa}, \underline{bacdbxa}, \underline{xcdbbaa}, \underline{xcdbxa}\}
Replacement is Nondeterministic

```
ab | b | ba | aba -> x
```

applied to “aba”

Four overlapping substrings match; we haven’t told it which one to replace so it chooses nondeterministically

```
ad
aba
aba
aba
ab
```

slide courtesy of L. Karttunen (modified)
More Replace Operators

- Optional replacement: \( a \ b \ (\rightarrow) \ b \ a \)

- Directed replacement
  - guarantees a unique result by constraining the factorization of the input string by
    - Direction of the match (rightward or leftward)
    - Length (longest or shortest)
@-> Left-to-right, Longest-match Replacement

\[
\begin{array}{c|c|c|c|c}
\text{a} & \text{b} & \text{a} & \text{b} & \text{a} \\
\text{a} & \text{x} & \text{a} & \text{a} & \text{x} \\
\text{a} & \text{x} & \text{a} & \text{a} & \text{x} \\
\text{a} & \text{x} & \text{a} & \text{a} & \text{x} \\
\end{array}
\]

applied to “aba”

@-> left-to-right, longest match
@> left-to-right, shortest match
->@ right-to-left, longest match
>@ right-to-left, shortest match
Using “…” for marking

\[ a|e|i|o|u \rightarrow [ \ldots ] \]

\[ pota\_to \]
\[ p[o]t[a]t[o] \]

Note: actually have to write as \rightarrow %[ \ldots %] or \rightarrow "[" \ldots "] since [] are parens in the regexp language
Using “…” for marking

\[
\begin{align*}
\text{a|e|i|o|u} & \rightarrow [ \ldots ] \\
p\ o\ t\ a\ t\ o\ & \rightarrow p[o]t[a]t[o]
\end{align*}
\]

Which way does the FST transduce potatoe?

\[
\begin{align*}
p\ o\ t\ a\ t\ o\ e & \quad \text{vs.} \quad p\ o\ t\ a\ t\ o\ e \\
p[o]t[a]t[o][e] & \quad \text{vs.} \quad p[o]t[a]t[o\ e]
\end{align*}
\]

How would you change it to get the other answer?

slide courtesy of L. Karttunen (modified)
Example: Finnish Syllabification

\[
\text{define } C \ [ \ b \mid c \mid d \mid f \ldots \\
\text{define } V \ [ \ a \mid e \mid i \mid o \mid u ];
\]

\[
[C^* \ V+ \ C^*] \ @-> \ldots \ "-" \ | \ _ \ [C \ V]
\]

“Insert a hyphen after the longest instance of the \text{C* V+ C*} pattern in front of a \text{C V} pattern.”

\[
\text{strukturaleismi}
\]
\[
\text{strukturaleismi}
\]
Conditional Replacement

A -> B

Replacement

L _ R

Context

The relation that replaces A by B between L and R leaving everything else unchanged.

Sources of complexity:

- Replacements and contexts may overlap
- Alternative ways of interpreting “between left and right.”
Hand-Coded Example: Parsing Dates

Today is [Tuesday, July 25, 2000].  

Best result

Today is Tuesday, [July 25, 2000].  
Today is [Tuesday, July 25], 2000.  
Today is Tuesday, [July 25], 2000.  
Today is [Tuesday], July 25, 2000.  

Bad results

Need left-to-right, longest-match constraints.
Source code: Language of Dates

Day = Monday | Tuesday | ... | Sunday
Month = January | February | ... | December
Date = 1 | 2 | 3 | ... | 3 1
Year = %0To9 (%0To9 (%0To9 (%0To9))) - %0?*

AllDates = Day | (Day “,” ) Month “ ” Date (“,” Year)
Object code: All Dates from 1/1/1 to 12/31/9999

Actually represents 7 arcs, each labeled by a string

13 states, 96 arcs
29 760 007 date expressions

Slide courtesy of L. Karttunen
Parser for Dates

AllDates @-→ “[DT " . . . "]"

Compiles into an unambiguous transducer (23 states, 332 arcs).

Today is [DT Tuesday, July 25, 2000] because yesterday was [DT Monday] and it was [DT July 24] so tomorrow must be [DT Wednesday, July 26] and not [DT July 27] as it says on the program.
Problem of Reference

Valid dates

- Tuesday, July 25, 2000
- Tuesday, February 29, 2000
- Monday, September 16, 1996

Invalid dates

- Wednesday, April 31, 1996
- Thursday, February 29, 1900
- Tuesday, July 26, 2000
Refinement by Intersection

- MaxDays In Month
  - “31” => Jan|Mar|May|...
  - “30” => Jan|Mar|Apr|...

- LeapYears
  - Feb 29, => ...

- Valid Dates
  - Q: Why does this rule end with a comma?
  - Q: Can we write the whole rule?

- WeekdayDate
  - Q: LeapYears made use of a “divisible by 4” FSA; can we build a “divisible by 7” FSA (base-ten input)?

- AllDates

Xerox contextual restriction operator

Q: Why do these rules start with spaces? (And is it enough?)
Defining Valid Dates

\[
\text{AllDates} \quad \& \quad \text{MaxDaysInMonth} \quad \& \quad \text{LeapYears} \quad \& \quad \text{WeekdayDates} = \text{ValidDates}
\]

AllDates: 13 states, 96 arcs
29 760 007 date expressions

ValidDates: 805 states, 6472 arcs
7 307 053 date expressions
Parser for Valid and Invalid Dates

\[ \text{AllDates} - \text{ValidDates} \rightarrow \"[ID \ " \ldots \ "]" \]

\[ \text{ValidDates} \rightarrow \"[VD \ " \ldots \ "]" \]

Today is \[\text{VD Tuesday, July 25, 2000}\], a valid date, not \[\text{ID Tuesday, July 26, 2000}\], an invalid date.

Comma creates a single FST that does left-to-right longest match against either pattern.

2688 states, 20439 arcs
More Engineering Applications

- Markup
  - Dates, names, places, noun phrases; spelling/grammar errors?
  - Hyphenation
  - Informative templates for information extraction (FASTUS)
  - Word segmentation (use probabilities!)
  - Part-of-speech tagging (use probabilities – maybe!)

- Translation
  - Spelling correction / edit distance
  - Phonology, morphology: series of little fixups? constraints?
  - Speech
  - Transliteration / back-transliteration
  - Machine translation?

- Learning ...
Input: Bridgestone Sports Co. said Friday it has set up a joint venture in Taiwan with a local concern and a Japanese trading house to produce golf clubs to be shipped to Japan. The joint venture, Bridgestone Sports Taiwan Co., capitalized at 20 million new Taiwan dollars, will start production in January 1990 with ...

Output:
Relationship: TIE-UP
Entities: “Bridgestone Sports Co.”
          “A local concern”
          “A Japanese trading house”
Joint Venture Company: “Bridgestone Sports Taiwan Co.”
Amount: NT$20000000
FASTUS: Successive Markups
(details on subsequent slides)

- Tokenization
  - Multiwords
    - Basic phrases (noun groups, verb groups ...)
      - Complex phrases
        - Semantic Patterns
          - Merging different references
FASTUS: Tokenization

- Spaces, hyphens, etc.
- wouldn’t → would not
- their → them ‘s
- company. → company .
  but
  Co. → Co.
FASTUS: Multiwords

- “set up”
- “joint venture”
- “San Francisco Symphony Orchestra,” “Canadian Opera Company”

- ... use a specialized regexp to match musical groups.
- ... what kind of regexp would match company names?
FASTUS : Basic phrases

Output looks like this (no nested brackets!):
... [NG it] [VG had set_up] [NG a joint_venture] [Prep in] ...

Company Name: Bridgestone Sports Co.
Verb Group: said
Noun Group: Friday
Noun Group: it
Verb Group: had set up
Noun Group: a joint venture
Preposition: in
Location: Taiwan
Preposition: with
Noun Group: a local concern
FASTUS: Noun Groups

Build FSA to recognize phrases like
approximately 5 kg
more than 30 people
the newly elected president
the largest leftist political force
a government and commercial project

Use the FSA for left-to-right longest-match markup

What does FSA look like? See next slide ...
FASTUS: Noun Groups

Described with a kind of non-recursive CFG ...
(a regexp can include names that stand for other regexps)

NG → Pronoun | Time-NP | Date-NP
NG → (Det) (Adjs) HeadNouns

...  
Adjs → sequence of adjectives maybe with commas, conjunctions, adverbs

...  
Det → DetNP | DetNonNP
DetNP → detailed expression to match “the only five, another three, this, many, hers, all, the most ...”
FASTUS: Semantic patterns

BusinessRelationship =
  NounGroup(Company/ies) VerbGroup(Set-up)
  NounGroup(JointVenture) with
  NounGroup(Company/ies) | ...

ProductionActivity =
  VerbGroup(Produce) NounGroup(Product)

NounGroup(Company/ies) → NounGroup & ...
  is made easy by the processing done at a previous level

Use this for spotting references to put in the database.