Computational Morphology: Regular expressions

Yulia Zinova

15 February 2016 – 19 February 2016

Overview

Simple expressions

Examples

Complex Expressions

Acknowledgement: The material presented here relies heavily on the material of Chapter 2 of Karttunen 2003

Atomic expressions: Symbols

- The *epsilon* symbol **0** denotes the empty-string language or the corresponding identity relation.
- The *any* symbol **?** denotes the language of all single-symbol strings
- Any single symbol, **a**, denotes the language that consists of the corresponding string, here "a," or the identity relation on that language.
- The boundary symbol .#. designates the beginning of the string in the left context and the end of the string in the right context of a restriction or a rule-like replace expression.
- The identity relation **?** maps any symbol to itself.
- Multicharacter symbols such as `PLURAL` are also symbols, but they happen to have multicharacter print names.

Atomic expressions: Pairs

- Any pair of symbols `a:b` separated by a colon denotes the relation that consists of the corresponding ordered pair of strings, {<"a", "b">}, where **a** is the *upper symbol* and **b** is the *lower symbol* of the pair.
- The pair `?:?` denotes the relation that maps any symbol to any symbol including itself. It is an *equal-length relation*, in case of `?:?` *length*=1.

Brackets

- [A] = A
- [] = 0
- [. .] has a special meaning in replace expressions and will be discussed later
- Bracketing is optional if there i no ambiguity.
- (A) = [*A* | 0]

Iteration

- **A+** denotes the concatenation of **A** with itself one or more times, the **+** operator is called *Kleene-plus* or *sigma-plus*.
- **A*** denotes the union of **A+** with the empty string language, the ***** operator is called *Kleene-star* or *sigma-star*.
- ?∗ denotes *universal language*
- [? :?] denotes the *universal equal-length relation*

Complementation

- $\sim$**A** denotes the complement of the language A.
- The complementation operator $\sim$ is also called *negation*.
- $\sim$A $= [?^* \ - \ A]$
- **A** denotes the term complement language (the set of all single-symbol strings that are not in A.
- the \\ operator is also called *term negation*.
- \\A $= [? \ - \ A]$
- Note: A must denote a language, the complementation operation in not defined for relations.

Concatenation

- Where A and B are arbitrary regular expressions, $[A\ B]$ is the *concatenation* of A and B. The white space serves as a concatemation operator.
- Concatenation is *associative*, which means that $[\ [A\ B]\ C]{=}[A\ [B\ C]\ ]$, so inner brackets can be omitted.
- $[a\ b\ c\ d]\ =\ \{abcd\}$
- $A^\wedge n$ denotes the n-ary concatenation of A with itself: $A^\wedge 3\ =\ [aaa]$
- $A^\wedge < n$ denotes less then n concatenations of A, including the empty string.
- $A^\wedge > n$ denotes more then n concatenations of A.
- $A^\wedge\{i, k\}$ denotes from i to k concatenations of A.

Containment and ignoring

- $A = [?* A ?*]
- [A / B] denotes the language or relation obtained from A by splicing in B^* everywhere withing the strings of A.
- For example, [[a b] / x] denotes the set of strings like "xxaxxxbxxx" that distort "ab" by arbitrary insertions of "x".
- [A ./. B] denotes the language or relation obtained from A by splicing in B^* everywhere in the *inside* of the elements of A but not at the edges.
- For example, [[a b] ./. x] contains strings like "axxxb" but not "xab" or "axxbxx".

Union and Intersection

- Where A and B are arbitrary regular expressions, $[A|B]$ is the union of A and B which denotes the union of the languages denoted by A and B respectively.
- The union operator is also called disjunction.
- Write down the strings in the language
  $a \mid b \mid Charley$
- Where A and B are arbitrary regular expressions (either languages or equal-length relations), $[A\&B]$ is the intersection of A and B.
- The intersection operator is also called conjunction.
- Write down the strings in the language
  $[a \mid b \mid c \mid d \mid e] \;\&\; [d \mid e \mid f \mid g]$

Substraction

- $[A \ - \ B]$ denotes the set difference of the languages denoted by A and B (the set of all strings in A that are not in B).
- What is the language denoted by
  [dog | cat | elephant] - [elephant | horse | cow]

Crossproduct

- $[A\ .x.\ B]$ denotes a relation that pairs every string of language A with every string of language B.
- A is called the *upper* language and B is called the *lower* language.
- [?* .x. ?*] denotes the *universal relation*, the mapping from any string to any string.
- $[[A]\ :\ [B]]$ denotes the same as $[A\ .x.\ B]$.
- $[a\ .x.\ b]$ and $a : b$ are equivalent expressions.
- The operator : has very high precedence and .x. has very low precedence (lower than concatenation).
- $[c\ a\ t\ .x.\ c\ h\ a\ t] = [[c\ a\ t]\ .x.\ [c\ h\ a\ t]]$
- $[c\ a\ t\ :\ c\ h\ a\ t] = [c\ a\ [t\ :\ c]\ h\ a\ t]]$

Projection

- $A.u$ denotes the upper language of the relation A.
- $A.l$ denotes the lower language of the relation A.

Reverse and inverse

- `A.r` denotes the reverse of the language or relation A.
- if A contains `<"abc", "xy">`, `A.r` contains `<"cba", "yx">`
- `A.i` denotes the inverse of the relation A.
- if A contains `<"abc", "xy">`, `A.i` contains `<"abc", "xy">`

Composition and substitution

- `[A .o. B]` denotes the composition of the relation A with the relation B.
- if A contains the string pair $< x, \; y >$, and B contains $< y, \; z >$, `[A .o. B]` contains the string pair $< x, \; z >$
- `` `[[A], s, L] `` denotes the language or relation derived from A by substituting every symbol x in the list `L` for every occurence of the symbol **s**.
- `` `[[a -> b], b, x y z] `` denotes the same relation as `[a -> [x | y | z]`

Minimal languages

- Which languages or relations are encoded by the following expressions?
- $\sim [?^*]$

Minimal languages

- Which languages or relations are encoded by the following expressions?
- $\sim$ [?*]
- {} The *empty language* that contains no strings
- []

Minimal languages

- Which languages or relations are encoded by the following expressions?
- $\sim$ [?*]
- {} The *empty language* that contains no strings
- []
- {""} The *empty string* language
- a

Minimal languages

- Which languages or relations are encoded by the following expressions?
- $\sim$ [?*]
- {} The *empty language* that contains no strings
- []
- {""} The *empty string* language
- a
- {"a"}
- (a)

Minimal languages

- Which languages or relations are encoded by the following expressions?
- $\sim$ [?*]
- {} The *empty language* that contains no strings
- []
- {""} The *empty string* language
- a
- {"a"}
- (a)
- {"", "a"}

Iteration

- Which languages or relations are encoded by the following expressions?
- $[a^*]$

Iteration

- Which languages or relations are encoded by the following expressions?
- $[a^*]$
- $\{$"", "a", "aa", $\ldots\}$
- $[a+]$

Iteration

- Which languages or relations are encoded by the following expressions?
- $[a^*]$
- $\{$“”, “a”, “aa”, $\ldots\}$
- $[a+]$
- $\{$“a”, “aa”, $\ldots\}$
- `a 0 b`

Iteration

- Which languages or relations are encoded by the following expressions?
- $[a^*]$
- {“”, “a”, “aa”, ...}
- $[a+]$
- {“a”, “aa”, ...}
- `a 0 b`
- {“ab”}
- `a:0 b:a`

Iteration

- Which languages or relations are encoded by the following expressions?
- $[a^*]$
- $\{$“”, “a”, “aa”, ...$\}$
- $[a+]$
- $\{$“a”, “aa”, ...$\}$
- `a 0 b`
- $\{$“ab”$\}$
- `a:0 b:a`
- $\{<$“ab”,“a”$>\}$
- `a b:0`

Iteration

- Which languages or relations are encoded by the following expressions?
- [a^*]
- {“”, “a”, “aa”, ...}
- [$a+$]
- {“a”, “aa”, ...}
- `a 0 b`
- {“ab”}
- `a:0 b:a`
- {<“ab”,“a”>}
- `a b:0`
- {<“ab”,“a”>} (same relation, different network!)

Crossproduct

- `a .x. b`

Crossproduct

- `a .x. b`
- $\{<$“a”,“b”$>\}$
- `[a b] .x. c`

Crossproduct

- `a .x. b`
- {<"a","b">}
- `[a b] .x. c`
- {<"ab","c">}
- When the pairs of strings are of different length, there are different ways to encode this. Draw three different networks for the last relation.
- The **Xerox** compiler pairs the strings from left to right, symbol-by symbol, so epsilon symbols are only introduced at the right end if needed (this is an arbitrary choice).

Composition

- `a:b .o. b:c`

Composition

- `a:b .o. b:c`
- {<“a”,“c”>}
- `a:b .o. b .o. b:c`

Composition

- `a:b .o. b:c`
- $\{<$“a”,“c”$>\}$
- `a:b .o. b .o. b:c`
- $\{<$“a”,“c”$>\}$

Closure

- Regular expressions were invented as a meta-language to describe languages, but then their usage extended to relations.
- A set operation has a corresponding relation on finite-state networks only if the set of regular relations and languages is **closed** under that operation.
- Closure means that if the sets to which the operation is applied are regular, the result is also regular, that is, encodable as a finite-state network.
- The table shows the closure properties of various operations.

Closure properties

Operation	Regular Languages	Regular Relations
union	yes	yes
concatenation	yes	yes
iteration	yes	yes
reversal	yes	yes
intersection	yes	no
substraction	yes	no
complementation	yes	no
composition	not applicable	yes
inversion	not applicable	yes

Precedence

Type	Operators
Unary operations	\, *
Crossproduct	:
Prefix	~, \, $
Suffix	+, *, ^, .r, .u, .l, .i
Ignoring	/
Concatenation	(whitespace)
Boolean	| , &, -
Restriction and replacement	=>, ->
Crossproduct and composition	.x., .o.

Special symbols

- To avoid the special interpretation of a symbol, one has to prefix it with % or enclose in double quotes.
- "\n" is the newline symbol
- "\t" is the tab symbol
- Multicharacter symbols are allowed. E.g., "[Noun]" or %[Noun%] denote [Noun]
- In order to not confuse the multicharacter symbols with the concatenated symbols, it is common to surround or precede the multicharacter symbols with special characters.

Restriction

- The restriction operator is one of the two fundamental operators in the traditional two-level calculus.
- `[A => L _ R]` denotes the language in which any string from A that occurs as a substring is immediately preceded by some string from L and immediately followed by some string from R.
- `[A => L1 _ R1, L2 _ R2]` denotes the language in which every instance of A is surrounded either by strings from L1 and R1 or by strings from L2 and R2.
- The list of contexts can be arbitrarily long.
- Restrictions: all the components must denote regular languages, not relations.

Replacement

- Replacement expresions describe strings of one language in terms of how they differ from the strings of the other language.
- The family of replacement operations is specific to the Xerox regular-expression calculus.

Simple replacement

- `[A -> B]` denotes the relation in which every each string of the upper language to a string that is identical to it except that all the occurrences of A are replaced by the occurrences of a string from B.
- `[A <- B]` denotes the inverse of `[B -> A]`
- `[A (->) B]` denotes an optional replacement (the union of `[A -> B]` with the identity relation A).
- `[[. A .] -> B]` is equivalent to `[A -> B]` if the language denoted by A does not contain the empty string.
- Restriction: A and B must be regular languages, not relations.

Marking and parallel replacement

- `[A -> B ... C]` denotes a relation in which each string of the upper-side universal language is paired with all strings that are identical to the original except that every instance of A that occurs as a substring is represented by a copy that has a string from B as a prefix and a string from C as a suffix.
- `[a | e | i | o | u -> %[ ...%]]` maps `"abide"` to `"[a]b[i]d[e]"`
- `[A -> B, C -> D]` denotes the simultaneous replacement of A by B and C by D. Any number of components is allowed.

Conditional replacement (1)

- `[A -> B || L _ R]`
  Every replaced substring in the upper language is immediately preceded by an upper-side string from L and immediately followed by an upper-side string from R.
- In other words, both left and right contexts are matched in the upper-language string.
- This is the most used type of replacement.
- But sometimes other types are needed.

Conditional replacement (2)

- `[A -> B / / L _ R]`
  Every replaced substring in the upper language is immediately followed by an upper-side string from R and the lower-side replacement string is immediately preceded by a string from L.
- `[A -> B \ \ L _ R]`
  Every replaced substring in the upper language is immediately preceded by an upper-side string from L and the lower-side replacement string is immediately followed by a string from R.
- `[A -> B \ / L _ R]`
  Every lower-side replacement string is immediately preceded by a lower-side string from L and immediately followed by a lower-side string from R.
- A, B, R, and L are languages, not relations.

Parallel conditional replacement

- `[A -> B || L1 _ R1 ,, C -> D || L2 _ R2]`
  replaces A by B in the context of L1 and R1 and simultaneously C by D in the context of L2 and R2.
- Example of use: replace Roman numerals with Arabic (there is a dependence on the position of symbol, e.g., 1 can be I or X).

Directed replacement

- `[A @-> B]`
  Replacement strings are selected from left to right, priority goes to the longest.
- `[A ->@ B]`
  Replacement strings are selected from right to left, priority goes to the longest.
- `[A @> B]`
  Replacement strings are selected from left to right, priority goes to the shortest.
- `[A >@ B]`
  Replacement strings are selected from right to left, priority goes to the shortest.
- A and B are languages, not relations.

References:

Karttunen, L. (2003). Finite-state morphology.