Regular Expressions

What are Regular Expressions?

What are Regular Expressions?

- Method of Pattern Matching and search within strings
- Method of describing a type of language known as a Regular Lanages (from Formal Language Theory)

Chomsky Hierarchy

So Far in this class we have talked about several language types, which can be expressed by:

- BNF Grammars
- Regular Expressions

Chomsky Hierarchy

These are part of the Chomsky hierarchy which composes languages depending on the restrictions imposed on the types of languages:

- Type 0: Recursively Enumerable languages -> Describable by a Turing Machine
- Type 1: Context Sensitive language -> Describable by a Linear-Bounded Non-deterministic Turing Machine
- Type 2: Context-free language -> Describable by a Non-deterministic Pushdown Automaton -> Programming Languages -> BNF Grammars
- Type 3: Regular language -> Describable by a finite-state automaton -> Regular Expressions

Chomsky Hierarchy

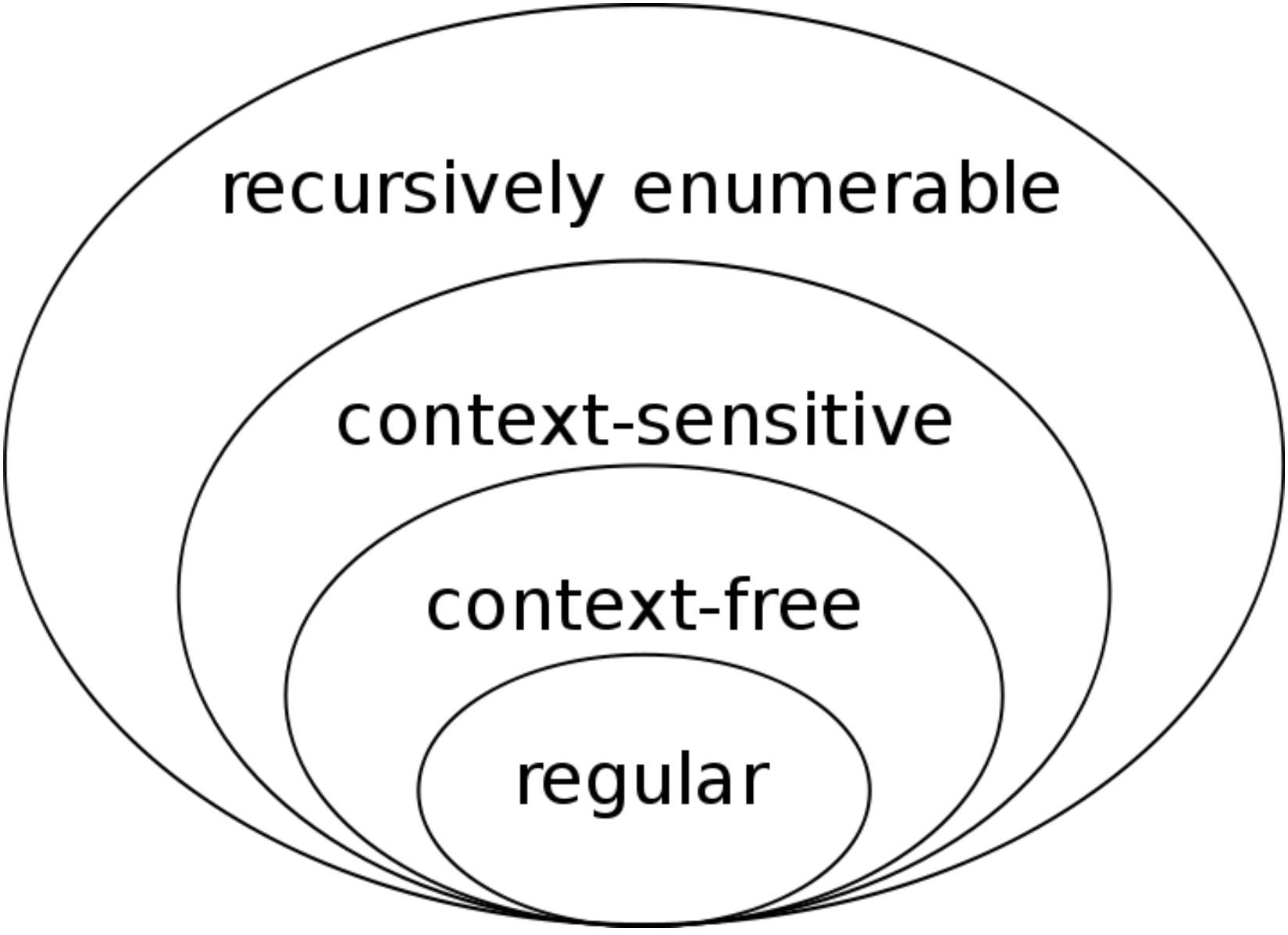

Figure 1: image

Regular Expressions

Returning to Regular Expressions, they describe a language but are typically used to identify strings of that language within the context of other strings.

That is they are used to strings matching a given pattern.

Regular Expression Character Classes

We can define a class which matches a given set of characters by placing the characters of concern in square brackets:

- `[A-Z]` The class matching A,B,. . . ,Y,Z
- `[a-zA-Z]` The class matching a,. . . ,z and A,. . . ,Z
- `[.?!]` The class matching ‘.’ ? and !

Some classes are so common that they are predefined

Predefined classes

Symbol	Description
`\w`	Matches word characters
`\W`	Matches non-word characters
`\s`	Matches whitespace, `[\t\n\r\f]`
`\S`	Matches non-whitespace
`\d`	Matches digits `[0-9]`
`\D`	Matches non-digits

Other characters and Repetition

- any character by itself e matches that character. The character `.` matches any character.
- To match a character, class, or group repeatedly you can use the following:
- re* – The star matches zero or more times
- re+ – The plus matches one or more times
- re? – The ? matches zero or one time
- re{n} – matches exactly n number of occurrences
- re{n,} – matches n or more occurrences
- re{n,m} – matches at least n and at most m occurrences
- a | b – matches a or b
- Greedy vs. Non-greedy
- `/<.*>/` - matches `<ruby>perl>`
- `/<.*?>/` - matches `<ruby>` in `<ruby>perl>`

Boundaries

We can also control matching with boundaries

- `^` - matches the beginning of a line
- `$` - matches the end of a line
- `\A` - matches beginning of a string
- `\Z` - matches end of a string
- `\b` - matches word boundaries when outside brackets, and backspace when inside brackets
- `\B` - matches non word boundaries
- `\1 ... \9` - matches nth grouped subexpression

Expressions can be grouped into sets of sub-expressions with by placing sections in parentheses

Lets see some examples