NTNU

Norwegian University of Science and Technology

Regular expressions, Backus Naur and Compilers

Advanced Simulation 2017 IKP - Marius Reed

11th December

NTNU

Regular expressions

A regular expression is a sequence of characters that defines a search pattern.

- Applications
  - Search
  - Validate
  - Replace
  - Reformat

Single character

Symbol = a

Regex = a

a

A computer once beat me at chess, but it was no match for me at kick boxing. - *Emo Philips*

Dot character (.)

Symbol = .

Regex = a.e

a.e

Software is like entropy: It is difficult to grasp, weighs nothing, and obeys the Second Law of Thermodynamics; i.e., it always increases. - *Norman Augustine*

Ranges

Symbol = []
Regex = er[ey]

er[ey]
There is only one problem with common sense; it's not very common. - *Milt Bryce*

Shortcuts (ASCII): [0-9], [a-z], [0-9a-zA-Z]

NTNU

Multipliers

- * : item occurs zero or more times.
- + : item occurs one or more times.
- ? : item occurs zero or one time.
- {a} : item occurs exactly a times.
- {a,b} : item occurs between a and b times.
- {a,} : item occurs at least a times.

Multipliers

et+

Getting information off the Internet is like taking a drink from a fire hydrant. - *Mitchell Kapor*

Greedy

a.*e

The greatest enemy of knowledge is not ignorance, it is the illusion of knowledge. - *Stephen Hawking*

Not greedy

a.*?e

The greatest enemy of knowledge is not ignorance, it is the illusion of knowledge. - *Stephen Hawking*

Shortcut for character classes

- \s : Matches anything which is considered whitespace.
- \S : Matches anything which is NOT considered whitespace.
- \d : Matches anything considered a digit. (Same as [0-9])
- \D : Matches anything whis is NOT considered a digit.
- \w : Matches anything which is considered a word character. (Same as [A-Za-z0-9_])
- \W : Matches anything which is NOT considered a word character.

\w+

Imagination is more important than knowledge. For knowledge is limited, whereas imagination embraces the entire world, stimulating progress, giving birth to evolution. - *Albert Einstein*

Anchors and word boundaries

- Symbol = ^ and $
- Regex = ^\w+ and \w+ [\.\!\?]?$

^\w+
Your most unhappy customers are your greatest source of learning. - *Bill Gates*

\w+ [\.\!\?]?$
Your most unhappy customers are your greatest source of learning. - *Bill Gates*

- Symbol = \b and \b
- Regex = \bi\w+\b

\bi\w+\b
The best way to get accurate information on Usenet is to post something wrong and wait for corrections. - *Matthew Austern*

Grouping

- Symbol = ()
- Regex = (\d{1,3}\.){3}\d{1,3}

(\d{1,3}\.){3}\d{1,3}
This is an IP address 210.1.41.103, but this is not 112.3332.124.1

\b((25[0-5]|2[0-4][0-9]|1[0-9][0-9]|[1-9]?[0-9])\.){3}(
25[0-5]|2[0-4][0-9]|1[0-9][0-9]|[1-9]?[0-9])\b

Alternation (or)

- Symbol = |
- Regex = (John|Michael) Smith

(John|Michael) Smith

We want to match John Smith and Michael Smith, but not Paul Smith.

Example: NTNU email address

- Alt. 1 = [a-z]+@(stud\.)?ntnu.no
- Alt. 2 = [a-z]+@(stud\.ntnu\.no|ntnu\.no)

[a-z]+@(stud\.)?ntnu.no

mariusre@stud.ntnu.no; mariusre@ntnu.no; marius.reed@lyse.net;john.smith@gmail.com

NTNU

Re module in python

- re.compile(pattern, flags=0)
- re.search(pattern,string,flags=0)
- re.match(pattern,string,flags=0)
- re.split(pattern,string,maxsplit=0, flags = 0)
- re.findall(pattern,string,flags=0)
- re.finditer(pattern,string,flags=0)
- re.sub(pattern,repl,string,count=0,flags=0)

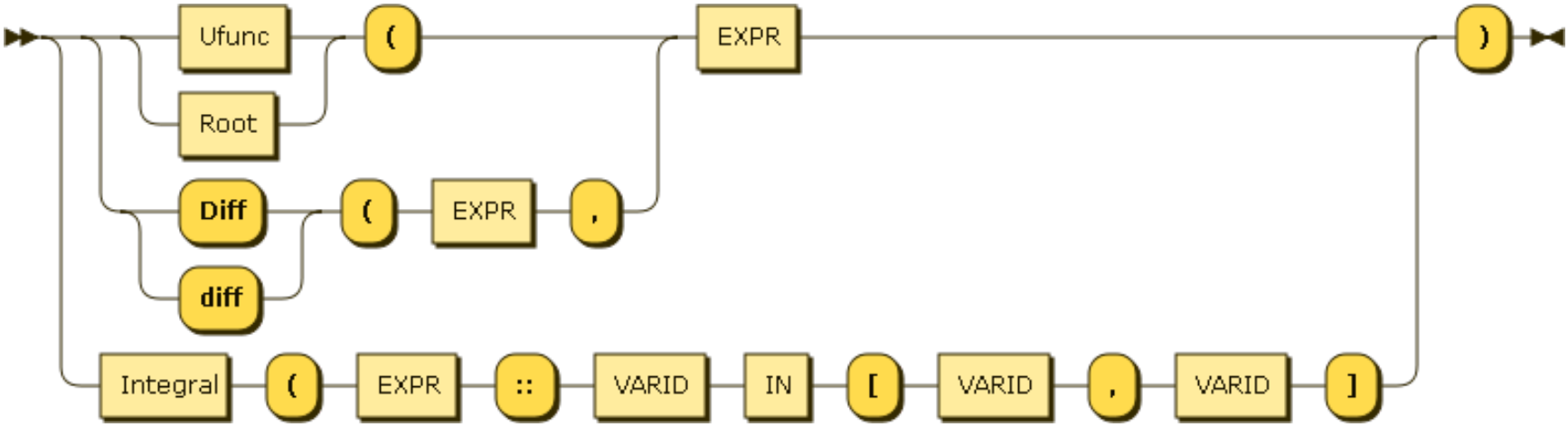
Backus-Naur form

- Notation technique for context free grammar
- Describe syntax

```
<postal-address>   ::= <name-part> <street-address> <zip-part>
<name-part>        ::= <personal-part> <last-name> | <personal-part> <name-part>
<personal-part>    ::= <initial> "." | <first-name>
<street-address>   ::= <street-name> <house-num> <opt-apt-num>
<zip-part>         ::= <ZIP-code> <town-name>
<opt-apt-num>      ::= <apt-num> | ""
```

Rail road diagram (Syntax Diagram)

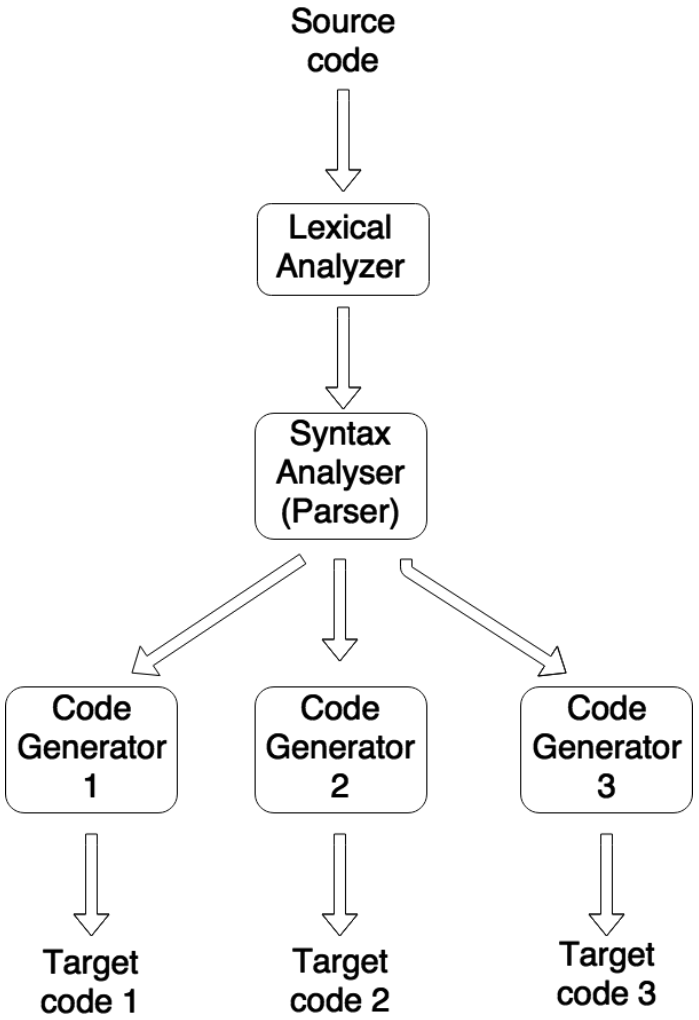
<FUNC> ::= <Ufunc> "(" <EXPR> ")" | <Integral> "(" <EXPR> "::" <VARID> <IN> "[" <VARID> "," <VARID> "]" ")" | <Root> "(" <EXPR> ")" | "Diff" "(" <EXPR> "," <EXPR> ")" | "diff" "(" <EXPR> "," <EXPR> ")"



Compilers

"A compiler is a special program that processes statements written in a particular programming language and turns them into machine language or "code" that a computer's processor uses"

- Lexical analyzer
- Parser
- Code Generator



Objective

```
[44]
lhs = ndiff
incidence_list = ['Fn_d', 'Fn_v', 'Pnrn', 'fn_d', 'fn_v', 'tnr']
rhs = Fn_v|arc_v & species|fn_v+Fn_d|arc_d & species|fn_d+Pnrn|node_conversion &
↪  species|tnr
layer = ontology_physical
```

```
ndiff = reduceProduct(Fn_v,add(fn_v,reduceProduct(Fn_d,add(fn_d,reduceProduct(
↪  Pnrn,tnr,node_conversion  &  species)),arc_d  &  species)),arc_v  &
↪  species)
```

Lua and LPeg

- PEG = Parsing expression grammar
  - Choice operator is ordered

- LPeg
  - LPeg patterns
  - Recursive descent parser

LPEG	Description	re python
lpeg.P(*string*)	Creates a pattern that matches *string* literally	re.compile(*string*)
lpeg.S(*string*)	Creates a pattern that matches any character in *string*	re.compile([*string*])
lpeg.R("xy")	Creates a pattern that matches any character between x and y	re.compile([x-y])
patt1 * patt2	Matches patt1 followed by patt2	Not introduced
patt1 + patt2	Matches patt1 or patt2	patt1 | patt2
patt^n	Matches at least n repetitions of patt	patt{n,}
patt^-n	Matches at most n repetitions of patt	patt{,n}

Lexer

```
-- Lexer
spc = L.S(" \t\n")^0
Ufunc         = spc * L.P("inv") + L.P("sqrt") + L.P("exp") + L.P("log") +
↪  L.P("sin")
              + L.P("cos") + L.P("tan") + L.P("asin") + L.P("acos")
              + L.P("atan") + L.P("sign") + L.P("neg") + L.P("abs")
              + L.P("min") + L.P("max") * spc
Root          = spc * L.P("root") * spc
Left          = spc * L.P("<") * spc
Right         = spc * L.P(">") * spc
IN                  = spc * L.P("in") * spc
Integral = spc * L.P("integral") * spc
Selection = spc * L.P("select") * spc
Instantiate = spc * L.P("set") * spc
VarID = spc * L.R("az","AZ","__","09")^0 * spc
sum = spc *  L.S('+-') * spc
power = spc * L.P("^") * spc
dot = spc * L.P(".") * spc
DL = spc * L.P("|") * spc
KR = spc * L.P("'") * spc
par = spc * L.P("(") + L.P(")") * spc
real = spc * L.C(
           L.P('-')^-1 *
           L.R('09')^0 *
           ( L.P('.') *
             L.R('09')^0
           )^-1 ) * spc /
         tonumber
integer = L.R("09")^1
```

NTNU

Parser

- *The parser will parse the token sequence to identify the syntactic structure of the program.*
- “Connected” to the source language, not the target language.
- *Captures* tokens that is sent to the code generator

Parser and Backus-Naur form

<START> ::= <INSTANTIATE> "(" <ATOM> ")"|<EXPR>
<EXPR> ::= <TERM> sum <EXPR> | <SEL>
<TERM> ::= "(" <EXPR> ")" | <FACT> dot <EXPR> | <FACT> KR <EXPR>|
<FACT> DL <INDID> DL <EXPR>| <FACT>

```
myGrammar = L.P{
        "START";
        START = (
                L.Ct(Instantiate * "(" * L.V("ATOM") *  ")")/instant
                + L.V("EXPR")
                ),
        EXPR = (
                L.Ct(L.V("TERM") * L.C(sum) * L.V("EXPR"))/add
                + L.V("SEL")
                ),
        TERM = (
            L.Ct("(" * L.V("EXPR") * ")") /group
            + L.Ct(L.V("FACT") * dot * L.V("EXPR"))/ExpandProduct
                + L.Ct(L.V("FACT") * KR * L.V("EXPR"))/KhatriRao
                + L.Ct(L.V("FACT") * DL * L.V("INDID") * DL *
                 ↪  L.V("EXPR"))/ReduceProduct
                + L.V("FACT")
                ),
```

NTNU

Expression	Function called
$\underbrace{\text{half.(abs(Fm\_vv) + (Fm\_vv.D\_v))}}_{EXPR}$	
$\underbrace{\text{half}}_{FACT}\ \underbrace{.}_{dot}\ \underbrace{\text{(abs(Fm\_vv) + (Fm\_vv.D\_v))}}_{EXPR}$	ExpandProduct(FACT, EXPR)
$\underbrace{\text{half}}_{VARID}$	Just captured
$\underbrace{\text{abs(Fm\_vv)}}_{TERM}\ \underbrace{+}_{sum}\ \underbrace{\text{(Fm\_vv.D\_v)}}_{EXPR}$	add(TERM, EXPR)
$\underbrace{\text{abs}}_{Ufunc}\ \underbrace{\text{(Fm\_vv)}}_{EXPR}$	abs(EXPR)
$\underbrace{\text{Fm\_vv}}_{VARID}$	Just captured
$\underbrace{\text{Fm\_vv}}_{FACT}\ \underbrace{.}_{dot}\ \underbrace{\text{D\_v}}_{EXPR}$	ExpandProduct(FACT, EXPR)
$\underbrace{\text{Fm\_vv}}_{VARID}$	Just captured
$\underbrace{\text{D\_v}}_{VARID}$	Just captured
TOTAL	ExpandProduct(half, add(abs(Fm_vv),ExpandProduct(Fm_vv, D_v)))

Code generation

- Captures from the parser is sent to the code generator
- In this case, several Lua functions

```
myGrammar = L.P{
        "START";
        START = (
                L.Ct(Instantiate * "(" * L.V("ATOM") *  ")")/instant
                + L.V("EXPR")
                ),
```

```
function instant(myList)
    return ("Instantiate(" .. myList[1] .. ")" )
end
```