Lect 07

CST8152 Compilers

Algonquin College

Computer Engineering Technology

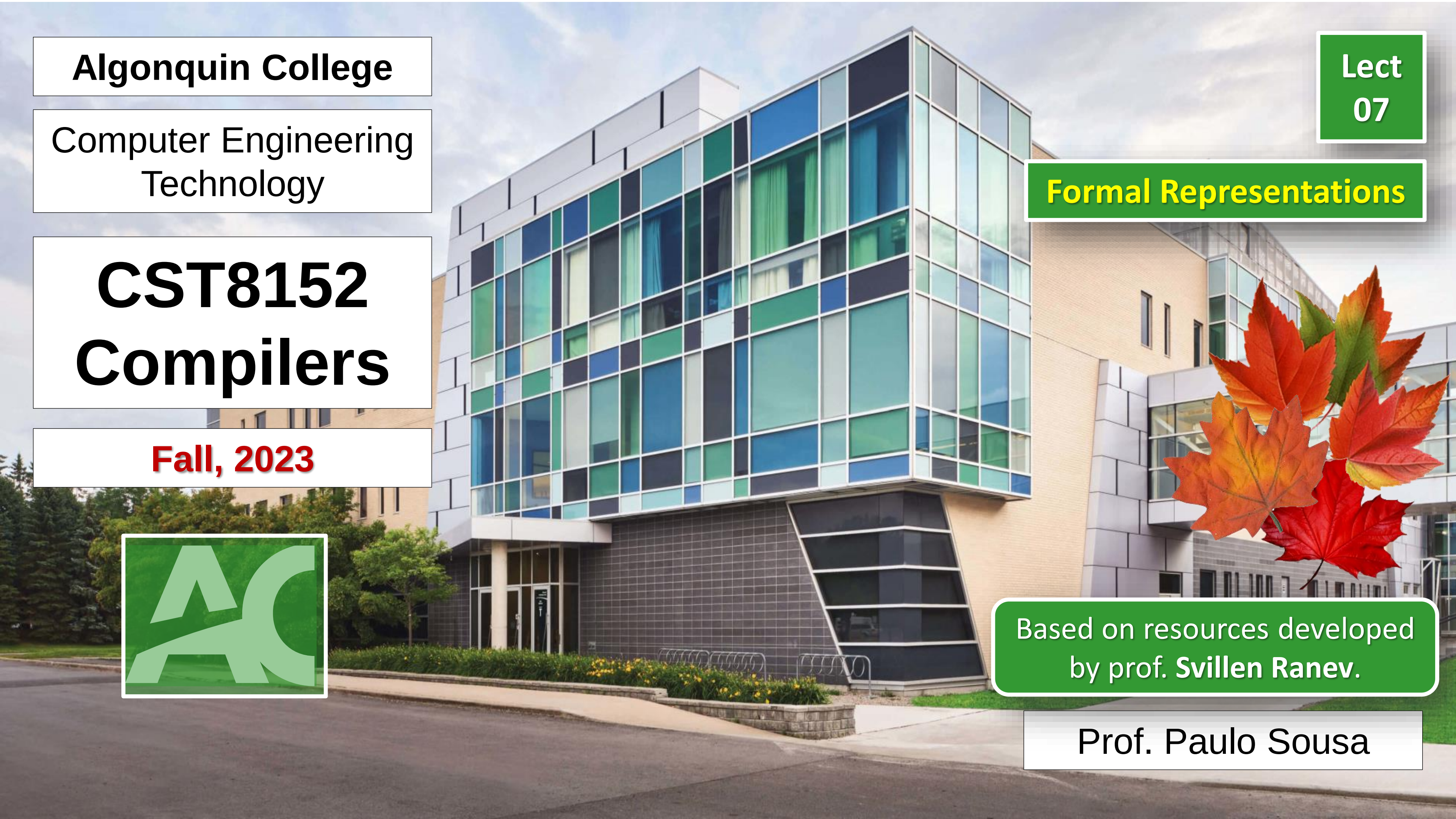
CST8152 Compilers

Fall, 2023

Lect 07

Based on resources developed by prof. **Svillen Ranev**.

Prof. Paulo Sousa



Algonquin College

Computer Engineering Technology

CST8152 Compilers

Fall, 2023

Lect 07

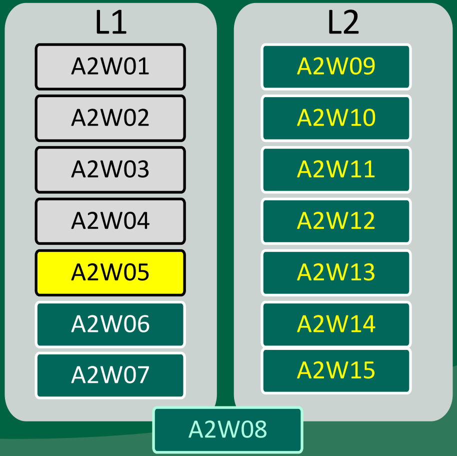
Formal Representations

Based on resources developed by prof. **Svillen Ranev.**

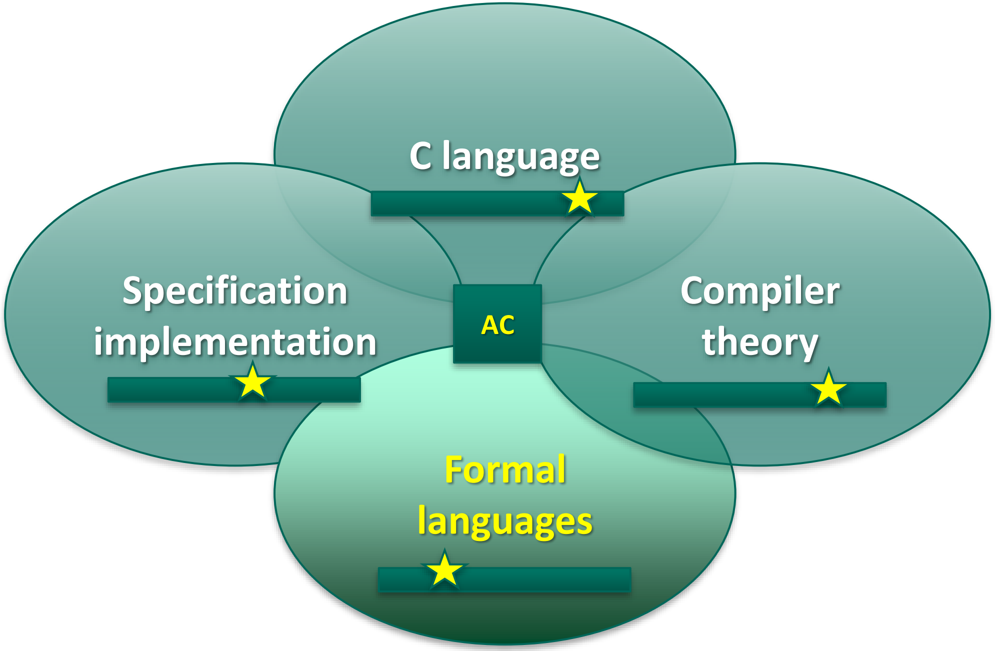
Prof. Paulo Sousa

Art 7: Formal Representations

- *General View*
- *Regular Expressions*
- *Grammar*
- *Automata*



Let's start…


C language
Specification implementation
AC
Compiler theory
Formal languages

Compilers – Art. 7

Formal Representations

Universal Concepts

Alphabet:

- An **alphabet** ∑ is a finite, non-empty, set of symbols. For example:
- The binary alphabet is {0, 1}
- The decimal alphabet is {0,1,2,3,4,5,6,7,8,9}
- **Note**: The metasymbols { , and } used here that are not in the alphabet.

*** IMPLEMENTATION NOTE ***

- For the scanner, the alphabet may be characters in the **ASCII** character set.
- For the parser the alphabet is the set of tokens produced by the scanner.
- Ex. Important sets: *keywords*

MOLD Language keywords: { “data", “code", “int", “real", “string", “if", “then", “else", “while", “do"}

Universal Concepts

String:

- A string is a finite set of symbols from an alphabet (not necessarily in a grammar).
- **Example**:
  - For the alphabet ∑ = {a, b, c} some strings are: {a, b, c, aa, ab, ac, ba, bb, bc, ca, cb, cc, abc, acb, bac, bca, cab, cba}
- **Note**: The order of symbols in a string **matter**.

Empty string:

- ε is the **empty** string.
- It is the string consisting of no symbols.
- The length of a string s is |s| and is equal to the number of symbols in the string.
- So: |ε| = 0, |abc| = 3.

Universal Concepts

Equipotency of ε:

- ε can be considered as a neutral element in operations:

$$\varepsilon = \varepsilon\varepsilon = \varepsilon^k$$

$$\varepsilon x = x\varepsilon = x$$

Think about this [2]:
- What is the best model for PL?

Empty Language:

Useless

- **The regular expression that matches nothing: Φ.**
  - Defined by Φ, it is the pattern for nothing; it generates the set containing nothing;
  - **Representation**: **L(Φ) = { }**
- **Note:** This is not the same as the empty string.
  - By contrast, ε is the pattern for the set that contains the string that contains no characters.

Operations in Languages

Concatenation of sets

- The concatenation of two sets A and B is defined by:

AB = { xy | x in A and y in B }

which reads "*the set of strings xy such that x is in A and y is in B*".

- For example.
- If A = {a,b} and B = {c,d} then AB = { ac, ad, bc, bd }

Powers of sets

- The power of a set A: The repetition of A several times.

A^4 = { x | 4-symbol string }

which reads "*the set of strings with four symbols*".

- This is just repeated:

$A^0 = \{ \varepsilon \}$, $A^1 = A$, $A^2 = AA$, $A^3 = AAA$, ...

- Note that $A^0 = \{ \varepsilon \}$ (for any set)

Operations in Languages

Union of sets

- The union of two sets A and B is defined by:

$$A \cup B = \{ x \mid x \text{ in } A \text{ or } x \text{ in } B \}$$

which reads "*the set of strings x such that x is in A or x is in B*".

- For example.
- If A = {a,b} and B = {c,d} then AUB = { a, b, c, d }

Kleene closure

- The Kleene closure of a set A is the * operator defined as the set of all strings including the empty string:

$$A^* = \bigcup_{i=0}^{\infty} A^i$$

- It is the union of all powers of A.

$A^* = A^0 + A^1 + A^2 + A^3 + \ldots$

Operations in Languages

Positive closure

- The positive closure is the + operator defined as the set of all strings excluding the empty string:

$$A^{+} = \bigcup_{i=1}^{\infty} A^{i}$$

- It means:

$A+ = A^* - \{\varepsilon\} = A^1 + A^2 + A^3 + \ldots$

Examples

Let L be the set {A, B, … Z, a, b, … z} Let D be the set {0, 1, …….9}

- $L \cup D$ is the set of letters and digits: {A, B, … Z, a, b, … z, 0, 1, …….9}
- LD is the set of strings consisting of a letter followed by a digit: {A0, A1, A2 .., B0, B1,… Z9, a0, b0, a1, b1 … }
- L^4 is the set of all four-letter strings
- L^* is the set of all strings of letters, including ε.
- $L(L \cup D)^*$ is the set of all strings of letters and digits beginning with a letter
- D+ is the set of all strings of one or more digits

Compilers – Art. 7

RE (Regular Expressions)

ALGONQUIN COLLEGE

Remember Kleene Theorem

- **Main Idea:**

The language that can be defined by:

1. Regular Expressions (compact language);
2. Regular Grammar (syntax production rules);
3. Finite Automaton (DFA);
4. Transition graph (transition / state diagrams).

Prof. Kleene

Source: Wikipedia

Model 1: Regular Expressions:

1. Regular expressions are a **convenient notation** (or means or tools) for specifying certain simple (though possibly infinite) set of strings over some alphabet.

2. A regular expression is a shorthand equivalent to a regular grammar.

L(RE) = L(G)

Remember Kleene Theorem

- **TIP**: A regular expression can be used to construct a **Deterministic Finite Automaton (DFA)** which therefore can recognize strings (words) of the grammar, which is the purpose of the Scanner.

- The sets of strings defined by regular expression are termed **regular sets**.

To define the RE (as any expression notation) use operands and operations.

- The **operands** are alphabet symbols or strings defined by regular expressions (regular definitions).
- The standard operations are **catenation** (concatenation) , union or **alternation** (|), and **recursion** or Kleene closure (*)
- Regular expressions use the **metasymbols** |, (,), {, } , [,], * , + (and others ?, ^) to define its operations.

RE Operations:

- **Catenation**

1. RE: For the alphabet Σ = {a, b, c}, if x = ab and y = b&c, then x.y = abbc.
2. Note: Since ε is a valid word (empty string), remember that: εx = xε = x

Alternation:

1. RE: For example if Σ = {a, b, c, d} then L(a|b) = {a,b} and L(c|d) = {c,d} so

- a|b is either a or b
- c|d is either c or d

RE Operations:

- **Recursion**

1. RE: For a regular expression r, Kleene Closure is defined by:

- $L(r^*) = L(r)^*$
- which means concatenation with all powers of L.

Example:

$L((a|bb)^*) = L(a|bb)^* = L(a|bb)^0 + L(a|bb)^1 + L(a|bb)^2 + L(a|bb)^3 + \ldots$

- $L(a|bb)^0 = \{\varepsilon\}$
- $L(a|bb)^1 = \{a, bb\}$
- $L(a|bb)^2 = \{a, bb\}\{a, bb\} = \{aa, abb, bba, bbbb\}$ or:

- $L((a|bb)^*) = \{\varepsilon, a, bb, aa, abb, bba, bbbb, aaa, abba, bbaa, bbbba, aabb, \ldots\}$

- $\{dc, ddc, dddc, ddddc, \ldots\} = d^+c(a|c|dd+)^*$

More about RE

Special operations

- Positive Closure (One or more +)
  - a+ = aa* (also a* = a+ | ε)
- Exponentiation or Power operation (exactly k)
  - a^k = aaa…a (exactly k times)
- Optional Inclusion (Zero or one ?)
  - a? = a | ε

(a|c|dd+)*? = ε | (a|c|dd+)* = ε | a | c | dd | aa

Character classes:

- Specify a **range** of characters or numbers that follow a sequence.
- Examples:
  - [a-z] means any character in the range a to z.
  - [A-Z] means any character in the range A to Z.
  - A regular expression for the **pattern** for an identifier that begins with a letter or an underscore and is followed by any number of numbers and letters is:

[a-zA-Z_][A-Za-z0-9]*

{ a, b, …, A, B, ..., _, aA, aa0, ….}

ALGONQUIN COLLEGE

More about RE

Complements:

- A **complement character class** is specified using the ^ or (~) symbols, or **Not** operator.
- Examples:
  - [^a-z] matches any character except a to z.
  - Comment = // (^CR)*CR

TIP

Sometimes, the complement is the better (shortest) to represent collections.

Precedence of operators:

- Decrease precedence order:

- Grouping: ( )
- Character classes: [ ]
- Power / Recursion (Kleene star): *
- Concatenation: , . , , ,
- Alternation: |
- Positive closure: +
- Optional: ?
- Iteration: k

More about RE (1)

- **Remember:**

Example 3.4: Let $\Sigma = \{a, b\}$.

1. The regular expression $\mathbf{a}|\mathbf{b}$ denotes the language $\{a, b\}$.
2. $(\mathbf{a}|\mathbf{b})(\mathbf{a}|\mathbf{b})$ denotes $\{aa, ab, ba, bb\}$, the language of all strings of length two over the alphabet Σ. Another regular expression for the same language is $\mathbf{aa}|\mathbf{ab}|\mathbf{ba}|\mathbf{bb}$.
3. $\mathbf{a}^*$ denotes the language consisting of all strings of zero or more a's, that is, $\{\epsilon, a, aa, aaa, \ldots\}$.
4. $(\mathbf{a}|\mathbf{b})^*$ denotes the set of all strings consisting of zero or more instances of a or b, that is, all strings of a's and b's: $\{\epsilon, a, b, aa, ab, ba, bb, aaa, \ldots\}$. Another regular expression for the same language is $(\mathbf{a}^*\mathbf{b}^*)^*$.
5. $\mathbf{a}|\mathbf{a}^*\mathbf{b}$ denotes the language $\{a, b, ab, aab, aaab, \ldots\}$, that is, the string a and all strings consisting of zero or more a's and ending in b.

More about RE (2)

- **RE Properties:**

Law	Description
$r \mid s = s \mid r$	$\mid$ is commutative
$r \mid (s \mid t) = (r \mid s) \mid t$	$\mid$ is associative
$r(st) = (rs)t$	Concatenation is associative
$r(s \mid t) = rs \mid rt;\ (s \mid t)r = sr \mid tr$	Concatenation distributes over $\mid$
$\epsilon r = r\epsilon = r$	ϵ is the identity for concatenation
$r^* = (r \mid \epsilon)^*$	ϵ is guaranteed in a closure
$r^{**} = r^*$	$*$ is idempotent

Figure 3.7: Algebraic laws for regular expressions

More about RE (3)

- **Common Syntax for RE (Lex):**
- Conventions are used in order to express RE in a specific notation.
- For instance, the Lex file (used for automated generated parsers), has the following notation…

EXPRESSION	MATCHES	EXAMPLE
c	the one non-operator character c	`a`
$\backslash c$	character c literally	`\*`
"s"	string s literally	`"**"`
.	any character but newline	`a.*b`
^	beginning of a line	`^abc`
$	end of a line	`abc$`
[s]	any one of the characters in string s	`[abc]`
[^s]	any one character not in string s	`[^abc]`
$\boxed{r*}$	zero or more strings matching r	`a*`
$r+$	one or more strings matching r	`a+`
$r?$	zero or one r	`a?`
$r\{m,n\}$	between m and n occurrences of r	`a{1,5}`
$\boxed{r_1 r_2}$	an r_1 followed by an r_2	`ab`
$\boxed{r_1 \mid r_2}$	an r_1 or an r_2	`a|b`
(r)	same as r	`(a|b)`
r_1/r_2	r_1 when followed by r_2	`abc/123`

Figure 3.8: Lex regular expressions

Grammar Examples

VID = AVID | SVID

- Examples: {a, b, c, a1, b12, c123, …, a$, b$, c$, a1$, b12$, c123$

AVID = L (L | D)*

- Examples: {a, b, c, ….abc, abcdf, abc123…}

L = a | b |…| z | A | B | …| Z

- Examples: {a, b, c,…z, A,…, Z }

D = 0 | … | 9

- Examples: {0, 1, 2, 3,…9 }

SVID = AVID$

- Examples: {a$, b$, c$, ….abc$, abcdf$, abc123$…}

Suppose a grammar in which string variables must finish with "$"

IL = DIL

- Examples: {1, 2, , 0, 00, 111, 123, … , 777, …}

DIL = ZDS | NzDD*

- Examples: {1, 2, , 0, 00, 111, 123, … , 777, …}

ZDS = 00*

- Examples: {0, 00, 000, … , 00000000}

NzD = 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

- Examples: {1, 2, … , 9}

D = 0 | NzD

- Examples: {0, 1, 2, … , 9}

Suppose a grammar in which numbers starting with "0" cannot include other digits.

Compilers – Art. 7

BNF – Grammar (review)

Remember Kleene Theorem

S
/ \
Sb a

- **Model 2: Grammar Formalization**

$$G = \{V_T, V_N, P, S\}$$

- Where:

1. V_T = Terminals;
2. V_N = Non-Terminals (V_N -> V_N | V_T)
3. P = Production rules: $\{A \rightarrow X_1X_2\ldots X_N\}$;
4. S = Start symbol.

Example:

1. G1: {{a,b}, {S}, P = {S $\rightarrow$Sb|a}, S}

The infinite set of words can be given by:

a, ab, abb, abbb, …

The corresponding RE1: ab*

Note: The grammar G1 and RE1 are equivalent!

Grammar Examples

<variable identifier> → <arithmetic variable identifier> | <string variable identifier>

- Example: {a, b, c, a1, b12, c123, …, a$, b$, c$, a1$, b12$, c123$, …}

<arithmetic variable identifier> → <letter> <opt_letters or digits>

- Example: {a, b, c, …,abc, abcdf, abc123…}

<opt_letters or digits> → <letters or digits> | ε

- Example: { ε, a, b, c, …, abc, abcdf, abc123…}

<letters or digits> → <letter or digit> | <letters or digits> <letter or digit>

- Example: {a, b, c, … ,1, 2, 3, 1s, e2 …1111, aaaaa,…}

<letter or digit> → <letter> | <digit>

- Example: {a, b, c, … ,1, 2, 3 }

<letter> -> a | b |…| z | A | B | …| Z

- Example: {a, b, c, … , z, A,…, Z }

<digit> -> 0 | … | 9

- Example: {0, 1, 2, 3, …, 9 }

Grammar Examples

<string variable identifier> -> <arithmetic variable identifier>$

- Example: {a$, b$, c$, ….abc$, abcdf$, abc123$…}

<integer literal> -> <decimal integer literal>

- Example: {1, 2, , 0, 00, 111, 123, … , 777, …}

<decimal integer literal> -> <zeros> | <non zero digit> <opt_digits>

- Example: {1, 2, , 0, 00, 111, 123, … , 777, …}

<zeros> -> 0 | <zeros>0

- Example: {0, 00, 000, … , 00000000}

<opt_digits> -> <digits> | ε

- Example: { ε , 1, 2, , 0, 00, 111, 123, … , 777, …}

<digits> -> <digit> | <digits> <digit>

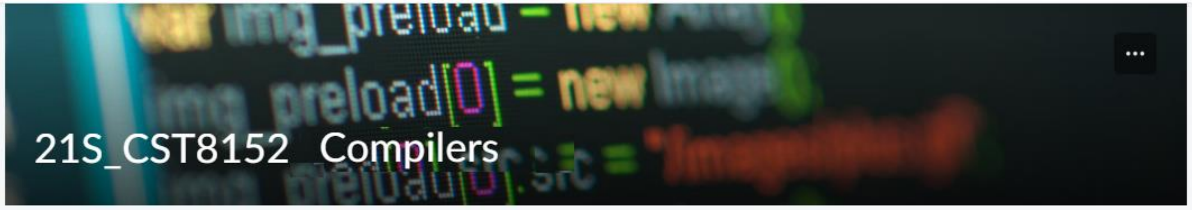
- Example: { 1, 2, , 0, 00, 111, 123, … , 777, …}

<digit> -> 0 | <non zero digit>

- Example: { 0, 1, 2, … , 9}

<non zero digit> -> 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

- Example: {1, 2, … , 9}



Compilers – Art. 7

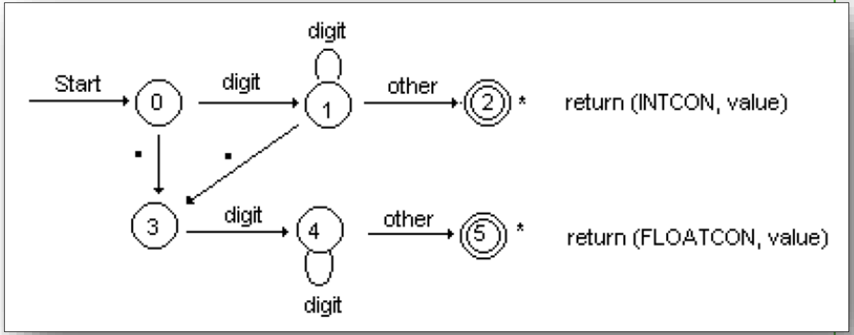
Starting Automata

ALGONQUIN COLLEGE

Remember Kleene Theorem

- **Model 3: Automaton:**

1. Functional way to define the evolution of an acceptable string in a language.
2. Can be visual such as:



Finite Automaton:

1. Mathematical representation of transitions that transform inputs in outputs that describes a language.

$$L(G) = A(\Sigma, Q, q0, Qf, \Delta)$$

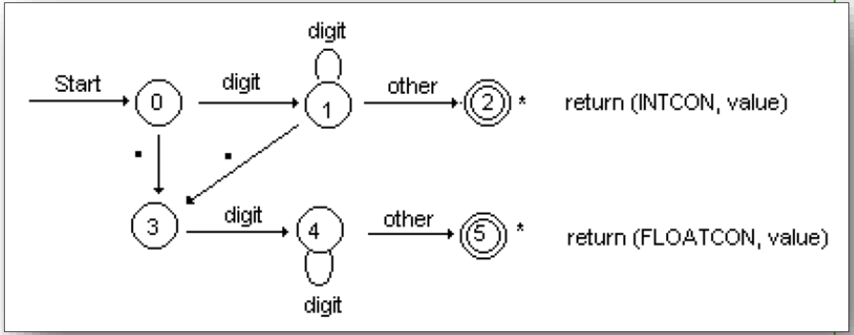
2. This notation includes the alphabet (Σ) that, starting in a state (q0) can perform words in the end (Qf), by productions (Δ) between states (Q).

Note: Empty string (ε) is also acceptable.

Remember Kleene Theorem

- **Model 3: Automaton:**

1. Modeling: $L(G) = A(\Sigma, Q, q0, Qf, \Delta)$



Delta transitions:

Δ = {

δ(q0,digit) = q1;

δ(q0,period) = q3;

δ(q1,digit) = δ(δ(q0,digit),digit) = q1;

δ(q1,period) = δ(δ(q0,digit),period) = q3;

//...

}

Q \ Σ	digit	period	other(#)
-> 0	1	3	-
1	1	3	2
2*	-	-	-
3	4	-	-
4	4	-	5
5*	-	-	-

Remember Kleene Theorem

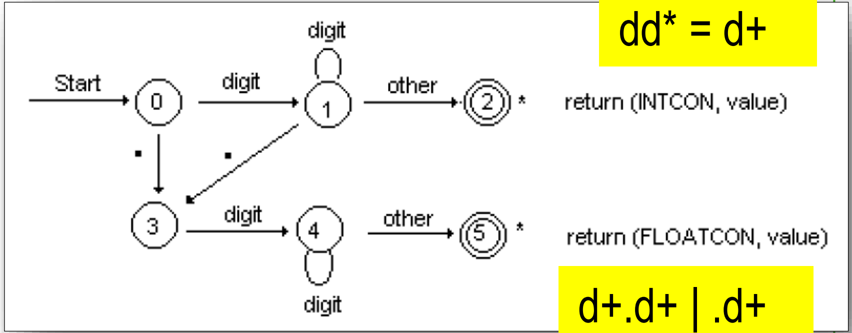
Other = ^digit && ^.
Σ=(digit,., other)

Q = set of states = {q0…q5}

Qf = {q2, q5}

- **Model 3: Automaton:**

1. Functional way to define the evolution of an acceptable string in a language.
2. Can be visual such as:



dd* = d+

d+.d+ | .d+

q1 = δ(q0, digit)

q3 = δ(q0, .), δ(q1, .)

Finite Automaton:

1. Mathematical representation of transitions that transform inputs in outputs that describes a language.

L(G) = A(Σ, Q, q0, Qf, Δ)

2. This notation includes the alphabet (Σ) that, starting in a state (q0) can perform words in the end (Qf), by productions (Δ) between states (Q).

Note: Empty string (ε) is also acceptable.

L(G) = Language from a Grammar (RE)

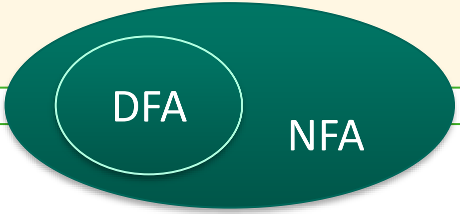
Formalization (1)

- **FA:**

$$FA = (\Sigma, Q, q0, Qf, \Delta)$$

- Where:

- Σ = Alphabet = {a0, a1, …, an}
- Q = Set of states = {q0, q1, …, qm}
- q0 = Initial state;
- Qf = Final states;
- Δ = P = Production rules: $\{q_y = \delta(q_x, a_m), \ldots, q_z = \delta(qy, an)\}$



NFA vs DFA

1. DFA: Deterministic, once you are in a state and read a symbol, you know exactly where to go.
2. NFA: Indeterministic because:

- It is possible to have more than one transition when read a symbol;
- Null (Epson) transitions: you can go to another state reading nothing.

Compilers – Art. 7

Concluding

ALGONQUIN COLLEGE

Review

- *Importance of Kleene Theorem.*

Source: https://static.wixstatic.com/media/7594af_51a81a8ccc5f418281f52c8bdd2dd618~mv2.jpg

Some Questions

1. *Why to use different models?*
2. *How to chose a model for a specific representation?*
3. *Can you see advantages and disadvantages of each model?*

Open questions…

- Any doubts / questions?
- How we are until now?

Image URL: https://cdn-az.allevents.in/events3/banners/26b150363d5757da8578fa6a1481585a368f12d4247adfe95837e6ea6c5ab2af-rimg-w1200-h549-gmir.jpg?v=1569691155

Compilers – Art. 7

Thank you for your attention