CS 460

Programming Languages

Fall 2023

Dr. Watts

(20 September 2023)

Course Administration

- Exercise 2 posted
- Project 1 Preliminary Exercise

Class money mutators and accessors (not discussed)

```
// Accessors and Mutators
unsigned getDollars () const;
unsigned getCents () const;
void setDollars (unsigned D);
void setCents (unsigned C);
unsigned * getCurrency () const;
void setCurrency (unsigned * C) const;
unsigned & Dollars ();
unsigned & Cents ();
```

class money

```
class money
{
    public:
      // Methods discussed on Monday

     // Accessors and Mutators
     int getDollars () const;
     int getCents () const;
     vector <unsigned> getCurrency () const;
     void setCurrency (vector <unsigned> & C);

    private:
     // Add attributes private member functions here.
     unsigned size;       // required
     unsigned * currency; // required
};
```

Project 1 Questions

- I noticed that anything input which matches to the LISTOP_T category would also match for the IDKEY_T category. Can we assume that the order the regular expressions are listed are also a "precedence" order, so that it first checks if "car" matches the LISTOP_T category?

Project 1 Questions

- If we did go that route, all of the "intermediate" states in the DFA for the LISTOP_T regular expression would have to be accepting states for the IDKEY_T regular expression because "ca" matches IDKEY_T.
  Does this sound correct? Most of what we did in class only had 1 accepting state per category and while I know it is valid to have multiple, I just wanted to confirm that.

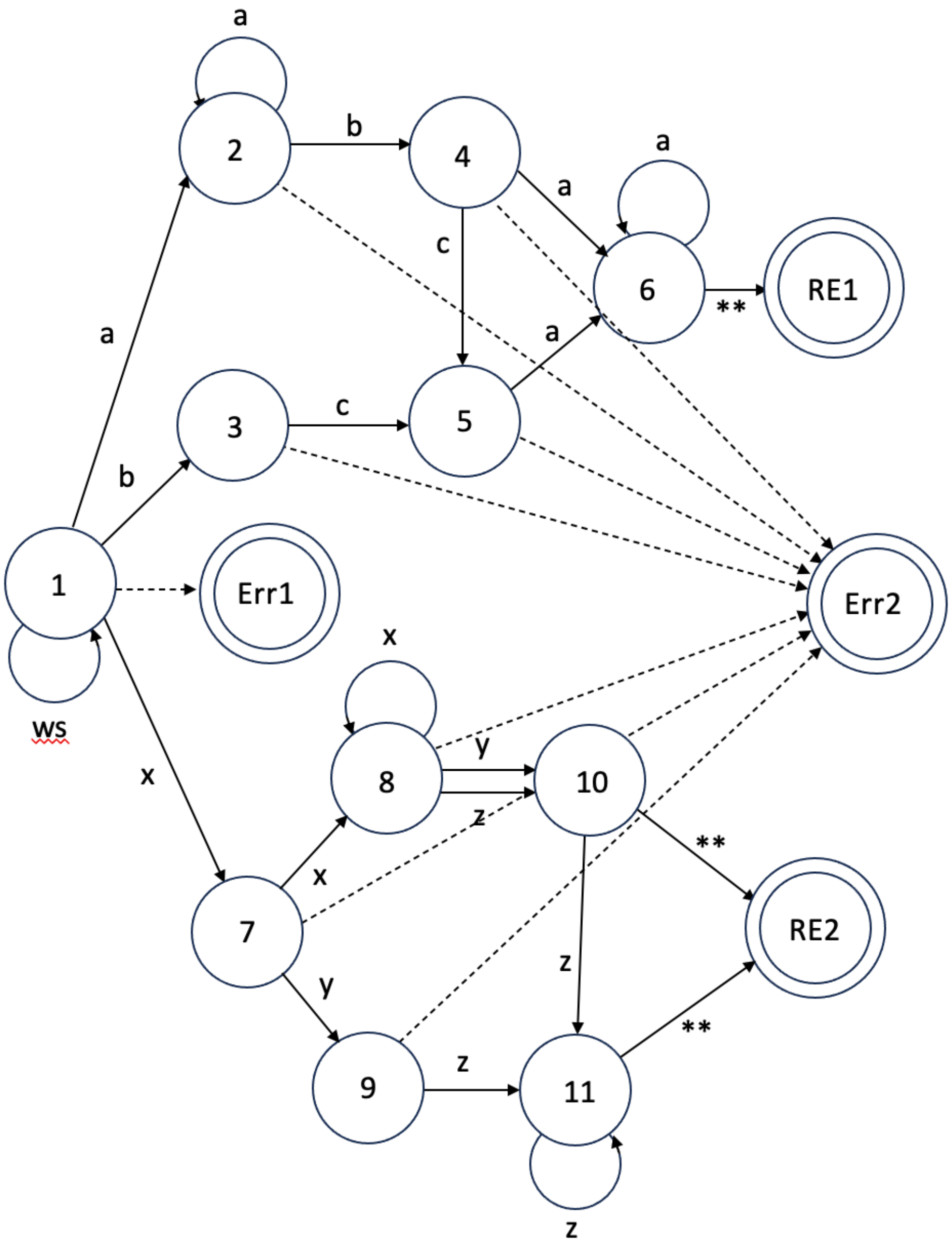
Project 1 Questions

- I wanted to confirm that DFAs shouldn't have any lambda transitions in them correct? as that wouldn't be "deterministic"?

DFAs as scanners (aka tokenizers)

- Alphabet = {a, b, c, x, y, z, ␣}
- Regular expression 1 (RE1)
  - a* (ab | bc) a+
- Regular expression 2 (RE2)
  - x+ (xy | yz | xz) z*
- Combined
  - (a* (ab | bc) a+) | (x+ (xy | yz | xz) z*)

(a* (ab | bc) a+) | (x+ (xy | yz | xz) z*)



Programming a DFA

- Table

	ws	a	b	c	x	y	z	other
1	1	2	3	Err1	7	Err1	Err1	Err1
2	Err2	2	4	Err2	Err2	Err2	Err2	Err2
3	Err2	Err2	Err2	5	Err2	Err2	Err2	Err2
4	Err2	6	Err2	5	Err2	Err2	Err2	Err2
5	Err2	6	Err2	Err2	Err2	Err2	Err2	Err2
6	RE1	6	RE1	RE1	RE1	RE1	RE1	RE1
7	Err2	Err2	Err2	Err2	8	9	Err2	Err2
8	Err2	Err2	Err2	Err2	8	10	10	Err2
9	Err2	Err2	Err2	Err2	Err2	Err2	11	Err2
10	RE2	RE2	RE2	RE2	RE2	RE2	11	RE2
11	RE2	RE2	RE2	RE2	RE2	RE2	11	RE2

Regular Expression for Numeric Literals

- Regular expression for general class of numeric literals signed/unsigned and integer/real
- Alphabet = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, -, +, .}
- Regular Expression

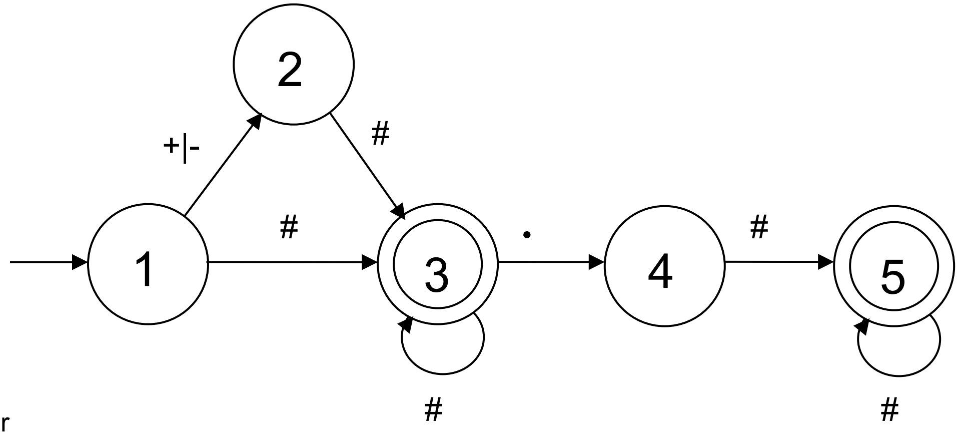
- How do you recognize the end of a numeric literal?

DFA for Numeric Literals

(+|-|λ)(0|1|2|3|4|5|6|7|8|9)+(.(0|1|2|3|4|5|6|7|8|9)+| λ)

DFA for Numeric Literals (+|-|λ)(0|1|2|3|4|5|6|7|8|9)+(.(0|1|2|3|4|5|6|7|8|9)+| λ)

a. 12
   a. 1 -> 3 -> 3 OK!
b. 1.2
   a. 1 -> 3 -> 4 -> 5 OK!
c. +12.34
   a. 1 -> 2 -> 3 -> 3 -> 4 -> 5 -> 5 OK!
d. 12.
   a. 1 -> 3 -> 3 -> 4 -> ends No!
e. .123
   a. 1 -> ends No!
f. 12.12.34
   a. 1 -> 3 -> 3 -> 4 -> 5 -> 5 see . error

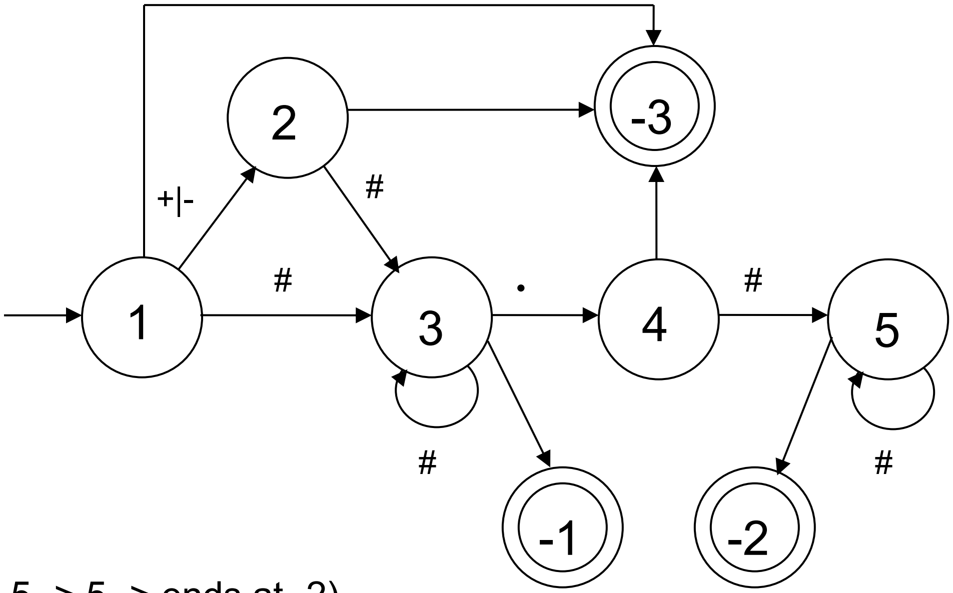


DFA for Numeric Literals – with terminating states

(+|-|λ)(0|1|2|3|4|5|6|7|8|9)+(.(0|1|2|3|4|5|6|7|8|9)+| λ)

DFA for Numeric Literals – with terminating states (+|-|λ)(0|1|2|3|4|5|6|7|8|9)+(.(0|1|2|3|4|5|6|7|8|9)+| λ)

- _ → represents a space
- 12 (1 -> 3 ->3 OK!)
- 1.2 (1-> 3 -> 4 -> 5 OK!)
- +12.34 (1 -> 2 -> 3-> 3 -> 4-> 5 -> 5)
- 12. (ends at 4)
- .123 (ends at 1)
- 12.12.34 (stops at 5 OK)
  - 12.12
- abcd (ends at -3)
- +abc (ends at -3)
- +_ (ends at -3)
- 4a (ends at -1)
  - 4
- 425_ (1 -> 3 -> 3 -> 3 -> ends at -1)
  - 4
- -12.345_ (1 -> 2 -> 3 -> 3 -> 4 -> 5 -> 5 -> 5 -> ends at -2)
  - -12.345
- What ends up at -1? integer
- What ends up at -2? double
- What ends up at -3? Non-numeric



How do the numeric literals for Project 1 differ from this example?

Next steps