COMP 520 - Compilers

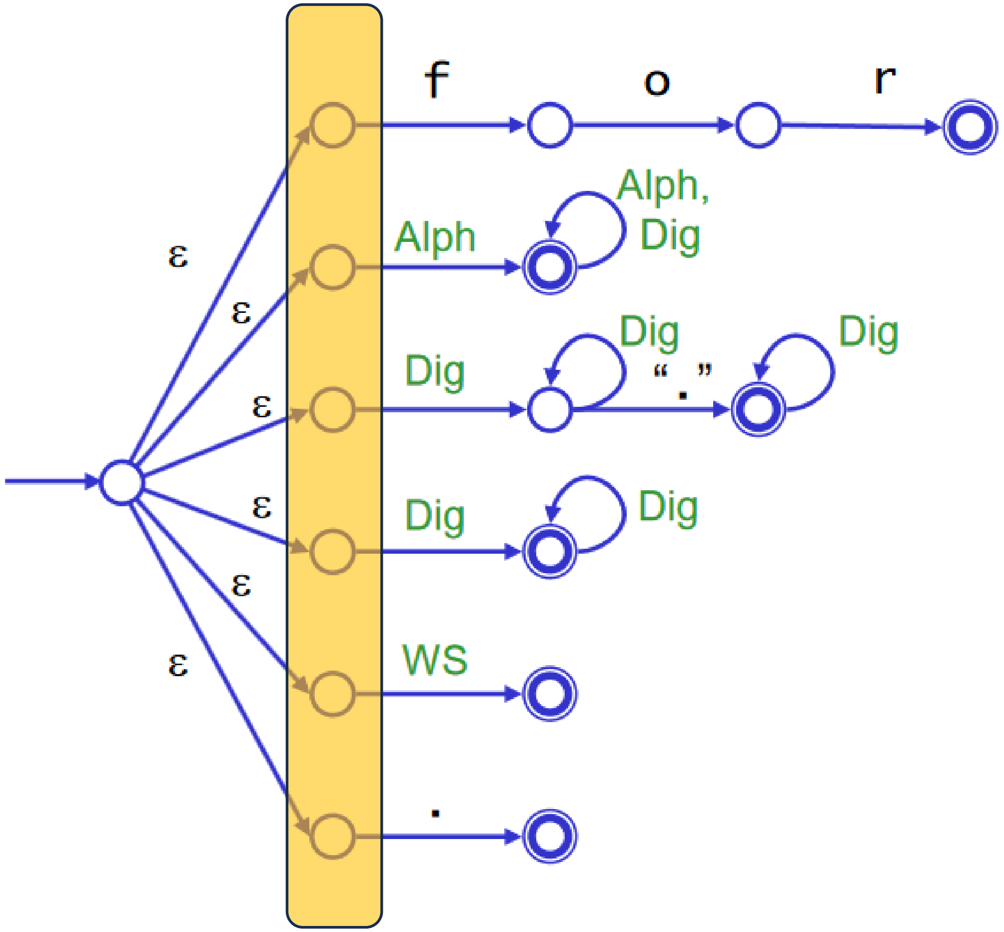
Lecture 05 – EBNF Properties, Prediction Sets

A second look at NFAs

- Goal: How can we go from an NFA to a DFA?
- Any tricky details?

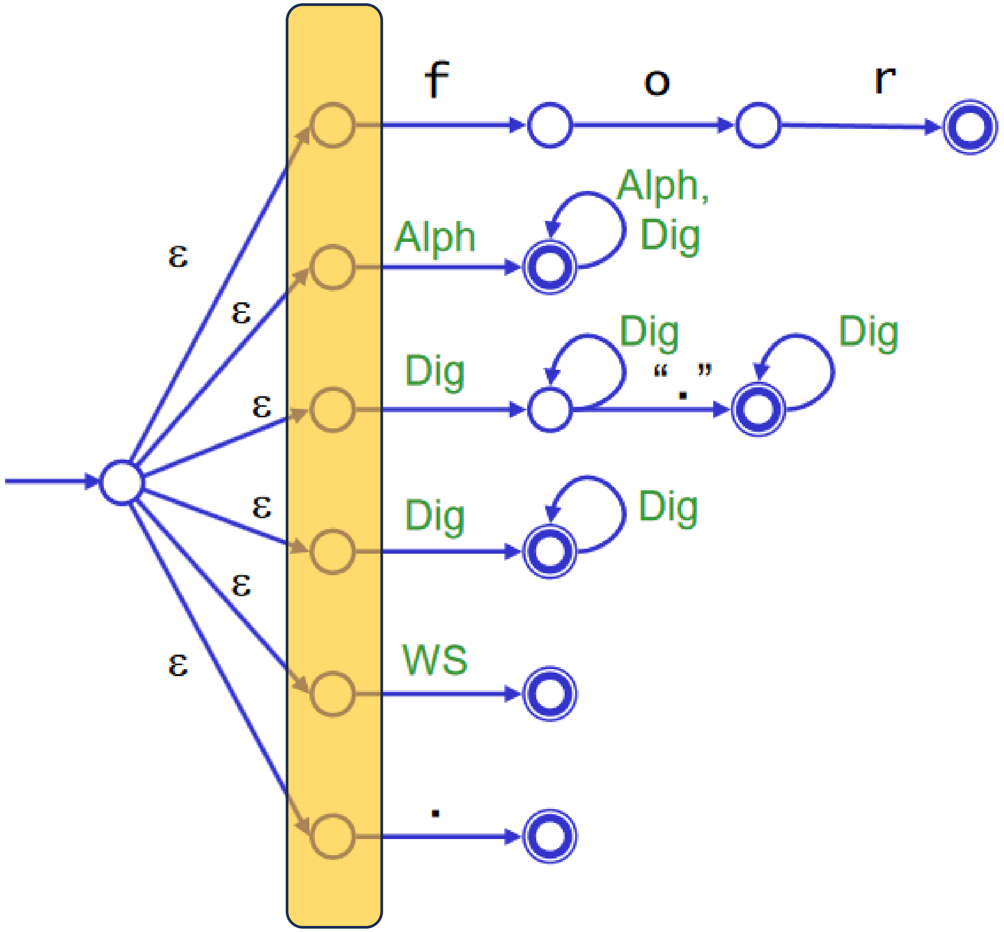
First, Initialize S

- S := ε-closure(q_0)
- S= highlighted regions



Look at input a= Dig

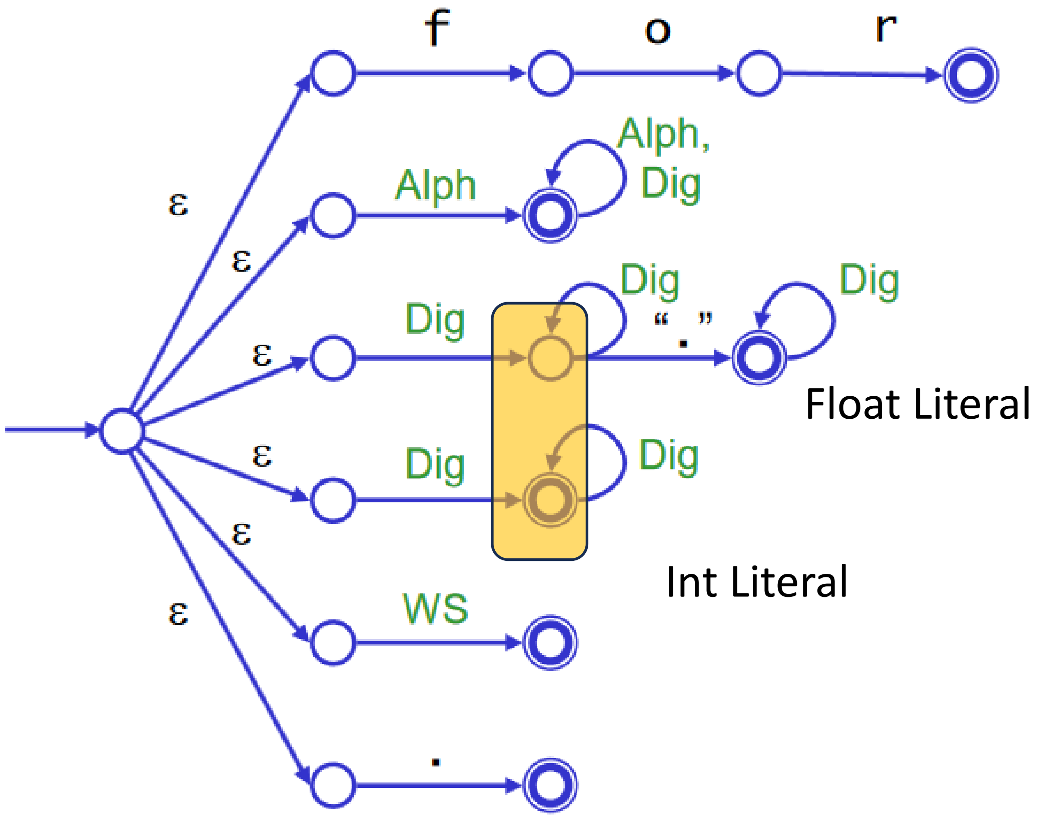
- a = Dig
- Find ε-closure($\hat{\delta}(S, a)$)



Look at input a= Dig

- a = Dig
- Find ε-closure($\hat{\delta}(S, a)$)

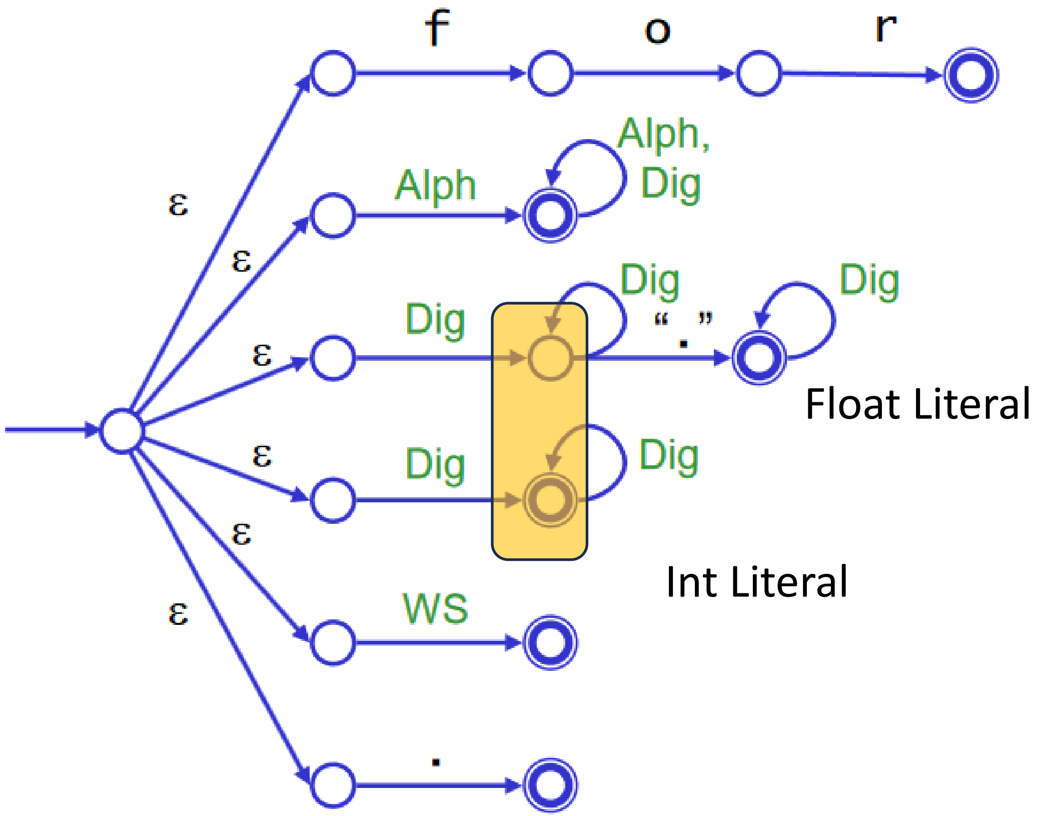
- S = highlighted region



Look at input a= Dig

- a = Dig
- Find ε-closure($\hat{\delta}(S, a)$)

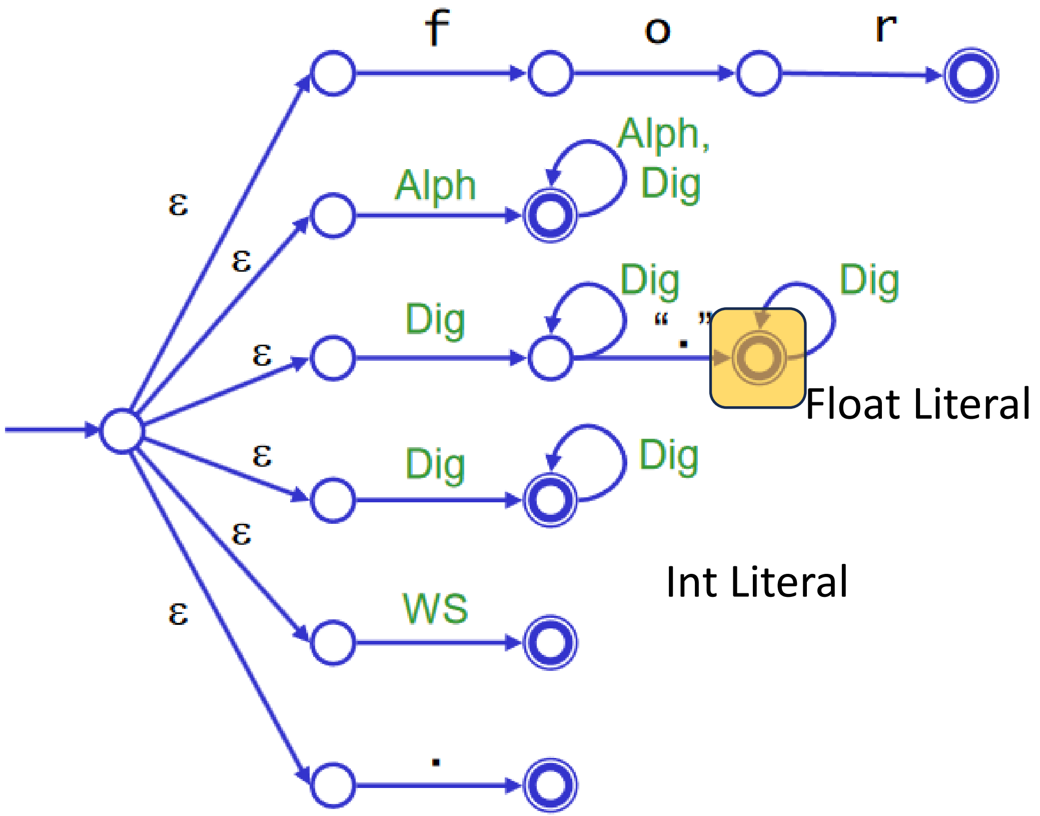
- S = highlighted region
- Because $S \cap F$ contains “Int Literal”, our current token is predicted to be “Int Literal”



Look at input a= “.”

- a = “.”
- Find ε-closure($\hat{\delta}(S, a)$)

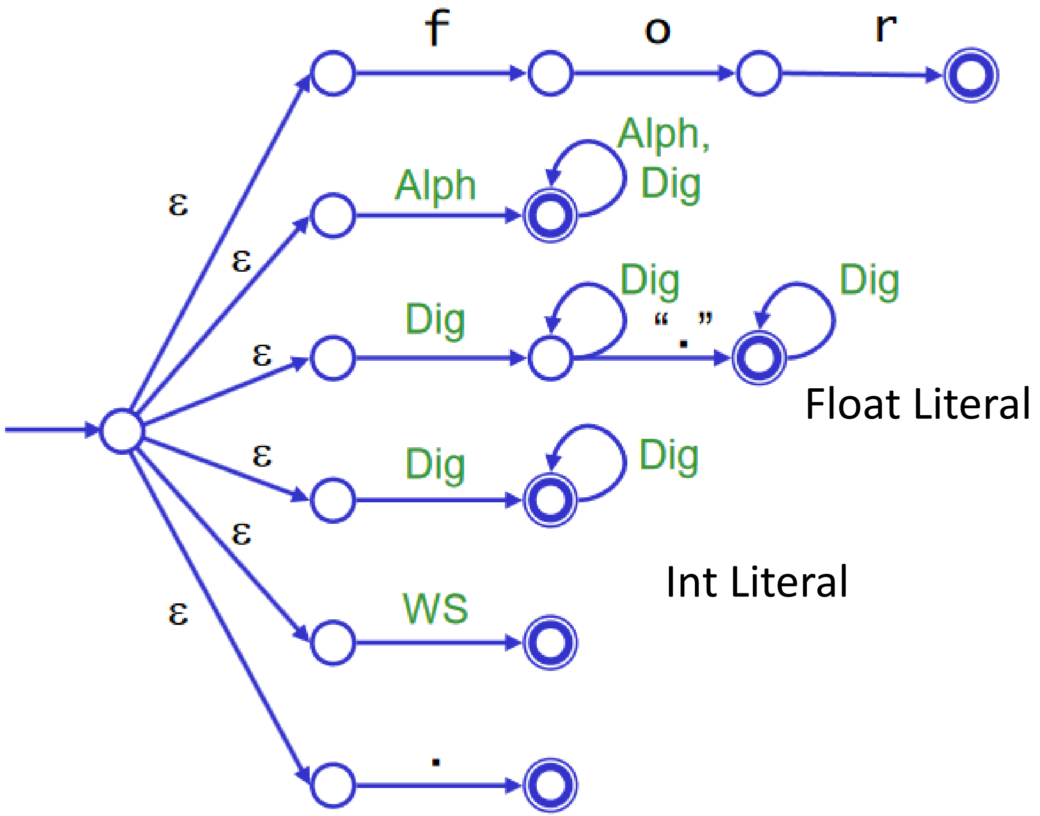
- S = highlighted region
- Now: $S \cap F$ contains Float Literal



Look at input a= “;”

- a = “;”
- Find ε-closure($\hat{\delta}(S, a)$)

- S = ∅
- Last known valid Token was “Float Literal”
- Restoring the state “rewinds” back to the Literal, and repeats.

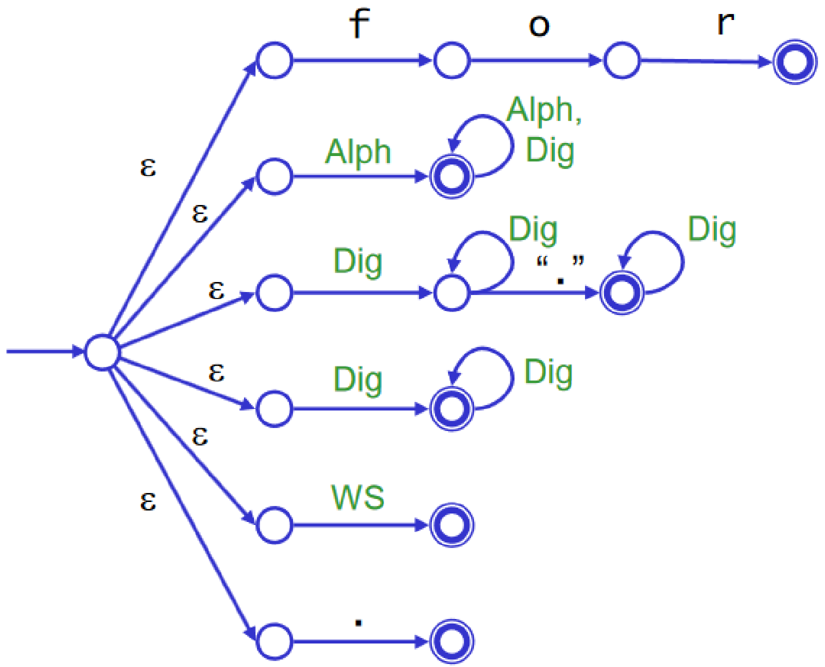


We can convert this to a DFA

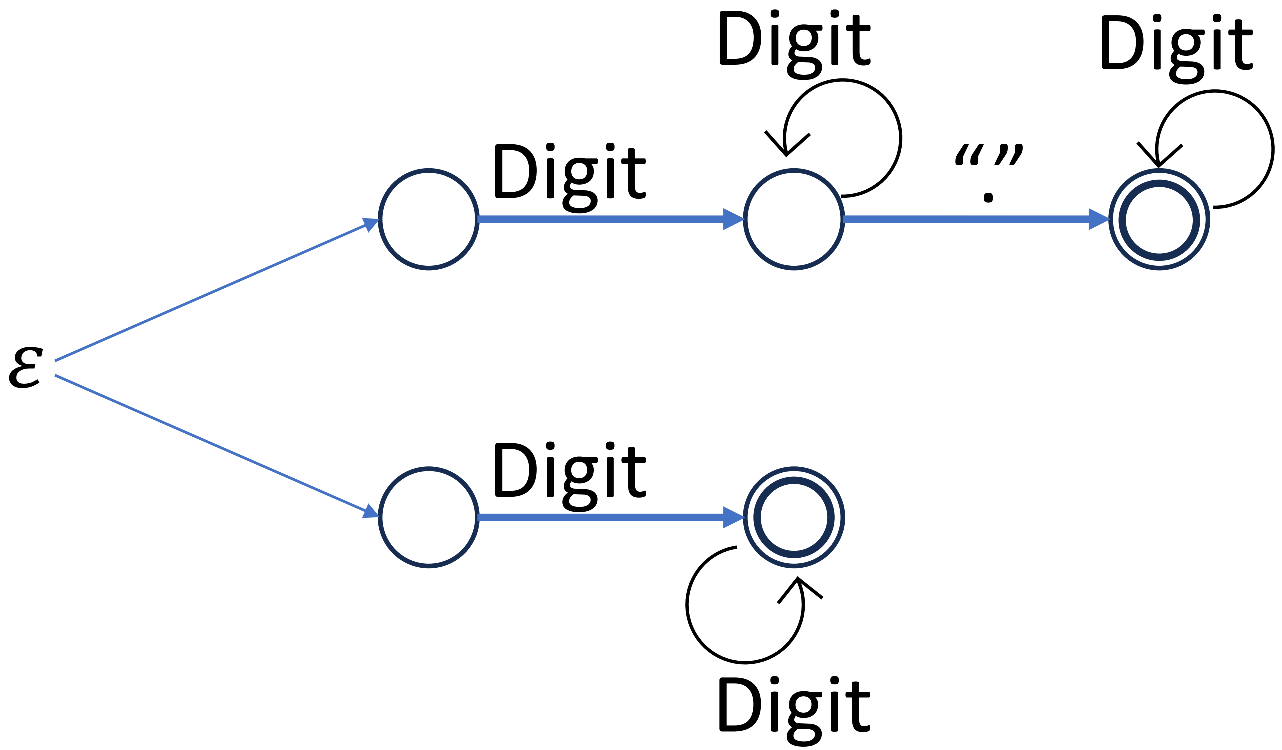
- $O(2^Q)$ operations is too expensive
- Must convert this to a DFA!

First step

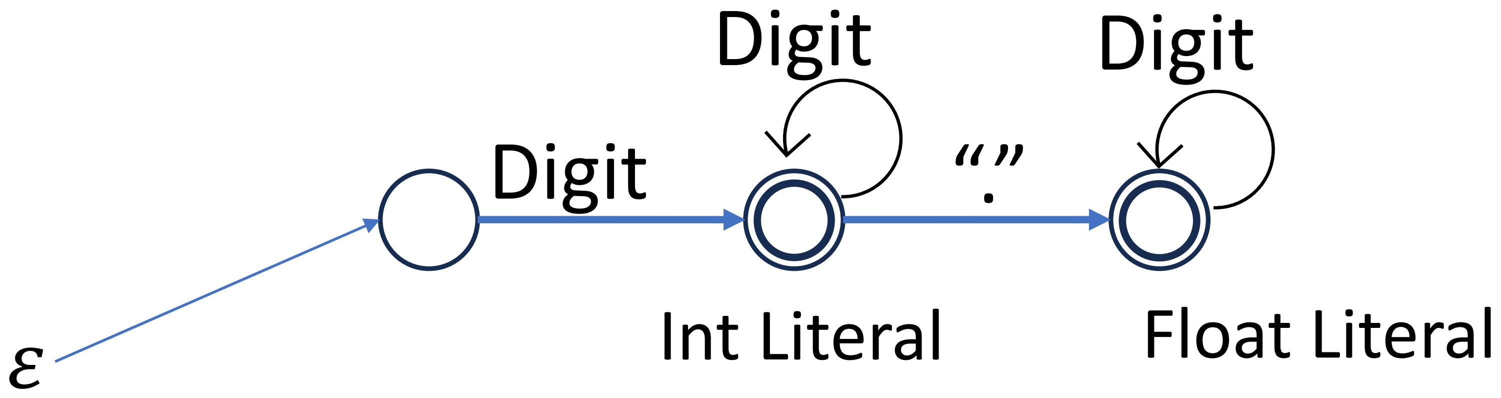
- Find the non-determinism after the initial ε-closure



Found it! How can we make this deterministic?



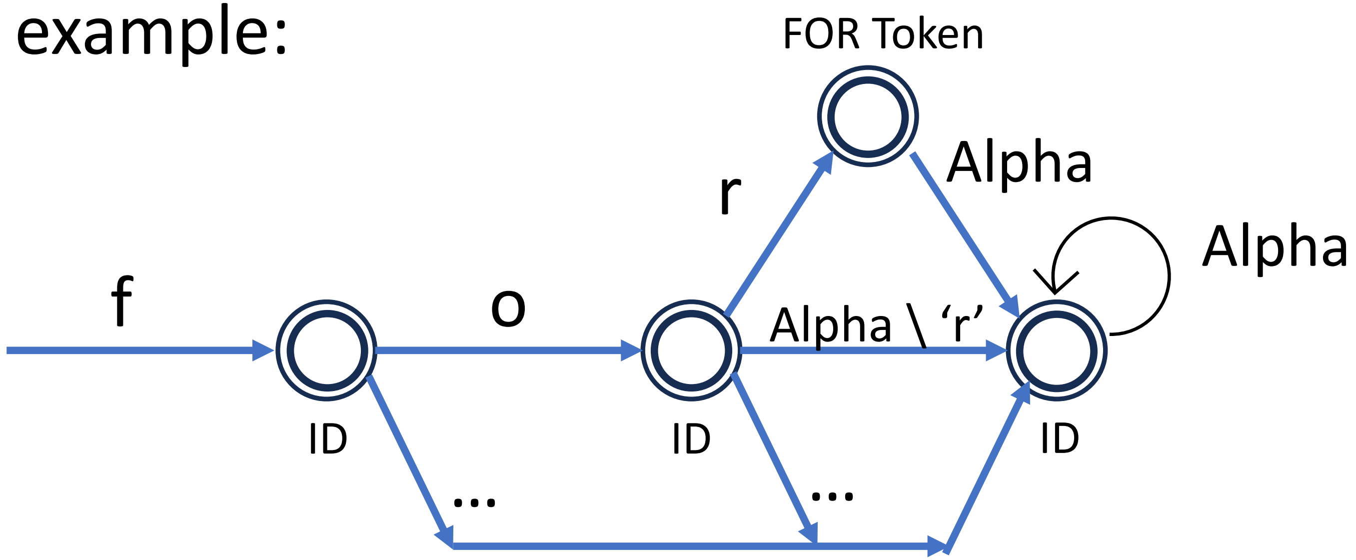
Combine Paths



Now, by combining this path, every path has a unique start, thus no longer non-deterministic

Apply a similar principal for reserved words

- Make the path that starts with “Alphabetic” branch off if the reserved word is encountered.
- Branch can rejoin “identifier” if more letters arrive
- Simplified example:



Non-deterministic unop vs binop

- Recall: `unop` and `binop` have a common operator
- Which one was it?

Non-deterministic unop vs binop

- Recall: `unop` and `binop` have a common operator
- For the Lexer, there is no deterministic way to find out (without making Lexing become Parsing):

Non-deterministic unop vs binop

- Recall: `unop` and `binop` have a common operator
- For the Lexer, there is no deterministic way to find out (without making Lexing become Parsing):

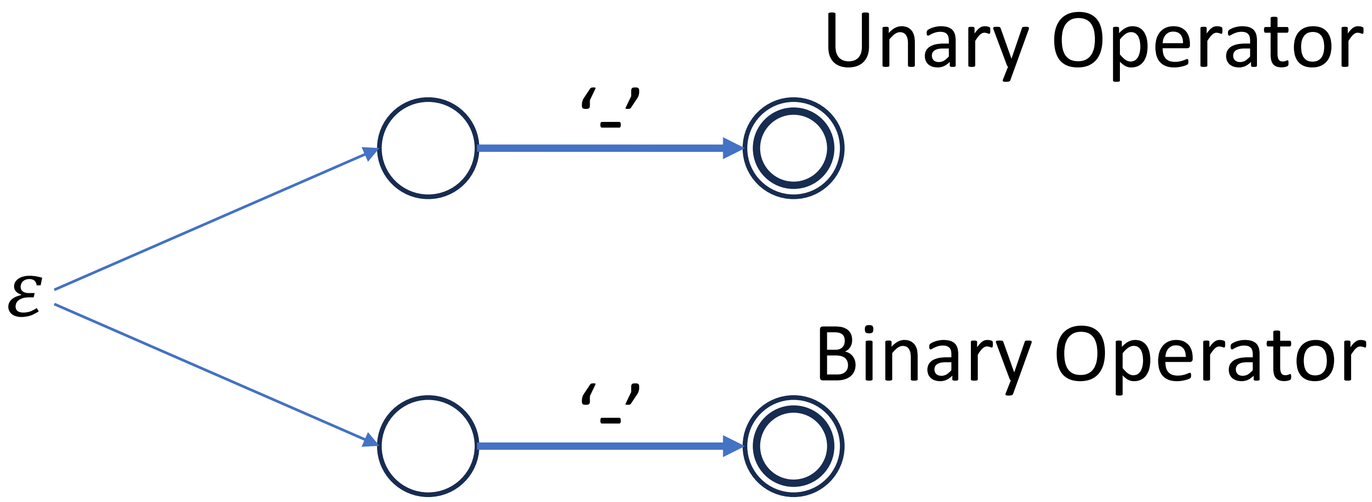
- Unop: { !, - }
- Binop: { -, +, *, /, &&, ||, … }

Non-deterministic unop vs binop

- Unop: { !, - }
- Binop: { -, +, *, /, &&, ||, … }

- Thus, these Lexer non-terminals (and Parser terminals) should be combined into a single "Operator" Token
- Operator: { !, -, +, *, /, … }

Otherwise…



Not scannable!

Non-determinism overview

- Too computationally expensive
- Great in theory, but we need to implement this

- Useful when deriving the DFA though!

EBNF Properties

Recap

- Nullable(A) ≡ …
- Starters(A) ≡ …
- Followers(A) ≡ …

Recap

- Nullable(A) ≡ A may result in ε

- Starters(A) ≡ …
- Followers(A) ≡ …

Recap

- Nullable(A) ≡ A may result in ε
- Starters(A) ≡ The set of terminals that indicate the current symbol may be A, and must include ε if Nullable(A)
- Followers(A) ≡ …

Recap

- Nullable(A) ≡ A may result in ε

- Starters(A) ≡ The set of terminals that indicate the current symbol may be A, and must include ε if Nullable(A)

- Followers(A) ≡ The set of terminals that follow A **after** A is parsed

To derive these, two new definitions

- For the sets S and T:
- $S \oplus T = \begin{cases} S & \text{if } \varepsilon \notin S \\ (S \setminus \{\varepsilon\}) \cup T & \text{if } \varepsilon \in S \end{cases}$
- Not to be confused with exclusive-or

To derive these, two new definitions

- For the sets S and T:
- $S \oplus T = \begin{cases} S & \text{if } \varepsilon \notin S \\ (S \backslash \{\varepsilon\}) \cup T & \text{if } \varepsilon \in S \end{cases}$
- Not to be confused with exclusive-or
- If A can derive ω: $A \Rightarrow^* \omega$
- Note, this definition looks at all possible derivations while $\oplus$ does not.

Start off with the easy one

- Shorthand, say N(A)≡ Nullable(A)

- Nullable(A)=$\begin{cases} true & \text{if } A \Rightarrow^* \varepsilon \\ false & \text{otherwise} \end{cases}$

Nullable Inductive Structure

- Rules for induction (to determine “eventually can derive”)

Observed Sequence	Rule
1. $\alpha = \varepsilon$	Nullable(α) = true
2. $\alpha = t$, where t is terminal	Nullable(α) = false
3. $\alpha = A$, where A is non-terminal	Nullable(α) = Nullable(A)
4. $\alpha = \alpha_1\alpha_2 \dots \alpha_n$	Nullable(α) = Nullable(α_1) ∧ … ∧ Nullable(α_n)
5. $\alpha = \alpha_1 \vert \alpha_2 \vert \dots \vert \alpha_n$	Nullable(α) = Nullable(α_1) ∨ … ∨ Nullable(α_n)
6. $\alpha = \beta^*$	Nullable(α) = true

Is Nullable(S)?

- S ::= A$
- A ::= BDA | a
- B ::= D | b
- D ::= d | ε

The hard way

- S ::= A$
- A ::= BDA | a
- B ::= D | b
- D ::= d | ε

- D is nullable
- B can derive D, which is nullable
- A can derive BDA, and BD are nullable
- Oh, A is recursive
- A will eventually have ‘a’
- Thus Nullable(S)=false

The easy way

- S ::= A$
- A ::= BDA | a
- B ::= D | b
- D ::= d | ε

- Well there is a $ right there, so from Rule 4:

Nullable(A) ∧ Nullable($) = Nullable(A) ∧ **false**

- False!

The normal way (Remove $)

- S ::= A
- A ::= BDA | a
- B ::= D | b
- D ::= d | ε

- Nullable(A) = Nullable(BDA) ∨ Nullable(a)
- Nullable(A) = Nullable(BDA) ∨ false
- Nullable(A) =
  Nullable(B) ∧ Nullable(D) ∧ Nullable(A)
- Nullable(A) = true ∧ true ∧ Nullable(A)

The normal way (Remove $)

- S ::= A
- A ::= BDA | a
- B ::= D | b
- D ::= d | ε

- Nullable(A) = Nullable(BDA) ∨ Nullable(a)
- Nullable(A) = Nullable(BDA) ∨ false
- Nullable(A) =
  Nullable(B) ∧ Nullable(D) ∧ Nullable(A)
- Nullable(A) = true ∧ true ∧ Nullable(A)
- Break down into N_1(A)=N_2(BDA|a)
  until at some fixed point, no more induction:
- N_i(A)=N_{i+1}(a) = false
- Thus: true ∧ true ∧ … ∧ false = **false**

Starters(A)

- Starters(A) = $\{t \mid A \Rightarrow^* t\beta \text{ for some } \beta\} \cup \begin{cases} \{\varepsilon\} & \text{if Nullable}(A) \\ \{\} & \text{otherwise} \end{cases}$

- Will need to define inductive structure

Starters(A)

- Starters(A) = $\{t \mid A \Rightarrow^* t\beta \text{ for some } \beta\} \cup \begin{cases} \{\varepsilon\} & \text{if Nullable}(A) \\ \{\} & \text{otherwise} \end{cases}$

Observed Sequence	Rule
1. $\alpha = \varepsilon$	Starters(α) = { ε }
2. $\alpha = t$, where t is terminal	Starters(α) = { t }
3. $\alpha = A$, where A is non-terminal	Starters(α) = Starters(A)
4. $\alpha = \alpha_1\alpha_2 \dots \alpha_n$	Starters(α) = Starters(α_1)$\bigoplus$Starters($\alpha_2 \dots \alpha_n$)
5. $\alpha = \alpha_1 \vert \alpha_2 \vert \dots \vert \alpha_n$	Starters(α) = Starters(α_1) ∪ … ∪ Starters(α_n)
6. $\alpha = \beta^*$	Starters(α) = Starters(β) ∪ { ε }

Followers(A)

- Followers(A) = $\{t \mid S \Rightarrow^* \alpha A t \beta\} \cup \begin{cases} \{\varepsilon\} & \text{if } S \Rightarrow^* \alpha A \\ \{\} & \text{otherwise} \end{cases}$

- Will need induction

Followers(A)

- Followers(A) = $\{t \mid S \Rightarrow^* \alpha A t \beta\} \cup \begin{cases} \{\varepsilon\} & \text{if } S \Rightarrow^* \alpha A \\ \{\} & \text{otherwise} \end{cases}$

- Induction:
- $FL_0(A) = \left(\bigcup_{C \Rightarrow \alpha A \beta} \text{Starters}(\beta)\right) \setminus \{\varepsilon\}$
- $FL_{i+1}(A) = FL_i(A) \cup \cdots \cup FL_i(C)$
  - Where $C \Rightarrow \alpha A \beta$ and Nullable(β)

Best shown through example..

S ::= A$
A ::= BDA | a
B ::= D | b
D ::= d | ε

- $FL_0(S)$= {}
- $FL_0(A)$= $(ST(\$) \cup ST(\varepsilon)) - \{\varepsilon\}$
  - Note still in 0^{th} iteration
- $FL_0(B)$= $ST(DA)$ - $\{\varepsilon\}$
- Because ST(DA) = ST(D)⊕ST(A)

- Need to compute Starters!

- S is not in the RHS of any rule
- = {$}
  - A is in RHS of 1^{st} and 2^{nd} rule

$S ::= A\$$
$A ::= BDA \mid a$
$B ::= D \mid b$
$D ::= d \mid \varepsilon$

Starters(A)

- $ST(A) = ST(BDA \mid a) = (ST(B) \oplus ST(D) \oplus ST(A)) \cup ST(a)$
- Need $ST(B) = ST(D) \cup ST(b) = ST(D) \cup \{b\}$
- Need $ST(D) = ST(d) \cup ST(\varepsilon) = \{d, \varepsilon\}$
- Thus: $ST(B) = \{d, \varepsilon\} \cup \{b\} = \{b, d, \varepsilon\}$

- $ST(A) = \big((ST(B) - \{\varepsilon\}) \cup ST(D) \big) \oplus ST(A) \cup \{a\}$
  - Simplification of $\oplus$
- $ST(A) = \{b, d, \varepsilon\} \oplus ST(A) \cup \{a\}$

S ::= A$
A ::= BDA | a
B ::= D | b
D ::= d | ε

Starters(A)

- ST(A) = {b,d,ε} ⨁ ST(A) ∪ {a}
- Iterate until A eventually generates {a}

- {b,d,ε}⨁{a}
- Thus, Starters(A) = {a,b,d}

$$
\begin{aligned}
S &::= A\$ \\
A &::= BDA \mid a \\
B &::= D \mid b \\
D &::= d \mid \varepsilon
\end{aligned}
$$

- $FL_0(S)$= {}
- $FL_0(A)$= $(ST(\$) \cup ST(\varepsilon)) - \{\varepsilon\}$
  - Note still in 0^{th} iteration
- $FL_0(B)$= $ST(DA)$ - $\{\varepsilon\}$
  - Because ST(DA) = ST(D)⨁ST(A)
- $FL_0(D)$= (ST(A) ∪ ST(ε)) – $\{\varepsilon\}$

- S is not in the RHS of any rule
- = {\$}
  - A is in RHS of 1^{st} and 2^{nd} rule
- = {a,b,d}
  - See previous slides
- = {a,b,d}
  - D is in RHS of 2^{nd} and 3^{rd} rule

Example skipped indices

- Because index can be large when inducting in an unintuitive way
- Some grammars will reach a fixed point no matter what

Why generate Follower Sets?

- Consider BlockStmt ::= { Statement* }

- What is Followers(Statement*)?

Why generate Follower Sets?

- Consider BlockStmt ::= { Statement* }
- What is Followers(Statement*)?
- That means we can keep taking in statements unless we see }

```
while( currentToken != RightCurlyBrace ) {
   parseStatement();
}
```

Final Property! Predict(A)

- For each choice in A ::= $\beta(\alpha_1| \ldots |\alpha_m)\gamma$ define…

$$\text{Predict}(\alpha_i) = \text{Starters}(\alpha_i\gamma) \oplus \text{Followers(A)}$$

- For repetitions in A ::= $\beta(\alpha)^*\gamma$

$$\text{Predict}(\alpha) = \text{Starters}(\alpha)$$

$$\text{Predict}(\gamma) = \text{Starters}(\gamma) \oplus \text{Followers(A)}$$

Can predict γ by its starters, or if γ is Nullable, then the followers of A as well will predict leaving the sequence $(\alpha)^*$

- For sequences A ::= $\alpha\beta\gamma$

$$\text{Predict(A)} = \text{Starters}(\alpha) \oplus \text{Starters}(\beta) \oplus \text{Starters}(\gamma)$$

Final Property! Predict(A)

- For each choice in A ::= $\beta(\alpha_1 | \ldots | \alpha_m)\gamma$ define...

$$\text{Predict}(\alpha_i) = \text{Starters}(\alpha_i\gamma) \oplus \text{Followers(A)}$$

- For repetitions in A ::= $\beta(\alpha)^*\gamma$

$$\text{Predict}(\alpha) = \text{Starters}(\alpha)$$

$$\text{Predict}(\gamma) = \text{Starters}(\gamma) \oplus \text{Followers(A)}$$

Can predict γ by its starters, or if γ is Nullable, then the followers of A as well will predict leaving the sequence $(\alpha)^*$

- For sequences A ::= $\alpha\beta\gamma$

$$\text{Predict(A)} = \text{Starters}(\alpha) \oplus \text{Starters}(\beta) \oplus \text{Starters}(\gamma)$$

Final Property! Predict(A)

- For each choice in A ::= $\beta(\alpha_1| \dots |\alpha_m)\gamma$ define…

$$\text{Predict}(\alpha_i) = \text{Starters}(\alpha_i\gamma) \oplus \text{Followers(A)}$$

- For repetitions in A ::= $\beta(\alpha)^*\gamma$

$$\text{Predict}(\alpha) = \text{Starters}(\alpha)$$

$$\text{Predict}(\gamma) = \text{Starters}(\gamma) \oplus \text{Followers(A)}$$

Can predict γ by its starters, or if γ is Nullable, then the followers of A as well will predict leaving the sequence $(\alpha)^*$

- For sequences A ::= $\alpha\beta\gamma$

$$\text{Predict(A)} = \text{Starters}(\alpha) \oplus \text{Starters}(\beta) \oplus \text{Starters}(\gamma)$$

LL(1) Condition

- All prediction sets are disjoint (including ones generated from within a single rule)

- For repetition $(\alpha)^*$, then α must not be nullable and

Starters(α) and (Starters(γ)$\bigoplus$Followers(A)) must be disjoint

$$\text{Predict}(\alpha) = \text{Starters}(\alpha)$$

$$\text{Predict}(\gamma) = \text{Starters}(\gamma) \bigoplus \text{Followers(A)}$$

Example: Is it LL(1)?

- S ::= A$
- A ::= BDA | a
- B ::= D | b
- D ::= d | ε

Is it LL(1)?

- S ::= A\$
- A ::= BDA | a
- B ::= D | b
- D ::= d | ε

- Identify choice and repetition points

Is it LL(1)?

- S ::= A\$
- A ::= BDA | a
- B ::= D | b
- D ::= d | ε

- Identify choice and repetition points

Predict(BDA)=Starters(B)$\oplus$Starters(D)$\oplus$Starters(A)

$= \{b,d,\varepsilon\} \oplus \{d,\varepsilon\} \oplus \{a,b,d\} = \{a,b,d\}$

Predict(a) = {a}

Is it LL(1)?

- S ::= A$
- A ::= BDA | a
- B ::= D | b
- D ::= d | ε

- Identify choice and repetition points

Predict(BDA)=Starters(B)$\oplus$Starters(D)$\oplus$Starters(A)

$= \{b,d,\varepsilon\} \oplus \{d,\varepsilon\} \oplus \{a,b,d\} = \{a,b,d\}$

Predict(a) = {a}

NOT DISJOINT!

Is it LL(1)?

- S ::= A$
- A ::= BDA | a
- B ::= D | b
- D ::= d | ε

- Identify choice and repetition points

Predict(D) = Starters(D) $\oplus$ Followers(B)

= $\{d,\varepsilon\} \oplus \{a,b,d\}$ = {a,b,d}

Predict(b) = {b}

Not Disjoint!

Wrap-up

- Predict sets useful for determining what rule you are in
- Useful for determining LL(1) condition

- With this, you can deterministically figure out which rule you are in (assuming the language allows for such)

- Written Assignment 1 will go over the PA-1 theory
- Midterms contain a mix of PA-1 implementation questions and theory

End