BOTTOM UP PARSING
Why? - Question1

2. Consider the language **L** $=\{u^n z^m \mid \mathbf{n > m}\}$

a) Give a grammar G such that L(G) = **L**

b) Is G $\in$LL(1) ?

c) Have transformations of G, if any, predictive parsers ?

Consider string $\gamma=u^{10}z^4$: γ is a string of L, because 10>4 satisfies the condition for inclusion, i.e n>m. Noting that, in order to conclude that $\gamma \in$L, we computed the occurrences of u, those of z and then, we compared the two values. Such arguments cannot be used in syntactic analyzers, that are very elementary structures (compared to those for general programming). Hence, the question is:

How can a syntactic analyzer proceed in deciding about the inclusion of γ in L?

Remember that, an analyser is moving left-to-right on the string, one symbol a time. So, now, the question is:

What has to do analyzer, when it reads (the first) u?

BOTTOM UP PARSING
Why? Question2

2. Consider the language $\mathbf{L} = \{u^n z^m \mid \mathbf{n} > \mathbf{m}\}$

. . . .

Remember that, an analyser is moving left-to-right on the string, one symbol a time. So, now, the question is:

What has to do analyzer, when it reads (the first) u?

Nothing else than store **u**'s in somewhere and, continue scanning until to the first **z**, if any. Then, for each **z** that is read, one stored **u** must be retrieved. When all **z**' s are paired to as many, previously stored, **u**'s, then, at least one **u** must be again in the store. So, now, the question is:

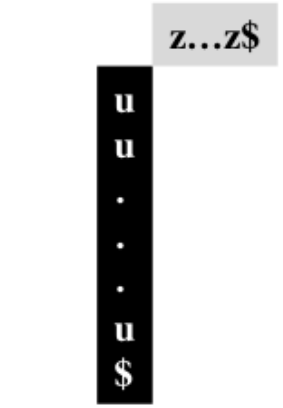
How can be expressed such a behavior, using analyzers?
And, using grammars?

BOTTOM UP PARSING
Why? Question3

2. Consider the language $\mathbf{L} = \{u^n z^m \mid \mathbf{n \geq m}\}$

. . . .

How can be expressed such a behavior, using analyzers?



After the storing of u

After the pairing of u and z

How can be expressed such a behavior, using grammars?

A::= u A B::= u B z
A::= u B B::= ε

BOTTOM UP PARSING
Why? The Answers

How can be expressed such a behavior, using grammars?

A::= u A B::= u B z
A::= u B B::= ε

Such a grammar is not LL(k) for any k because for each k, $u^k \in first_k(uA) \cap first_k(uB)$. So, no way for deterministic, leftmost derivations, that are looking for a limited lookahead. Trying to do it:

A => u A
or
A => u B

In contrast, rightmost derivation leads to the derivation below (obtained in reversed order):

u u u z <= u u u B z <= u u B <= u A <= A

BOTTOM UP PARSING

(now: => means $_r$=>, when omitted)

S::= abABe
A::= Abc | b
B::= d

abbbcde ∈L(S) ?

Reversed rigthmost

abbbcde<=abAbcde<=abAde<=abABe<=S

shift-reduce parsing (LR)

Reversed Reconstruction of a Right Derivation: How to do it?

To do it is needed to know what of the followings $\beta <= \alpha$:

abbbcde

$$\begin{aligned} \underline{a}bbbcde &\ {}_{A::=b}<= aAbbcde \\ a\underline{b}bbcde &\ {}_{A::=b}<= abAbcde \\ ab\underline{b}bcde &\ {}_{A::=b}<= abAcde \\ abbb\underline{c}de &\ {}_{B::=d}<= abbcBe \end{aligned}$$

is, in effect:

- involving **two Right Sentential Forms -** $\alpha, \beta \in RSF_G$
- (equally) a **Right Derivation -** $\alpha \ {}_r\!=\!> \beta$;
- (equally) a part of a (Reversed) **Right Star Derivation** from the start symbol - $S \ {}_r\!=\!>^* \alpha \ {}_r\!=\!> \beta$

BOTTOM UP PARSING

The Handle - The Viable Prefixes: The Process

Let $G \equiv <V, \Sigma, S, \Pi>$ be

Let $\gamma \equiv \gamma_1 \beta \gamma_2$ be in RSF.

$A ::= \beta$ is the Handle of γ *if and only if* $S \Rightarrow^* \gamma_1 A \gamma_2 \Rightarrow \gamma_1 \beta \gamma_2$

The 4 step Analysis Process

1) Scan the Right Sentential Form, γ, from left to right, one symbol a time, through (Viable) Prefixes of the Handle.

2) Stop when the Handle has been just, traversed.

3) Reduce it, thus obtaining a new RSF.

4) Repeat 1-3 until $\gamma = S$

BOTTOM UP PARSING

The Process

The Process Critical Point is Step 1

1) Scan the Right Sentential Form, from left to right, one symbol a time, through (Viable) Prefixes of the Handle. For

Two approaches for (unambiguous) grammars:

- **Backtrack** among (all) possible choises
- **Restrict** to the class of **Grammars** admitting:
  - **deterministic selection**
  - **in linear** space/time complexity

Such a class of good grammars exists and its name **is LR**

BOTTOM UP PARSING
LR Grammars include LL Grammars

LR Analyzers are based on a different kind of Push-Down Automata

driver D uses *shift* and *reduce* (state transition) operations

table M contains *states of items* and *handles*

LR is more powerful than LL:

It applies a rightmost reduction only after traversing the entire string to be replaced

apply it to the grammar below and to a string of your choise

S::= u S | u A
A::= u A z | u B z
B::= v B | v

LR Parsing

Mechanize the Handle Selection: Prefixes

Consider the grammar on the right and a string γ. Can γ have an handle that should be prefixed by α?

$\gamma \equiv vuuz, \quad \alpha \equiv \lambda$
$\gamma \equiv vuuz, \quad \alpha \equiv v$
$\gamma \equiv uuvz, \quad \alpha \equiv u$
$\gamma \equiv uuvz, \quad \alpha \equiv uu$

S::= u S | u A
A::= u A z | u B z
B::= v B | v

It means that

$\exists \beta\delta$: 1) $\gamma \equiv \alpha\beta\delta$; 2) $A ::= \beta \in \Pi$;
3) $S \underset{r}{\Rightarrow}^* \alpha A \delta \underset{r}{\Rightarrow} \alpha\beta\delta$

LR uses **prefixes**, like α, in order to **detect Handles**

LR Parsing
The set of (Viable) Prefixes is a Regular Language

Given a (LR) grammar G, the prefixes of the handles of RSF are called **viable prefixes**, and include the handle itself. The set of $\mathbf{VP_G}$ below, is a **Regular Language**

$$\mathbf{VP_G = \{prefix(\alpha\beta) \mid A ::= \beta \in \Pi_G,\ \alpha A \delta \Rightarrow \alpha\beta\delta \in RFS_G \text{ for some } \delta\}}$$

Hence VP_G has a Finite State Automaton that can recognize all and only strings $\alpha\beta$ whose terminal part is the handle, *if any*

It results in a table that will be used as the central core of the handle detection mechanization

LR Parsing
Three Different Parsers
Three Different Finite Automata for VP_G

SLR The simplest to construct:
- Compact FSA Tables
- Small set of **C**ontext **F**ree, **L**inear **T**ime Analyzable, Languages

LR Quite simple to construct:
- Large FSA Tables
- Largest set of **CF**, **LT** Analyzable, Languages

LALR A good compromise between the two, above:
- Same size of the SLR Tables
- Significantly large, set of **CF**, **LT** Aanalyzable, languages
- A bit intricate to construct

SLR Parsing

VP_G - Items of the LR(0) Collection

In order to detect **Viable Prefixes**, the grammar (normal) productions can be examined. The use of **Items** is fundamental in such an exam.

Let **A::=αβ** be a normal production. Then:

A::= α.β is a **LR(0) item.**

Moreover, let: S =>*γAδ => γ αβ δ. Then:

- α is a trait of a viable prefix γα
- A::= α.β is a **valid item** for γα

- **Valid Items** are collected into sets according to the prefixes that they can recognize;
- Each of such **sets is a state** of a finite state automaton of **VP_G**

SLR Parsing

The LR(0) Collection: State Closure

Two forms of closure (as for Dotted Automata):

- **State Closure**
- **State Transition Closure**

State Closure. Let I be a set of Valid Items. Then I is in the collection only if I=Closure(I)

$$\text{Closure}(I) =_{\min} I \cup \text{Closure}\{B ::= .\gamma \mid A ::= \alpha.B\beta \in I\}$$

Why? What is the rational of the requirement, above?

If $A ::= \alpha.B\beta$ is in I, then, for some ρ, it is considered valid for $\rho\alpha$. But $B ::= .\gamma$ too, is valid for $\rho\alpha$. As a matter of this fact:

if $S \Rightarrow^* \rho A\delta \Rightarrow \rho\,\alpha B\beta\,\delta$ *then* $S \Rightarrow^* \rho\,\alpha B\beta\,\delta \Rightarrow \rho\alpha\,\gamma\,\beta'\delta$

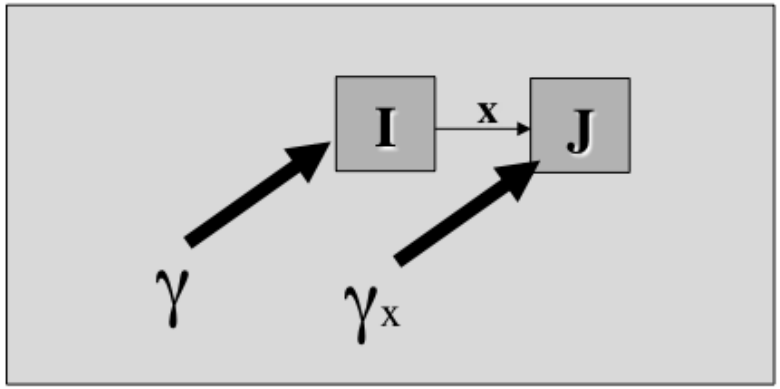
SLR Parsing

The LR(0) Collection: State Transition Closure

State Transition Closure. Let I be in the collection LR(0). Then all the outcomings of I are incomings of states of LR(0)

$$J = Goto(I,x) = Closure\{A ::= \alpha x.\beta \mid A ::= \alpha.x\beta \in I\}$$

Why? What is the rational of the requirement, abobe? - A graphical answer



SLR Parsing

The LR(0) Collection of G: The Initial State

The LR(0) Collection defines States and Transitions of the Viable Prefixes to SLR(1) parsing Handles.

The LR(0) Collection is a set of states of LR(0) Valid Items:

* Each state is closed w.r.t. a closure operation called **Closure**
* The transition set is closed w.r.t. a closure operation called **Goto**

The initial state of the LR(0) collection is **I0** and is defined below. Let G=<S,V,P,s> be the grammar. Then, the augmented grammar of G, is G'=<S∪{s'},V,P ∪{s'::=s},s'>.

I0=Closure({s'::=.s})

Example

Apply The construction of LR(0) collection to the grammar G

Grammar G

S::= aABe
A::= Abc | b
B::= d

Augmented Grammar G'

S'::=S
S::= aABe
A::= Abc | b
B::= d

Initial State I0

I0:Closure({S'::=.S})={S'::=.S
S::= .aABe}

Computation of the remaining states

Example

Computation of the remaining states

S'::=S
S::= aABe
A::= Abc | b
B::= d

- **Handle Items**. They have the dot just on the right end (here, are red marked)
- **Shift Items**. They have the dot just on the left of a terminal symbol.
- Handles always have an Handle Item at the end of the prefix.

I0:Closure({S'::=.S}) ={S'::=.S
S::= .aABe}
I1:Goto(I0,S) ={S'::=S.}
I2:Goto(I0,a) ={S::= a.ABe
A::=.Abc
A::=.b}
I3:Goto(I2,A) ={S::= aA.Be
A::=A.bc
B::= .d}
I4:Goto(I2,b) ={A::=b.}
I5:Goto(I3,B) ={S::= aAB.e}
I6:Goto(I3,b) ={A::=Ab.c}
I7:Goto(I3,d) ={B::= d.}
I8:Goto(I5,e) ={S::= aABe.}
I9:Goto(I6,c) ={A::=Abc.}

Example

The automaton of VP_G: Final States and Handles

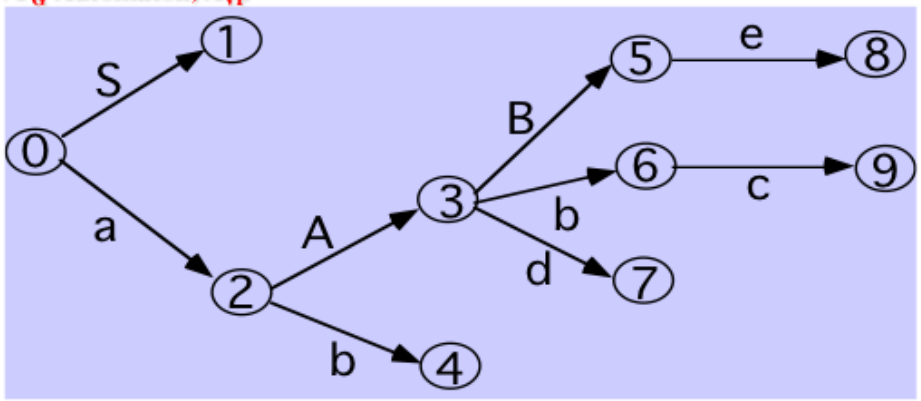
LR(0) Collection

I0:Closure({S'::=.S}) ={S'::=.S, S::= .aABe}
I1:Goto(I0,S) ={S'::=S.}
I2:Goto(I0,a) ={S::= a.Abe, A::=.Abc, A::=.b}
I3:Goto(I2,A) ={S::= aA.Be, A::=A.bc, B::= .d}
I4:Goto(I2,b) ={A::=b.}
I5:Goto(I3,B) ={S::= aAB.e}
I6:Goto(I3,b) ={A::=Ab.c}
I7:Goto(I3,d) ={B::= d.}
I8:Goto(I5,e) ={S::= aABe.}
I9:Goto(I6,c) ={A::=Abc.}

Augmented G'

S'::=S
S::= aABe
A::= Abc | b
B::= d

VP_G Automaton, A_{VP}



- Q: How a A_{VP} recognizes viable prefixes?
  - A: The set VP_G of the Viable Prefixes of the handle of G is just the language of A_{VP}, i.e. $L(A_{VP}) = VP_G$
- Q: Which are the final states of A_{VP}?
  - A: Each state is a final state. In particular, I0 is final for the prefix λ.
- Q: Which of the following is not a prefix: a) λ, b) a, c) ab, d) abb?
  - A: abb is not.
- Q: When we cannot continue to traversing the automaton:
  - -(1) we are on the right of the handle?
  - -(2) we have just passed the end of a prefix whose terminal trait is or is not the string handle depending from the symbol following the prefix
  - A: Answer can be obtained considering what happens in the following cases: a) ac, b) abd, c) abb.

Exercise: Use all the tools in this slide, and the answers to the questions above, to show the rightmost derivation of the string: abbcde
ab bcde$ <= aAbc de$ <= aAd e$ <= aABe $ <= S $ <= S'$

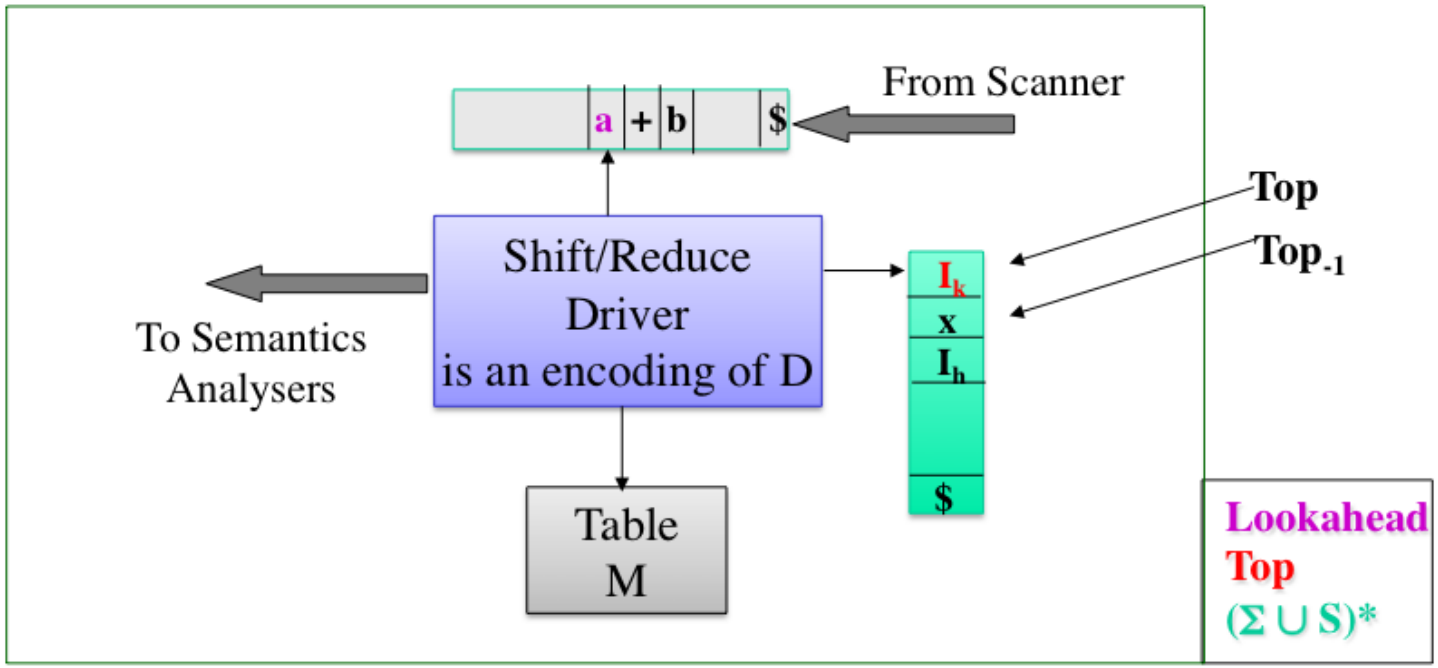
Push-Down Automata

are the perfect supports for SLR parsers

A **Pushdown Automaton** extends FSA and can be defined by the 6-tuple below:

<S, Σ, M: S x Σ --> S, D: M x (Σ ∪ S)* --> (Σ ∪ S)*, $s_0 \in S$, $F \subseteq S$>

where **S, Σ, M, s_0, F** are the same of FSA, while **(Σ ∪ S)*** is a stack.



Push-Down Automata

The definition of D for SLR(1) grammars

The function D for SLR(1)

- **Shift(k)** = push(lookahead); push(I_k)
- **Reduce(A::= α)** = popn(2*|α|);
  push(A);
  push(Goto(Top_{-1},A));

where **M≡<Action,Goto>**