BOTTOM UP PARSING
Why? - Question1

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

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

   b) Is G ∈LL(1) ?

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

Consider string $\boldsymbol{\gamma = u^{10} z^4}$: 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 **γ** ∈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 an 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 must doing analyzer, when reads the first u?

BOTTOM UP PARSING
Why? Question2

2. Consider the language **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 must doing analyzer, when reads the first u?

Nothing else than store it in somewhere and, continue scanning until to the first **z**, if any. Then, for each **z** that is read, one **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:

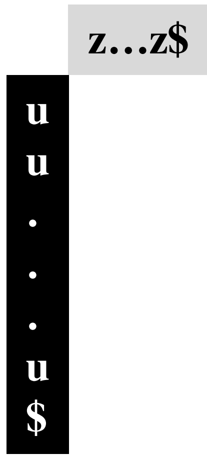
How can be expressed such a behavior, using analyzers?
And, using grammars?

BOTTOM UP PARSING
Why? Question3

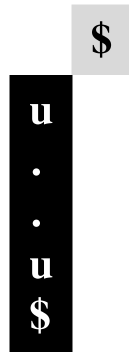
2. Consider the language **L** $=\{u^n z^m \mid \mathbf{n \geq m}\}$

. . . .

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 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

ab<u>b</u>bcde<=ab<u>Abc</u>de<=abA<u>d</u>e<=<u>abABe</u><=S

shift-reduce parsing (LR)

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

To do it means to know what of the followings β <= α :

abbbcde

a$\underline{b}$bbcde	$_{A::=b}$<=	aAbbcde
ab$\underline{b}$bcde	$_{A::=b}$<=	abAbcde
abb$\underline{b}$cde	$_{A::=b}$<=	abAcde
abbbc$\underline{d}$e	$_{B::=d}$<=	abbcBe

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 $G \equiv \langle V, \Sigma, S, \Pi \rangle$ 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 steps Analysis Process

1) Scan the Right Sentential Form, 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) Then repeat 1-3.

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

tabella 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) $A ::= \beta\ \Pi$;
3) $S \;{}_r\!\!\Rightarrow^* \alpha A \delta \;{}_r\!\!\Rightarrow \alpha\beta\delta$

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**

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

Hence VP_G has 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