Bottom-Up Parser

A bottom-up parser constructs the parse tree from the leaves to the root.

> It starts with the input string (tokens).

> Gradually reduces it to the start symbol of the grammar using productions in reverse.

> This is why it's often called a reduce-based parser.

> Trace the right-most derivation

> Example tools: YACC, Bison.

Ex: $S \rightarrow aABe$ i/p: abbcde

$A \rightarrow Abc \mid b$

$B \rightarrow d$

Shift-reduce parsing

> Uses a stack + input buffer.

> Two main operations:

Shift → push next input symbol onto stack.

Reduce → replace RHS of a production by LHS.

> Parsing ends when:

Stack contains only the start symbol

Input buffer is empty.

> Idea is to shift some symbols of input to the stack until we can find a situation where reduction can be applied.

> At each reduction step, a specific substring matching the body of the production is replaced by the non-terminal at the head of the production.

> Key decisions:

- **When to reduce**
- **What production to apply**

Right sentential Form

> A sentential form is any string derived from the start symbol of a grammar, using grammar rules (it may contain both terminals and non-terminals).

> A right sentential form is a specific type of sentential form obtained by a rightmost derivation.

> That means at each step, the rightmost non-terminal is replaced during the derivation.

> So, if a grammar generates a string using rightmost derivations, every intermediate string is called a right sentential form.

Handle

A handle of a right sentential form is a substring that:

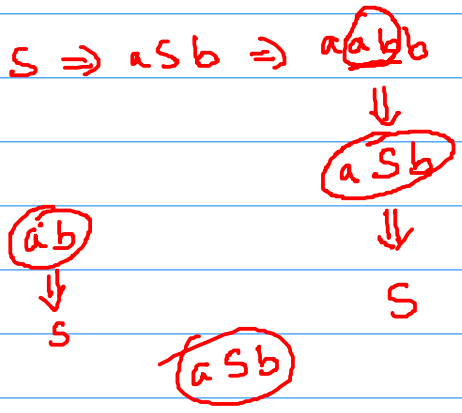
- Matches the RHS of a production rule.

- Its reduction to the LHS represents one step of the reverse of a rightmost derivation.

S→aSb | ab **String: aabb**

Right sentential form

Rightmost derivation (forward): S⇒aSb⇒aabb



Reverse (for parsing, i.e. reductions):

aabb → reduce ab to S: handle = ab.

Now we get aSb.

Reduce aSb to S.

So, the handles encountered are:

> ab

> aSb

Implementing Shift reduce parser

Parser Actions

Shift → push the next input symbol onto the stack.

Reduce → if the top of the stack matches RHS of a production, replace it with LHS.

Accept → if stack contains only the start symbol and input buffer is empty.

Error → if no valid shift/reduce is possible.

E → E + T | T
T → id

id + id

Stack	Input	Action
$	id + id $	Shift
$ id	+ id $	Reduce T→id
$ T	+ id $	Reduce E→T
$ E	+ id $	Shift
$ E +	id $	Shift
$ E + id	$	Reduce T→id
$ E + T	$	Reduce E→E+T
$ E	$	Accept