Left recursion by recursive ascent

Roman Redziejowski

Concurrency, Specification and Programming
Berlin 2021

Context-free grammar

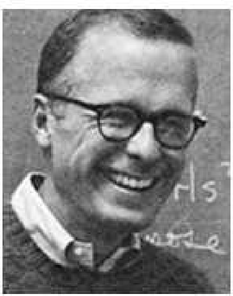
$A \rightarrow a\ B \mid a$
$B \rightarrow b\ A \mid b\ B \mid b$

Context-free grammar

A $\rightarrow$ a B | a
B $\rightarrow$ b A | b B | b

A $\rightarrow$ A1 | a
A1 $\rightarrow$ a B
B $\rightarrow$ B1 | B2 | b
B1 $\rightarrow$ b A
B2 $\rightarrow$ b B

BNF: Backus Naur Form



John Backus

Peter Naur

Derivation

A $\rightarrow$ A1 | a
A1 $\rightarrow$ a B
B $\rightarrow$ B1 | B2 | b
B1 $\rightarrow$ b A
B2 $\rightarrow$ b B

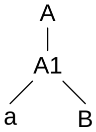
A

Derivation

A → A1 | a
A1 → a B
B → B1 | B2 | b
B1 → b A
B2 → b B

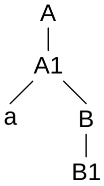
Derivation

A $\rightarrow$ A1 | a
A1 $\rightarrow$ a B
B $\rightarrow$ B1 | B2 | b
B1 $\rightarrow$ b A
B2 $\rightarrow$ b B



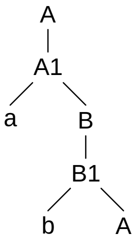
Derivation

A $\rightarrow$ A1 | a
A1$\rightarrow$ a B
B $\rightarrow$ B1 | B2 | b
B1$\rightarrow$ b A
B2$\rightarrow$ b B



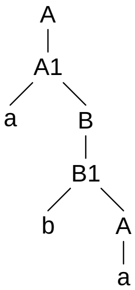
Derivation

A $\rightarrow$ A1 | a
A1 $\rightarrow$ a B
B $\rightarrow$ B1 | B2 | b
B1 $\rightarrow$ b A
B2 $\rightarrow$ b B



Derivation

A $\rightarrow$ A1 | a
A1$\rightarrow$ a B
B $\rightarrow$ B1 | B2 | b
B1$\rightarrow$ b A
B2$\rightarrow$ b B



aba

Parsing: recovering the syntax tree

A → A1 | a
A1→ a B
B → B1 | B2 | b
B1→ b A
B2→ b B

A
|
?

aba

Parsing: recovering the syntax tree

A $\rightarrow$ A1 | a
A1 $\rightarrow$ a B
B $\rightarrow$ B1 | B2 | b
B1 $\rightarrow$ b A
B2 $\rightarrow$ b B

Recursive descent:
Transcribe each line into procedure

aba

Parsing: recovering the syntax tree

A $\rightarrow$ A1 | a
A1$\rightarrow$ a B
B $\rightarrow$ B1 | B2 | b
B1$\rightarrow$ b A
B2$\rightarrow$ b B

Recursive descent:
Transcribe each line into procedure
returning a partial tree.

aba

A → A1 | a
A1→ a B
B → B1 | B2 | b
B1→ b A
B2→ b B

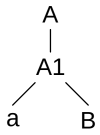
A
|
A1

A:
call A1; return tree (A △ A1); or
call a; return tree (A △ a);

aba

Recursive descent

A $\rightarrow$ A1 | a
A1 $\rightarrow$ a B
B $\rightarrow$ B1 | B2 | b
B1 $\rightarrow$ b A
B2 $\rightarrow$ b B

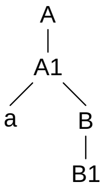


A1:
call a; call B; return tree (A1 $\triangle$ a B);

aba

Recursive descent

A $\rightarrow$ A1 | a
A1$\rightarrow$ a B
B $\rightarrow$ B1 | B2 | b
B1$\rightarrow$ b A
B2$\rightarrow$ b B



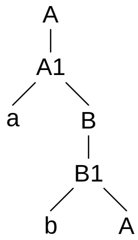
B:
call B1; return tree (B $\triangle$ B1); or
call B2; return tree (B $\triangle$ B2); or
call b; return tree (B $\triangle$ b);

aba

Recursive descent

A → A1 | a
A1→ a B
B → B1 | B2 | b
B1→ b A
B2→ b B

B1:
call b; call A; return tree (B1 △ b A);

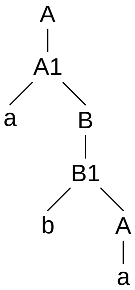


aba

Recursive descent

A $\rightarrow$ A1 | a
A1$\rightarrow$ a B
B $\rightarrow$ B1 | B2 | b
B1$\rightarrow$ b A
B2$\rightarrow$ b B

A:
call A1; return tree (A △ A1); or
call a; return tree (A △ a);



aba

Recursive descent parser

Grammar	Parsing procedure
A $\rightarrow$ A1 | a	A: call A1; return tree (A △ A1); or call a; return tree (A △ a);
A1$\rightarrow$ a B	A1: call a; call B; return tree (A1 △ a B);
B $\rightarrow$ B1 | B2 | b	B: call B1; return tree (B △ B1); or call B2; return tree (B △ B2); or call b; return tree (B △ b);
B1$\rightarrow$ b A	B1: call b; call A; return tree (B1 △ b A);
B2$\rightarrow$ b B	B2: call b; call B; return tree (B2 △ b B);

Another grammar: left-recursive

$A \rightarrow A1 \mid a$
$A1 \rightarrow B\ a$
$B \rightarrow B1 \mid B2 \mid b$
$B1 \rightarrow A\ b$
$B2 \rightarrow B\ b$

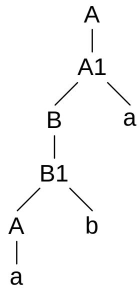
Another grammar: left-recursive

A $\rightarrow$ A1 | a
A1$\rightarrow$ B a
B $\rightarrow$ B1 | B2 | b
B1$\rightarrow$ A b
B2$\rightarrow$ B b

This grammar also derives ’aba’.

Another grammar: left-recursive

A $\rightarrow$ A1 | a
A1$\rightarrow$ B a
B $\rightarrow$ B1 | B2 | b
B1$\rightarrow$ A b
B2$\rightarrow$ B b



This grammar also derives ’aba’.

Parsing: recursive descent?

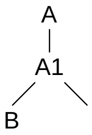
A $\rightarrow$ A1 | a
A1 $\rightarrow$ B a
B $\rightarrow$ B1 | B2 | b
B1 $\rightarrow$ A b
B2 $\rightarrow$ B b

A:
call A1; return tree (A $\triangle$ A1); or
call a; return tree (A $\triangle$ a);

aba

Recursive descent

A $\rightarrow$ A1 | a
A1$\rightarrow$ B a
B $\rightarrow$ B1 | B2 | b
B1$\rightarrow$ A b
B2$\rightarrow$ B b



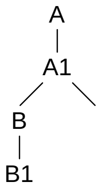
A1:
call B; call a; return tree (A1 $\triangle$ B a);

aba

Recursive descent

A $\rightarrow$ A1 | a
A1 $\rightarrow$ B a
B $\rightarrow$ B1 | B2 | b
B1 $\rightarrow$ A b
B2 $\rightarrow$ B b

B:
call B1; return tree (B $\triangle$ B1); or
call B2; return tree (B $\triangle$ B2); or
call b; return tree (B $\triangle$ b);

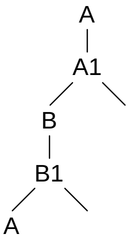


aba

Recursive descent

A → A1 | a
A1→ B a
B → B1 | B2 | b
B1→ A b
B2→ B b

B1:
call A; call b; return tree (B1 △ A b);

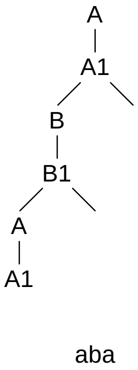


aba

Recursive descent

A → A1 | a
A1→ B a
B → B1 | B2 | b
B1→ A b
B2→ B b

A:
call A1; return tree (A △ A1); or
call a; return tree (A △ a);

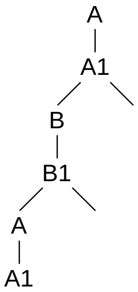


Recursive descent

A → A1 | a
A1→ B a
B → B1 | B2 | b
B1→ A b
B2→ B b

A:
call A1; return tree (A △ A1); or
call a; return tree (A △ a);

But A1 never returns!

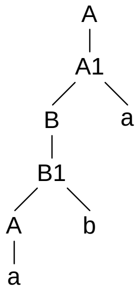


aba

Down to infinity

Idea

A → A1 | a
A1→ B a
B → B1 | B2 | b
B1→ A b
B2→ B b

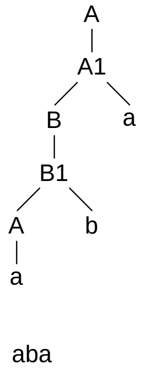


Reconstruct the tree bottom up!

aba

Idea

A $\rightarrow$ A1 | a
A1$\rightarrow$ B a
B $\rightarrow$ B1 | B2 | b
B1$\rightarrow$ A b
B2$\rightarrow$ B b



Reconstruct the tree bottom up!

Orlando Hill 'Support for Left-Recursive PEGs',
//https://github.com/orlandohill/peg-left-recursion.

Recursive ascent

A $\rightarrow$ A1 | a
A1$\rightarrow$ B a
B $\rightarrow$ B1 | B2 | b
B1$\rightarrow$ A b
B2$\rightarrow$ B b

A:
call a; plant a; call $A; or
call b; plant b; call $B;

($A adds parent A of a to tree)
($B adds parent B of b to tree)

a

aba

Recursive ascent

A $\rightarrow$ A1 | a
A1 $\rightarrow$ B a
B $\rightarrow$ B1 | B2 | b
B1 $\rightarrow$ A b
B2 $\rightarrow$ B b

$A:
build (A $\triangle$ a); return A or call $B1;

($B1 adds parent B1 of A to tree)

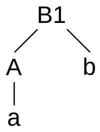
aba

Recursive ascent

A $\rightarrow$ A1 | a
A1$\rightarrow$ B a
B $\rightarrow$ B1 | B2 | b
B1$\rightarrow$ A b
B2$\rightarrow$ B b

$B1:
call b; build (B1 △ A b); call $B;

($B adds parent B of B1 to tree)

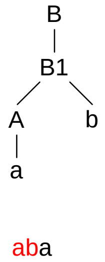


aba

$A \to A1 \mid a$
$A1 \to B\ a$
$B \to B1 \mid B2 \mid b$
$B1 \to A\ b$
$B2 \to B\ b$

$B:
build (B $\triangle$ B1); call $A1 or $B2;

($A1 adds parent A1 of B to tree)
($B2 adds parent B2 of B to tree)

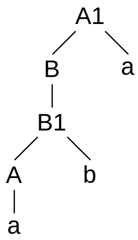


Recursive ascent

A $\rightarrow$ A1 | a
A1$\rightarrow$ B a
B $\rightarrow$ B1 | B2 | b
B1$\rightarrow$ A b
B2$\rightarrow$ B

$A1:
call a; build (A1 $\triangle$ B a); call $A;

($A adds parent A of A1 to tree)

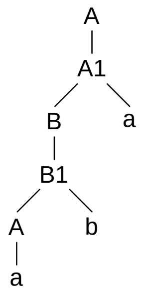


aba

Recursive ascent

A $\rightarrow$ A1 | a
A1$\rightarrow$ B a
B $\rightarrow$ B1 | B2 | b
B1$\rightarrow$ A b
B2$\rightarrow$ B b

$A:
add tree (A △ a); return A or call $B1;



aba

Dual grammar

Procedures	Grammar
A: call a; plant a; call \$A; or call b; plant b; call \$B;	A $\rightarrow$ a \$A | b \$A
\$A: build (A $\triangle$ a); call \$B1 or return tree A;	\$A $\rightarrow$ \$B1 | ε
\$A1: call a; build (A1 $\triangle$ B a); call \$A;	\$A1 $\rightarrow$ a \$A
\$B: build (B $\triangle$ B1); call \$A1 or \$B2;	\$B $\rightarrow$ \$A1 | \$B2
\$B1: call b; build (B1 $\triangle$ A b); call \$B;	\$B1 $\rightarrow$ b \$B
\$B2: call b; build (B2 $\triangle$ B b); call \$B;	\$B2 $\rightarrow$ b \$B

Dual grammar

A → a \$A | b \$B
\$A → \$B1 | ε
\$A1→ a \$A
\$B → \$A1 | \$B1
\$B1→ b \$B
\$B2→ b \$B

Dual grammar

A $\rightarrow$ a \$A | b \$B
\$A $\rightarrow$ \$B1 | ε
\$A1$\rightarrow$ a \$A
\$B $\rightarrow$ \$A1 | \$B1
\$B1$\rightarrow$ b \$B
\$B2$\rightarrow$ b \$B

Is not left recursive.

Dual grammar

A $\rightarrow$ a \$A | b \$B
\$A $\rightarrow$ \$B1 | ε
\$A1$\rightarrow$ a \$A
\$B $\rightarrow$ \$A1 | \$B1
\$B1$\rightarrow$ b \$B
\$B2$\rightarrow$ b \$B

Is not left recursive.

Recursive descent parser
reconstructs syntax tree for the original grammar.

General case: some terminology

For $A \rightarrow e_1 | \ldots | e_n$: $A \xrightarrow{\text{parent}} e_i$ for $1 \leq i \leq n$.

For $A \rightarrow e_1 \ldots e_n$: $A \xrightarrow{\text{parent}} e_1$.

General case: some terminology

For $A \to e_1 | \ldots | e_n$: $\quad A \xrightarrow{\text{parent}} e_i$ for $1 \le i \le n$.

For $A \to e_1 \ldots e_n$: $\quad A \xrightarrow{\text{parent}} e_1$.

A is *recursive* if $A \xrightarrow{\text{parent}} A_1 \xrightarrow{\text{parent}} A_2 \xrightarrow{\text{parent}} \ldots \xrightarrow{\text{parent}} A$.

General case: some terminology

For $A \to e_1 | \dots | e_n$: $\quad A \stackrel{\text{parent}}{\longrightarrow} e_i$ for $1 \le i \le n$.

For $A \to e_1 \dots e_n$: $\quad A \stackrel{\text{parent}}{\longrightarrow} e_1$.

A is *recursive* if $A \stackrel{\text{parent}}{\longrightarrow} A_1 \stackrel{\text{parent}}{\longrightarrow} A_2 \stackrel{\text{parent}}{\longrightarrow} \dots \stackrel{\text{parent}}{\longrightarrow} A$.

exit: recursive $A \to e_1 | \dots | e_n$ where at least one e_i is not recursive.

General case: some terminology

For $A \to e_1 | \dots | e_n$: $A \stackrel{\text{parent}}{\longrightarrow} e_i$ for $1 \le i \le n$.

For $A \to e_1 \dots e_n$: $A \stackrel{\text{parent}}{\longrightarrow} e_1$.

A is *recursive* if $A \stackrel{\text{parent}}{\longrightarrow} A_1 \stackrel{\text{parent}}{\longrightarrow} A_2 \stackrel{\text{parent}}{\longrightarrow} \dots \stackrel{\text{parent}}{\longrightarrow} A$.

exit: recursive $A \to e_1 | \dots | e_n$ where at least one e_i is not recursive.

seed: such e_i.

General case: Entry

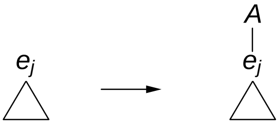
A:
call S_1; add tree S_1; call \$$X_1$; or
  . . .
call S_n; add tree S_n; call \$$X_n$;

$A \rightarrow S_1\,\$X_1 \mid \ldots \mid S_n\,\$X_n$

where $S_1, \ldots, S_n$ are all seeds and X_i is the exit containing S_i.

General case: \$A for choice

$\$A$ adds A on top of its child and continues with own parent.
$A \rightarrow e_1 | \ldots | e_j | \ldots | e_m$



$\$A$:
build $(A \triangle e_j)$; call $\$P_1$ or ... or $\$P_n$;

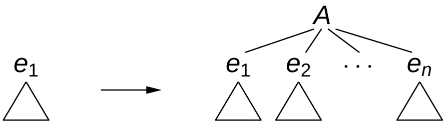
$\$A \rightarrow \$P_1 | \ldots | \$P_n$

where P_i are all parents of A
(Add ε after P_n if A is the entry.)

General case: \$A for sequence

\$$A$ adds A on top of its child and continues with own parent.

$A \rightarrow e_1\, , e_2 \ldots e_n$



\$$A$:

call e_2; $\cdots$; call e_n; build $(A \triangle e_1 \ldots e_n)$; call \$$P_1$ or $\ldots$ or \$$P_n$;

$\$A \rightarrow e_2 \ldots e_n\, (\$P_1 | \ldots |\$P_n)$

where P_i are all parents of A

(Add ε after P_n if A is the entry.)

Summary: dual grammar

$A \to S_1\,\$X_1 \mid \ldots \mid S_n\,\$X_n$

For $A \to e_1|\ldots|e_m$: $\$A \to \$P_1|\ldots|\$P_n\ [|\varepsilon]$

For $A \to e_1 e_2$: $\$A \to e_2\,(\$P_1|\ldots|\$P_n\ [|\varepsilon])$

Dual grammar

Rconstructs syntax tree according to original gammar.

Is not left recursive
(unless original grammar has a cycle).

Derives the same strings as the original grammar.

Example

L $\rightarrow$ P . x | x
P $\rightarrow$ P (n) | L

L $\rightarrow$ L1 | x
L1 $\rightarrow$ P . x
P $\rightarrow$ P1 | L
P1 $\rightarrow$ P (n)

L $\rightarrow$ x \$L
\$L $\rightarrow$ \$P | ε
\$L1 $\rightarrow$. x \$L
\$P $\rightarrow$ \$L | \$P1
\$P1 $\rightarrow$ (n) \$P

(From S. Medeiros, F.Mascarenhas, R. Ierusalimschy,
'Recursion in Parsing Expression Grammars',
Science of Computer Programming 96, P2 (2014), pp. 177 - 190.)

Example

E → F n | n
F → E + a | G –
G → H m | E
H → G k

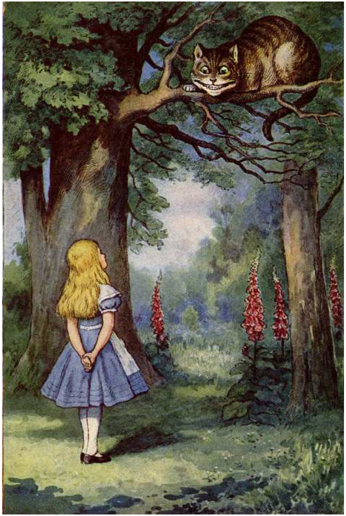
E → E1 | n
E1 → F n
F → F1 | F2
F1 → E + a
F2 → G –
G → G1 | E
G1 → H m
H → G k

E → n $E | ε
$E → $F1 | $G
$E1 → n $E
$F → $E1
$F1 → + a $F
$F2 → – $F
$G → $F2 | $H
$G1 → m $G
$H → k $G1

(From P. Sigaud 'Left Recursion'

https://github.com/PhilippeSigaud/Pegged/wiki/Left-Recursion)

Where to go?



$A \rightarrow e_1 | \ldots | e_n$
$A \rightarrow S_1 X_1 | \ldots | S_n X_n$
$\$A \rightarrow \$P_1 | \ldots | \$P_n$

– Looking at next letter?
– Limited backtracking?
– Full backtracking?
– ??

Choice of method depends on dual grammar.

Thanks

THANK YOU FOR YOUR PATIENCE!