Formal Languages
Simplifications of CFGs

A Substitution Rule

$$S \to aB$$
$$A \to aaA$$
$$A \to abBc$$
$$B \to aA$$
$$B \to b$$

Substitute

$$B \to b$$

Equivalent grammar

$$S \to aB \mid ab$$
$$A \to aaA$$
$$A \to abBc \mid abbc$$
$$B \to aA$$

A Substitution Rule

$$S \to aB \mid ab$$

$$A \to aaA$$

$$A \to abBc \mid abbc$$

$$B \to aA$$

Substitute

$$B \to aA$$

$$S \to \cancel{aB} \mid ab \mid aaA$$

$$A \to aaA$$

$$A \to \cancel{abBc} \mid abbc \mid abaAc$$

Equivalent grammar

In general:

$$A \to xBz$$

$$B \to y_1$$

Substitute $B \to y_1$

$$A \to xBz \mid xy_1z$$

equivalent grammar

Language?

Nullable Variables

$\lambda-\text{production:}$ $A \to \lambda$

Nullable Variable: $A \Rightarrow \ldots \Rightarrow \lambda$

Removing Nullable Variables

Example Grammar:

$$S \to aMb$$

$$M \to aMb$$

$$M \to \lambda$$

Nullable variable

$$S \to aMb$$

$$M \to aMb$$

$$\cancel{M \to \lambda}$$

Substitute

$$M \to \lambda$$

Final Grammar

$$S \to aMb$$

$$S \to ab$$

$$M \to aMb$$

$$M \to ab$$

Unit-Productions

Unit Production: $A \rightarrow B$

(single variables on both sides)

Removing Unit Productions

Observation:

$$A \to A$$

Is removed immediately

Example Grammar:

$$S \to aA$$

$$A \to a$$

$$A \to B$$

$$B \to A$$

$$B \to bb$$

$$S \to aA$$

$$A \to a$$

$$\cancel{A \to B}$$

$$B \to A$$

$$B \to bb$$

Substitute

$$A \to B$$

$$S \to aA \mid aB$$

$$A \to a$$

$$B \to A \mid B$$

$$B \to bb$$

$$S \to aA \mid aB$$

$$A \to a$$

$$B \to A \mid \cancel{B}$$

$$B \to bb$$

Remove

$$B \to B$$

$$S \to aA \mid aB$$

$$A \to a$$

$$B \to A$$

$$B \to bb$$

$$S \to aA \mid aB$$

$$A \to a$$

$$\cancel{B \to A}$$

$$B \to bb$$

Substitute

$$B \to A$$

$$S \to aA \mid aB \mid aA$$

$$A \to a$$

$$B \to bb$$

Remove repeated productions

$$S \to aA \mid aB \mid \cancel{aA}$$
$$A \to a$$
$$B \to bb$$

Final grammar

$$S \to aA \mid aB$$
$$A \to a$$
$$B \to bb$$

Language?

Useless Productions

$$S \to aSb$$

$$S \to \lambda$$

$$S \to A$$

$$A \to aA$$ Useless Production

Some derivations never terminate...

$$S \Rightarrow A \Rightarrow aA \Rightarrow aaA \Rightarrow \ldots \Rightarrow aa\ldots aA \Rightarrow \ldots$$

Another grammar:

$$S \to A$$

$$A \to aA$$

$$A \to \lambda$$

$$B \to bA$$ Useless Production

Not reachable from S

In general:

contains only terminals

if $S \Rightarrow \ldots \Rightarrow xAy \Rightarrow \ldots \Rightarrow w$

$w \in L(G)$

then variable A is useful

otherwise, variable A is useless

A production $A \to x$ is useless
if any of its variables is useless

Variables		Productions
	$S \to aSb$	
	$S \to \lambda$	
	$S \to A$	useless
useless	$A \to aA$	useless
useless	$B \to C$	useless
useless	$C \to D$	useless

Removing Useless Productions

Example Grammar:

$$S \to aS \mid A \mid C$$

$$A \to a$$

$$B \to aa$$

$$C \to aCb$$

Remove useless productions

First: find all variables that can produce strings with only terminals

$$S \to aS \mid A \mid C$$

$$A \to a$$

$$B \to aa$$

$$C \to aCb$$

Round 1: $\{A, B\}$

$$S \to A$$

Round 2: $\{A, B, S\}$

Keep only the variables
that produce terminal symbols: $\{A, B, S\}$

(other variables are useless)

$$S \rightarrow aS \mid A \mid \cancel{C}$$

$$A \rightarrow a$$

$$B \rightarrow aa$$

$$\cancel{C \rightarrow aCb}$$

$$\Longrightarrow$$

$$S \rightarrow aS \mid A$$

$$A \rightarrow a$$

$$B \rightarrow aa$$

Remove useless productions

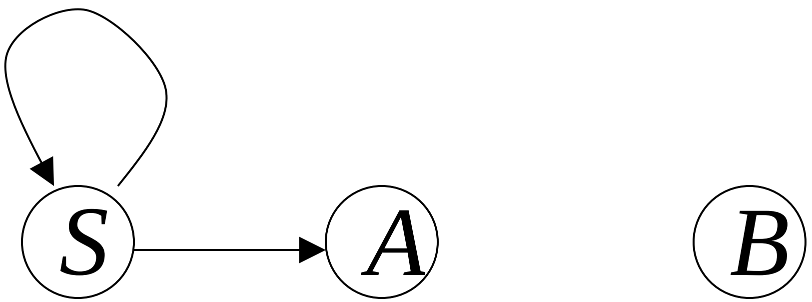
Second: Find all variables reachable from S

Use a Dependency Graph

$$S \to aS \mid A$$

$$A \to a$$

$$B \to aa$$



not reachable

Keep only the variables reachable from S

(the other variables are useless)

Final Grammar

$$S \to aS \mid A$$
$$A \to a$$
$$\cancel{B \to aa}$$

$$S \to aS \mid A$$
$$A \to a$$

Remove useless productions

Removing All

Step 1: Remove Nullable Variables

Step 2: Remove Unit-Productions

Step 3: Remove Useless Variables

Normal Forms for Context-free Grammars

Chomsky Normal Form

Each production has form:

$$A \rightarrow BC \qquad \text{or} \qquad A \rightarrow a$$

variable variable terminal

Examples:

$$S \to AS$$
$$S \to a$$
$$A \to SA$$
$$A \to b$$

Chomsky Normal Form

$$S \to AS$$
$$S \to AAS$$
$$A \to SA$$
$$A \to aa$$

Not Chomsky Normal Form

Conversion to Chomsky Normal Form

Example:

$$S \to ABa$$

$$A \to aab$$

$$B \to Ac$$

Not Chomsky Normal Form

Introduce variables for terminals: T_a, T_b, T_c

$$S \to ABa$$
$$A \to aab$$
$$B \to Ac$$

$$S \to ABT_a$$
$$A \to T_aT_aT_b$$
$$B \to AT_c$$
$$T_a \to a$$
$$T_b \to b$$
$$T_c \to c$$

Introduce intermediate variable: V_1

$$S \to ABT_a$$
$$A \to T_aT_aT_b$$
$$B \to AT_c$$
$$T_a \to a$$
$$T_b \to b$$
$$T_c \to c$$

$$S \to AV_1$$
$$V_1 \to BT_a$$
$$A \to T_aT_aT_b$$
$$B \to AT_c$$
$$T_a \to a$$
$$T_b \to b$$
$$T_c \to c$$

Introduce intermediate variable: V_2

$$S \to AV_1$$
$$V_1 \to BT_a$$
$$A \to T_aT_aT_b$$
$$B \to AT_c$$
$$T_a \to a$$
$$T_b \to b$$
$$T_c \to c$$

$$S \to AV_1$$
$$V_1 \to BT_a$$
$$A \to T_aV_2$$
$$V_2 \to T_aT_b$$
$$B \to AT_c$$
$$T_a \to a$$
$$T_b \to b$$
$$T_c \to c$$

Final grammar in Chomsky Normal Form:

Initial grammar

$$S \to ABa$$

$$A \to aab$$

$$B \to Ac$$

$$S \to AV_1$$

$$V_1 \to BT_a$$

$$A \to T_a V_2$$

$$V_2 \to T_a T_b$$

$$B \to AT_c$$

$$T_a \to a$$

$$T_b \to b$$

$$T_c \to c$$

In general:

From any context-free grammar
(which doesn't produce λ)
not in Chomsky Normal Form

we can obtain:

An equivalent grammar
in Chomsky Normal Form

The Procedure

First remove:

Nullable variables

Unit productions

Then, for every symbol a:

Add production $T_a \to a$

In productions: replace a with T_a

New variable: T_a

Replace any production $A \to C_1 C_2 \cdots C_n$

with
$$A \to C_1 V_1$$
$$V_1 \to C_2 V_2$$
$$\cdots$$
$$V_{n-2} \to C_{n-1} C_n$$

New intermediate variables: $V_1, V_2, \ldots, V_{n-2}$

Theorem: For any context-free grammar (which doesn't produce λ) there is an equivalent grammar in Chomsky Normal Form

Observations

- Chomsky normal forms are good for parsing and proving theorems

- It is very easy to find the Chomsky normal form for any context-free grammar

exercise

Find CNF for this grammar:

S -> 0A0 | 1B1 | BB
A -> C
B -> S | A
C -> S | epsilon

(exercise 7.1.3, Hopcroft, Motwani, Ullman)

Greibach Normal Form

All productions have form:

$$A \to a V_1 V_2 \cdots V_k \qquad k \geq 0$$

terminal variables

Examples:

$$S \to cAB$$

$$A \to aA \mid bB \mid b$$

$$B \to b$$

Greibach
Normal Form

$$S \to abSb$$

$$S \to aa$$

Not Greibach
Normal Form

Conversion to Greibach Normal Form:

$$S \to abSb$$

$$S \to aa$$

$$S \to aT_bST_b$$

$$S \to aT_a$$

$$T_a \to a$$

$$T_b \to b$$

Greibach
Normal Form

Theorem: For any context-free grammar (which doesn't produce λ) there is an equivalent grammar in Greibach Normal Form

Observations

- Greibach normal forms are very good for parsing

- It is hard to find the Greibach normal form of any context-free grammar

Try to compute Greibach Normal Form for grammar in CNF example

Compilers

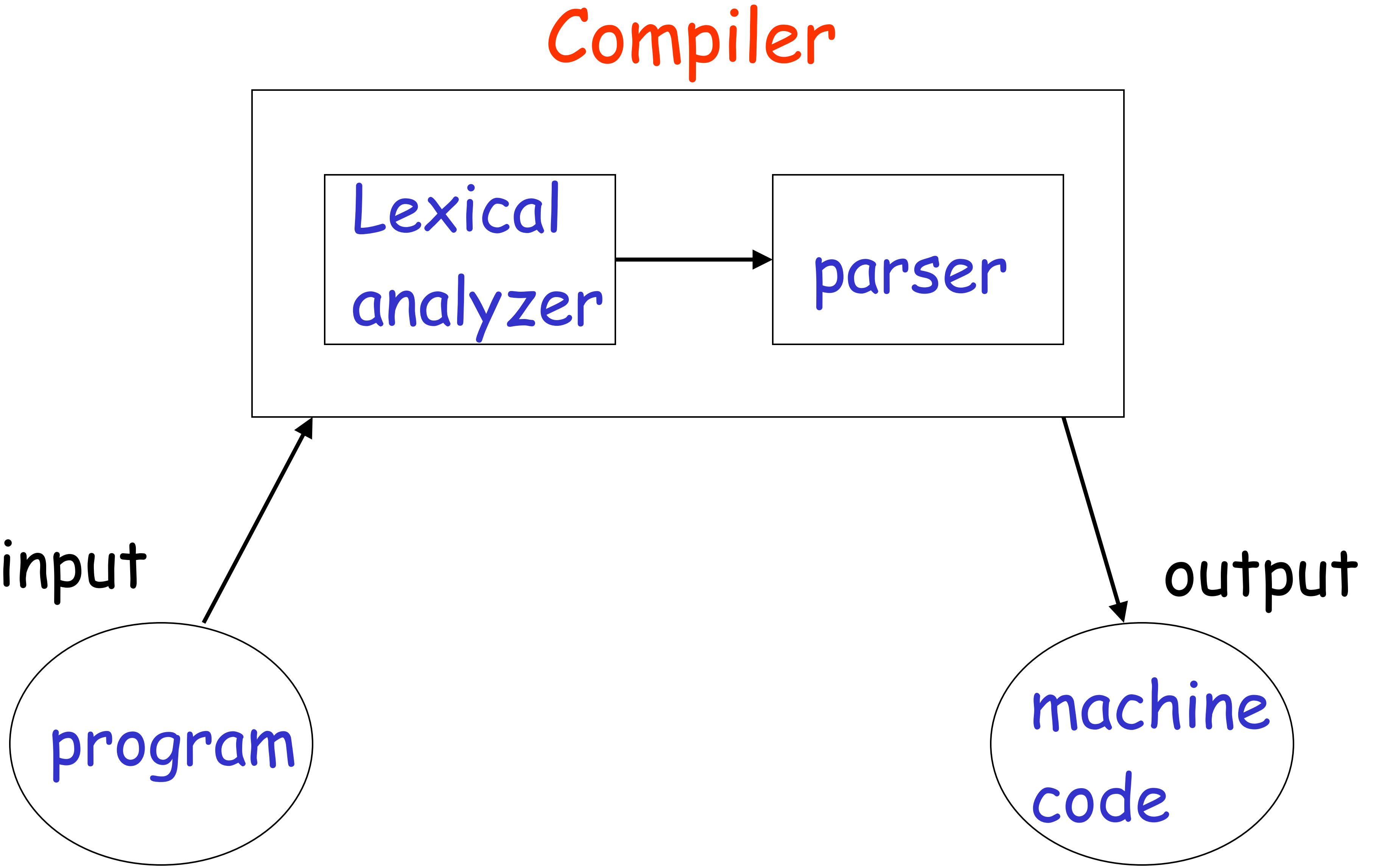
Machine Code

Program

```
v = 5;
if (v>5)
   x = 12 + v;
while (x !=3) {
  x = x - 3;
  v = 10;
}
......
```

Compiler

```
Add v,v,0
cmp v,5
jmplt ELSE
THEN:
  add x, 12,v
ELSE:
WHILE:
cmp x,3
...
```


Compiler
Lexical analyzer
parser
input
program
output
machine code

A parser knows the grammar of the programming language

Parser

PROGRAM → STMT_LIST

STMT_LIST → STMT; STMT_LIST | STMT;

STMT → EXPR | IF_STMT | WHILE_STMT

| { STMT_LIST }

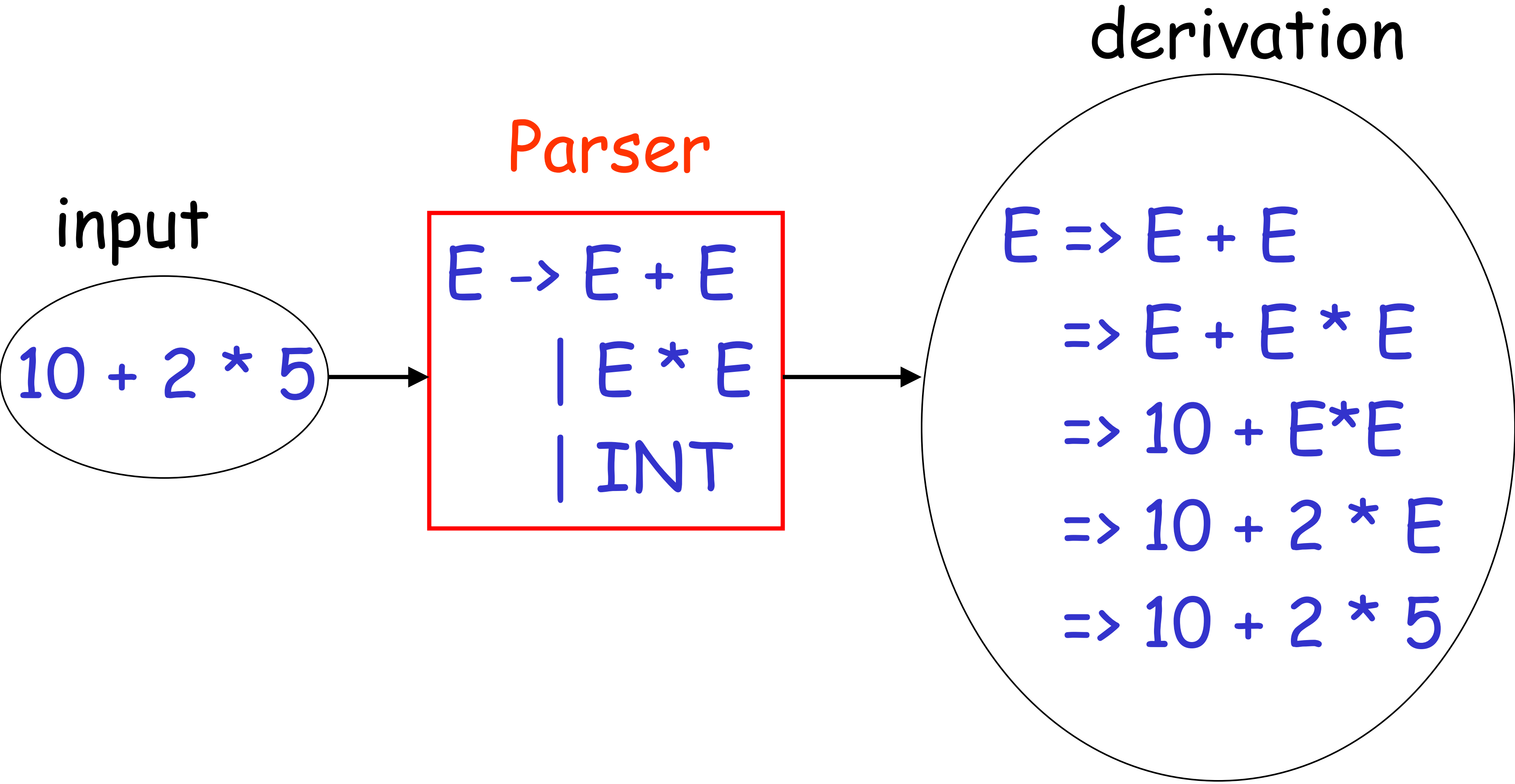
EXPR → EXPR + EXPR | EXPR - EXPR | INT

IF_STMT → if (EXPR) then STMT

| if (EXPR) then STMT else STMT

WHILE_STMT → while (EXPR) do STMT

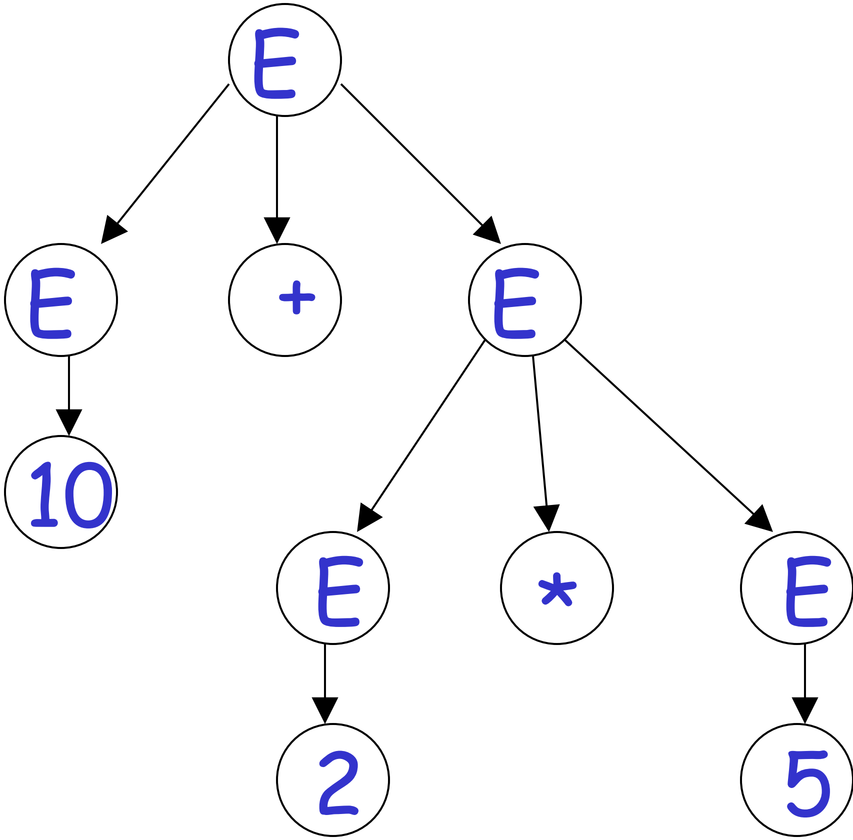
The parser finds the derivation of a particular input



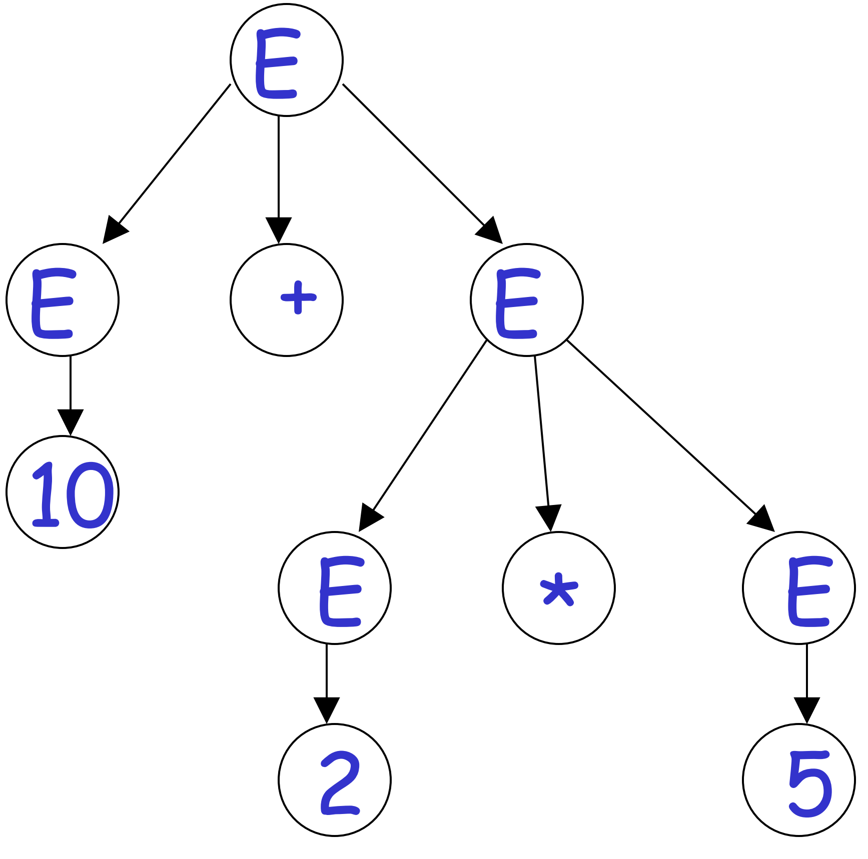
derivation

$$
\begin{aligned}
E &\Rightarrow E + E \\
&\Rightarrow E + E * E \\
&\Rightarrow 10 + E*E \\
&\Rightarrow 10 + 2 * E \\
&\Rightarrow 10 + 2 * 5
\end{aligned}
$$

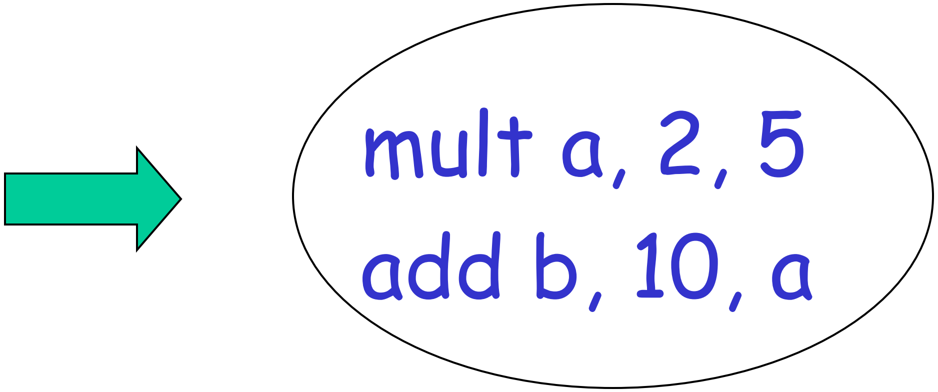
derivation tree



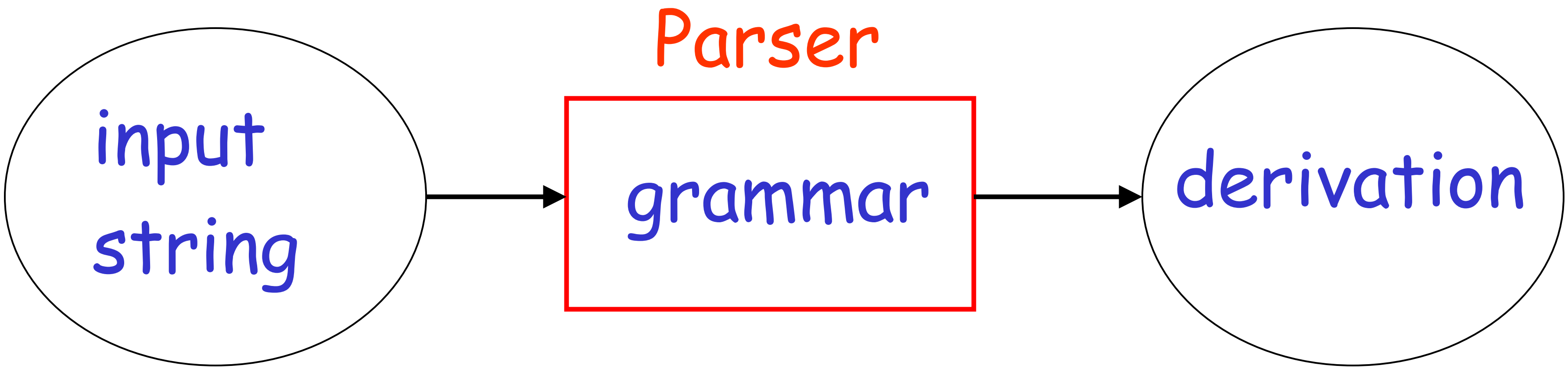
derivation tree



machine code



Parsing


Parser
input
string
grammar
derivation

Example:

input

$aabb$

Parser

$$S \to SS$$
$$S \to aSb$$
$$S \to bSa$$
$$S \to \lambda$$

derivation

?

Parsing algorithm?

Exhaustive Search

$$S \to SS \mid aSb \mid bSa \mid \lambda$$

Phase 1:

$$S \Rightarrow SS$$

$$S \Rightarrow aSb$$

$$S \Rightarrow bSa$$

$$S \Rightarrow \lambda$$

Find derivation of

$$aabb$$

All possible derivations of length 1

$$S \Rightarrow SS$$

$$aabb$$

$$S \Rightarrow aSb$$

$$\cancel{S \Rightarrow bSa}$$

$$\cancel{S \Rightarrow \lambda}$$

Phase 2 $S \to SS \mid aSb \mid bSa \mid \lambda$

$S \Rightarrow SS \Rightarrow SSS$

$S \Rightarrow SS \Rightarrow aSbS$

$aabb$

Phase 1

$\cancel{S \Rightarrow SS \Rightarrow bSaS}$

+ 2 more

$S \Rightarrow SS$

$S \Rightarrow SS \Rightarrow S$

$S \Rightarrow aSb$

$S \Rightarrow aSb \Rightarrow aSSb$

$S \Rightarrow aSb \Rightarrow aaSbb$

$\cancel{S \Rightarrow aSb \Rightarrow abSab}$

$\cancel{S \Rightarrow aSb \Rightarrow ab}$

$$S \to SS \mid aSb \mid bSa \mid \lambda$$

Phase 2

$$S \Rightarrow SS \Rightarrow SSS$$

$$S \Rightarrow SS \Rightarrow aSbS$$

$$aabb$$

$$S \Rightarrow SS \Rightarrow S$$

$$S \Rightarrow aSb \Rightarrow aSSb$$

$$S \Rightarrow aSb \Rightarrow aaSbb$$

Phase 3

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb$$

Final result of exhaustive search (top-down parsing)

input

$$aabb$$

Parser

$$S \to SS$$

$$S \to aSb$$

$$S \to bSa$$

$$S \to \lambda$$

derivation

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb$$

Is exhaustive search a good parsing algorithm?

Time complexity of exhaustive search

Suppose there are no productions of the form

$$A \to \lambda$$

$$A \to B$$

Number of phases for string w : $2|w|$

For grammar with k rules

Time for phase 1: k

k possible derivations

Time for phase 2: k^2

k^2 possible derivations

Time for phase $2|w|$: $k^{2|w|}$

$k^{2|w|}$ possible derivations

Total time needed for string w:

$$k + k^2 + \cdots + k^{2|w|}$$

phase 1 phase 2 phase 2|w|

Extremely bad!!!

There exist faster algorithms for specialized grammars

S-grammar: $A \to ax$

symbol

string of variables

Pair (A, a) appears once

S-grammar example:

$$S \to aS$$

$$S \to bSS$$

$$S \to c$$

Each string has a unique derivation

$$S \Rightarrow aS \Rightarrow abSS \Rightarrow abcS \Rightarrow abcc$$

For S-grammars:

In the exhaustive search parsing
there is only one choice in each phase

Time for a phase: 1

Total time for parsing string w: $|w|$

For general context-free grammars:

There exists a parsing algorithm
that parses a string $|w|$
in time $|w|^3$

(we will show this in the next class)

The CYK Parser

The CYK Membership Algorithm

Input:

- Grammar G in Chomsky Normal Form
- String w

Output:

find if $w \in L(G)$

The Algorithm

Input example:

- Grammar G :
$$S \to AB$$
$$A \to BB$$
$$A \to a$$
$$B \to AB$$
$$B \to b$$

- String w : $aabbb$

aabbb

a	a	b	b	b
aa	ab	bb	bb	
aab	abb	bbb		
aabb	abbb			
aabbb				

$$S \to AB$$

$$A \to BB$$

$$A \to a$$

$$B \to AB$$

$$B \to b$$

a	a	b	b	b
A	A	B	B	B
aa	ab	bb	bb	
aab	abb	bbb		
aabb	abbb			
aabbb				

$$S \to AB$$

$$A \to BB$$

$$A \to a$$

$$B \to AB$$

$$B \to b$$

a	a	b	b	b
A	A	B	B	B
aa	ab	bb	bb	
	S,B	A	A	
aab	abb	bbb		
aabb	abbb			
aabbb				

$$S \to AB$$

$$A \to BB$$

$$A \to a$$

$$B \to AB$$

$$B \to b$$

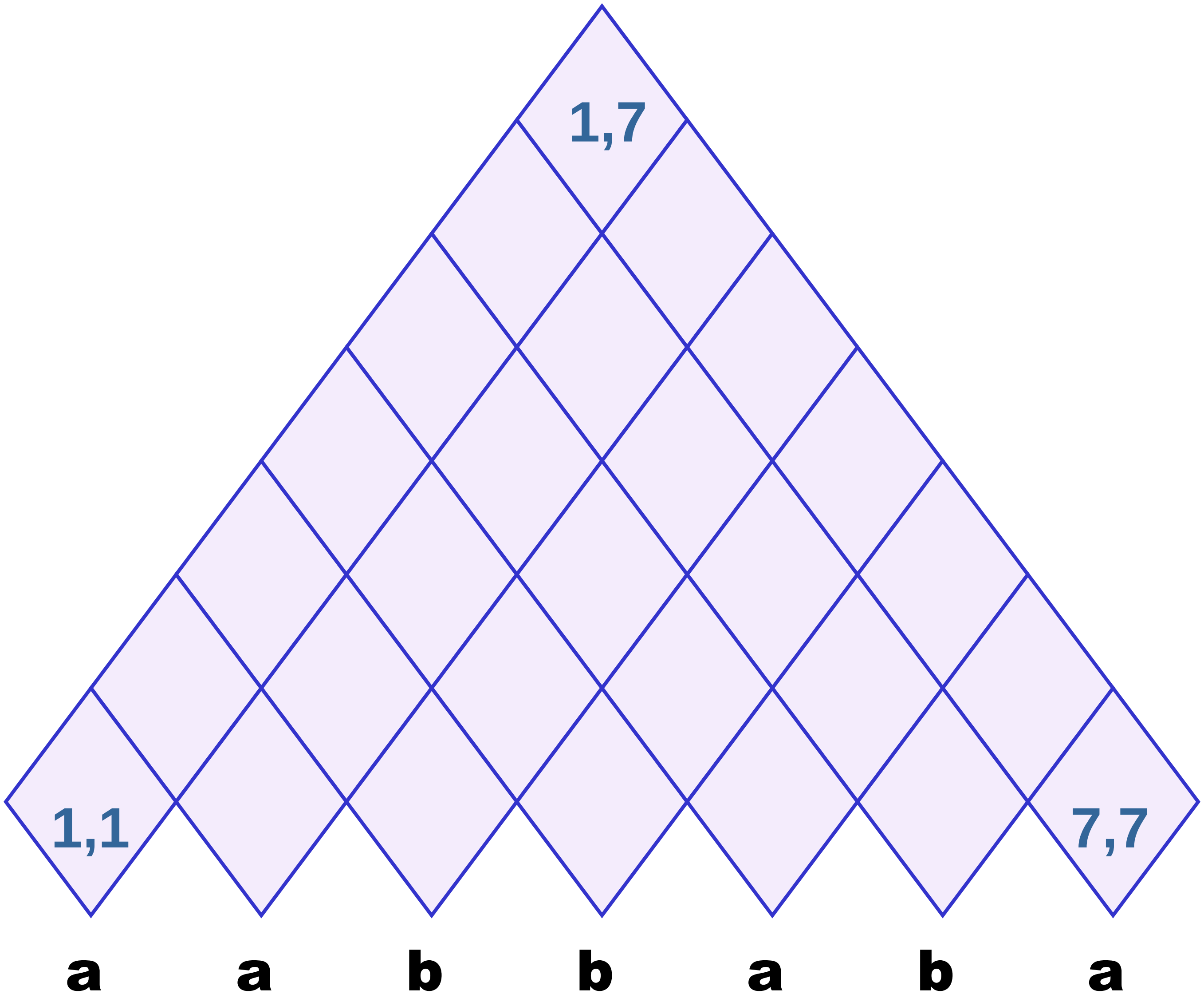
a A	a A	b B	b B	b B
aa	ab S,B	bb A	bb A	
aab S,B	abb A	bbb S,B		
aabb A	abbb S,B			
aabbb S,B				

Therefore: $aabbb \in L(G)$

Time Complexity: $|w|^3$

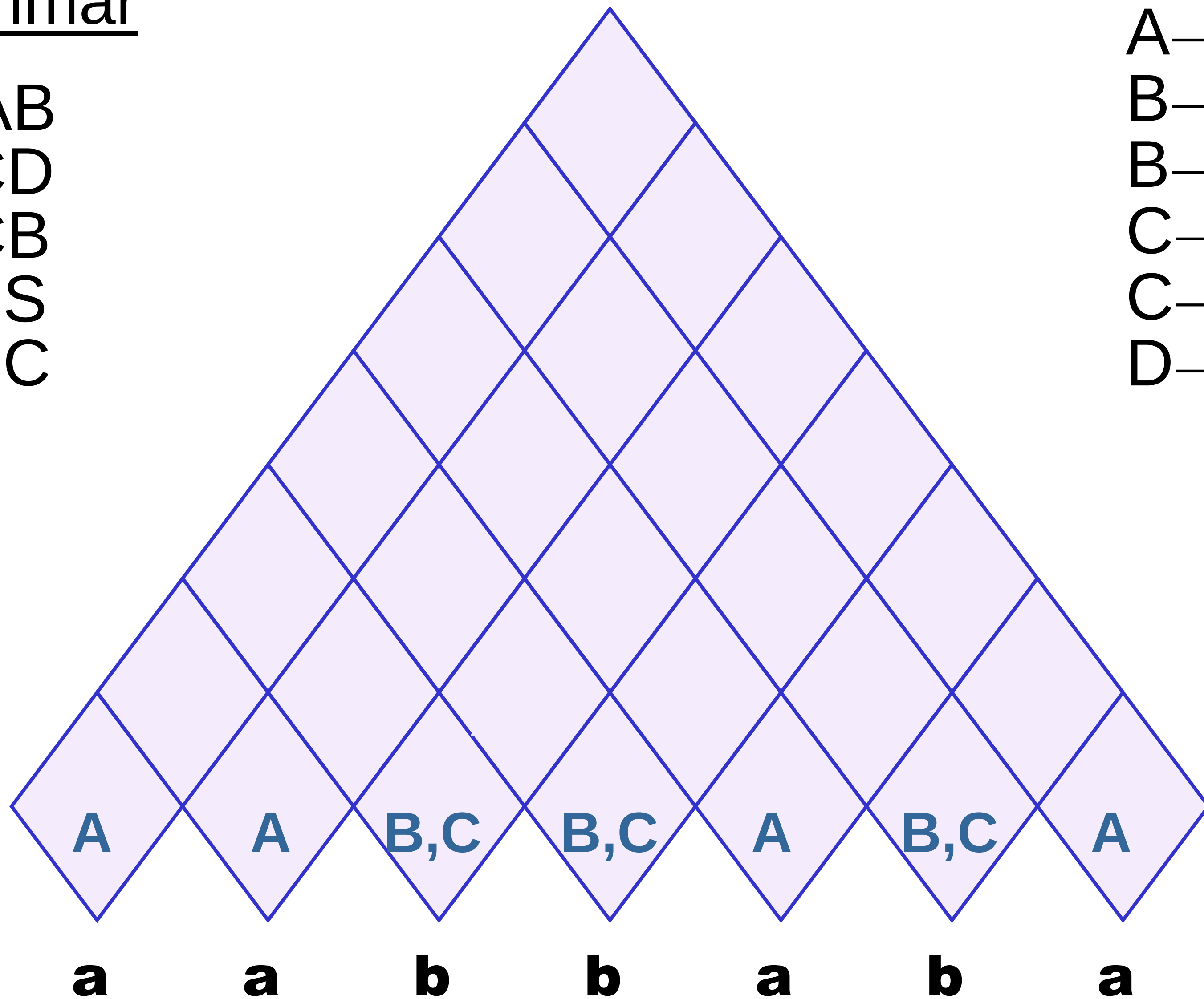
Observation: The CYK algorithm can be easily converted to a parser (bottom up parser)

The following slides are courtesy of Professor Papp, University of Debrecen.


1,7
1,1
7,7
a
a
b
b
a
b
a

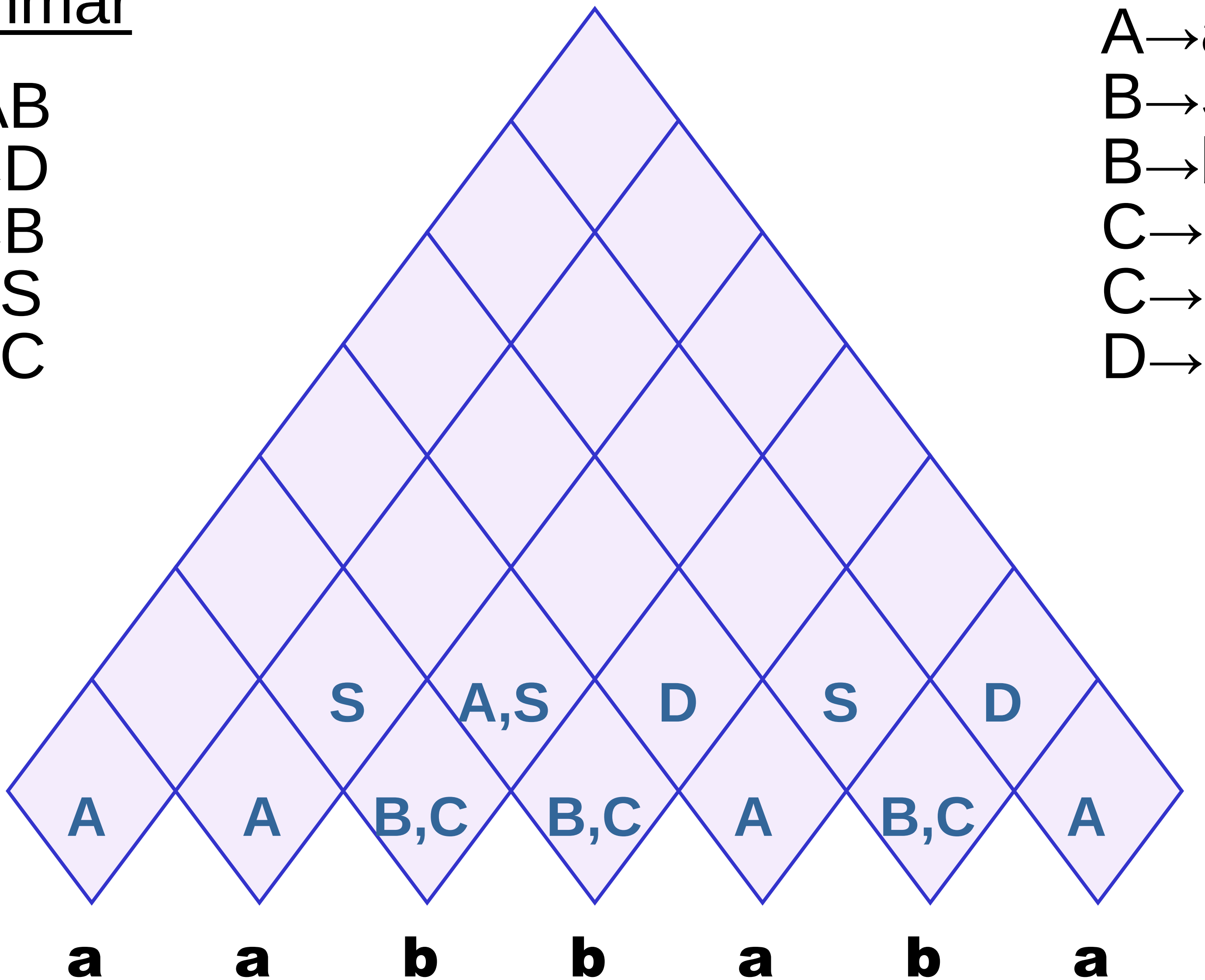
Grammar

$S \to AB$
$S \to CD$
$S \to CB$
$S \to SS$
$A \to BC$
$A \to a$
$B \to SC$
$B \to b$
$C \to DD$
$C \to b$
$D \to BA$



Grammar

S→AB
S→CD
S→CB
S→SS
A→BC

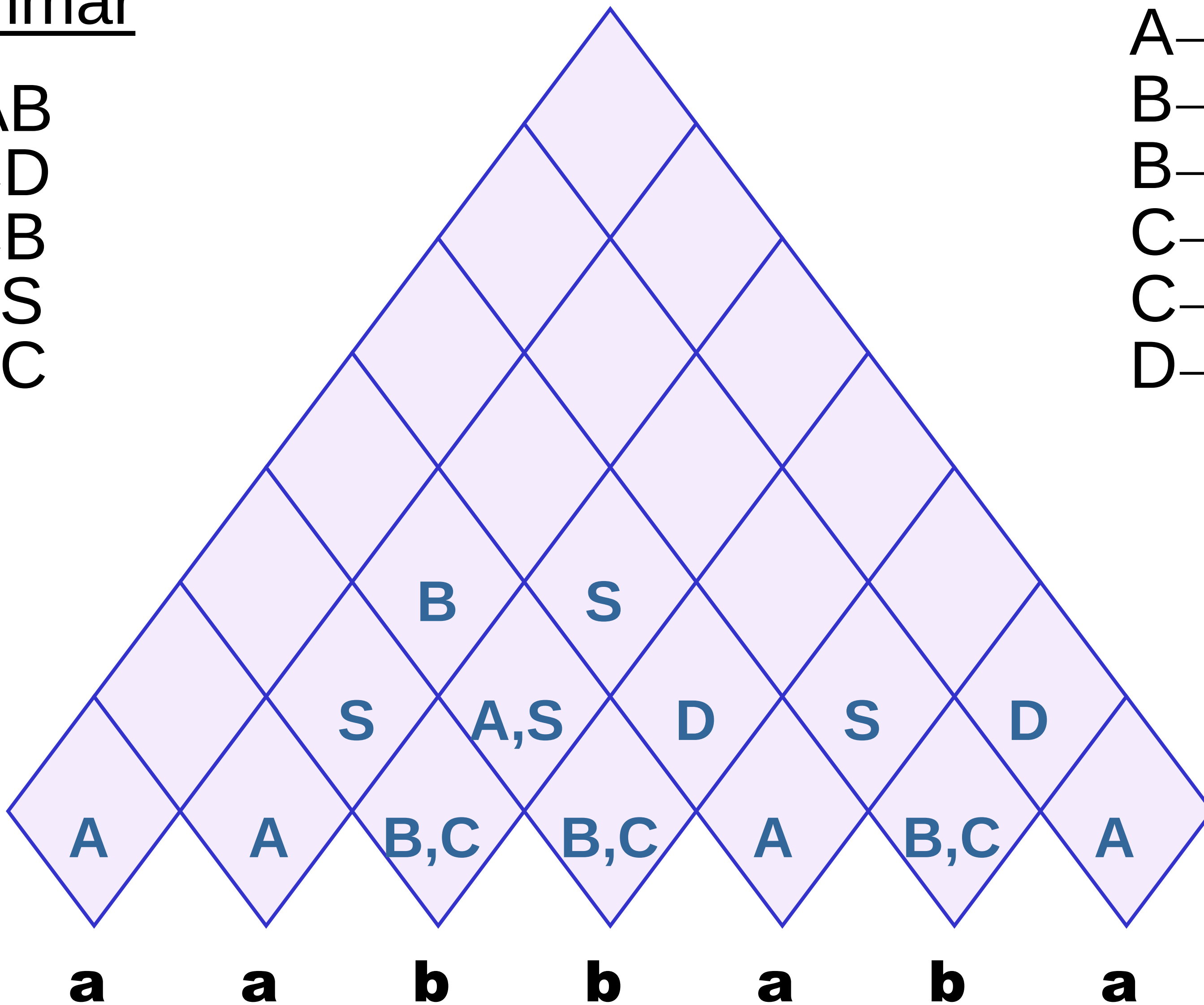


A→a
B→SC
B→b
C→DD
C→b
D→BA

Grammar

S→AB
S→CD
S→CB
S→SS
A→BC

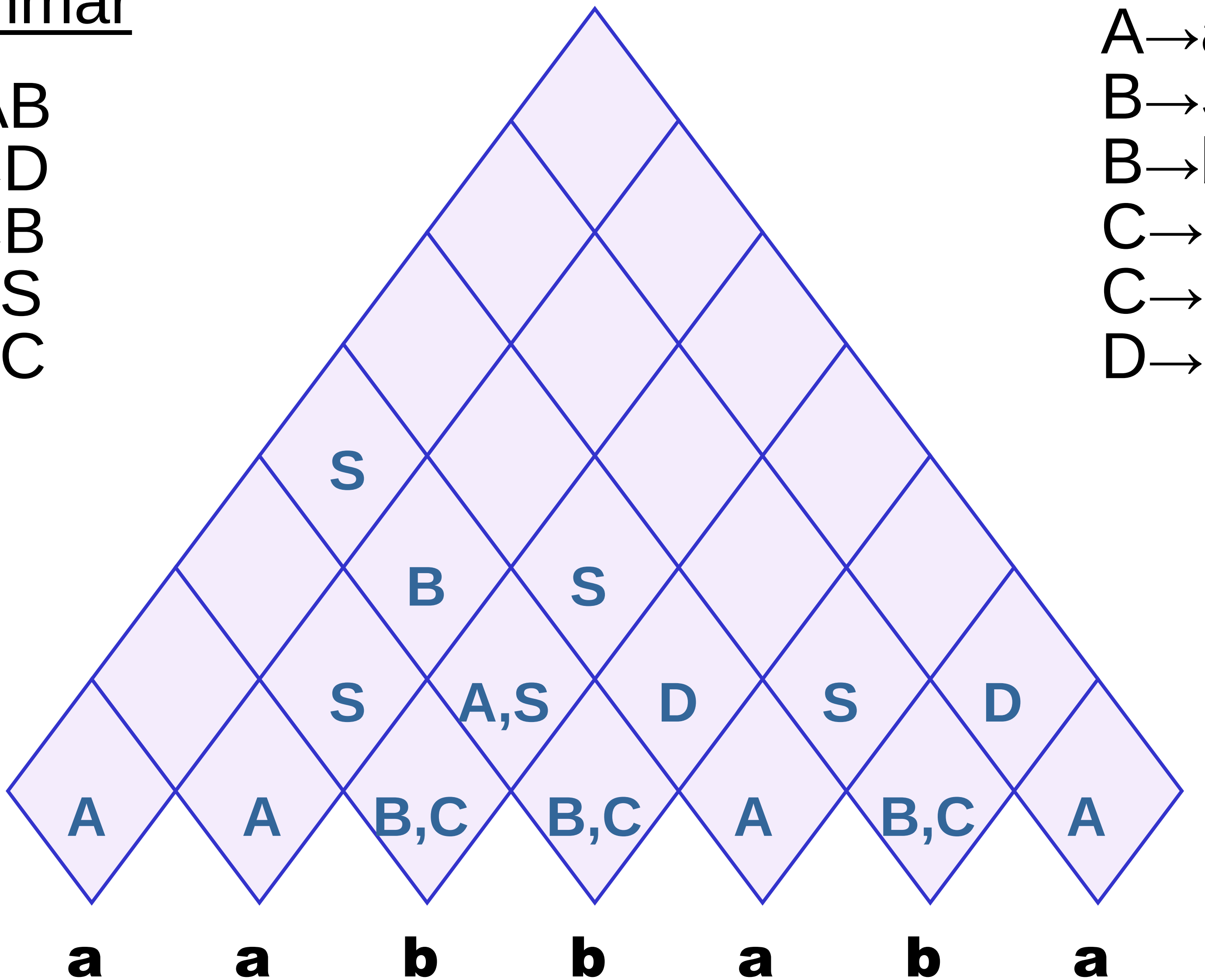
A→a
B→SC
B→b
C→DD
C→b
D→BA



Grammar

S→AB
S→CD
S→CB
S→SS
A→BC

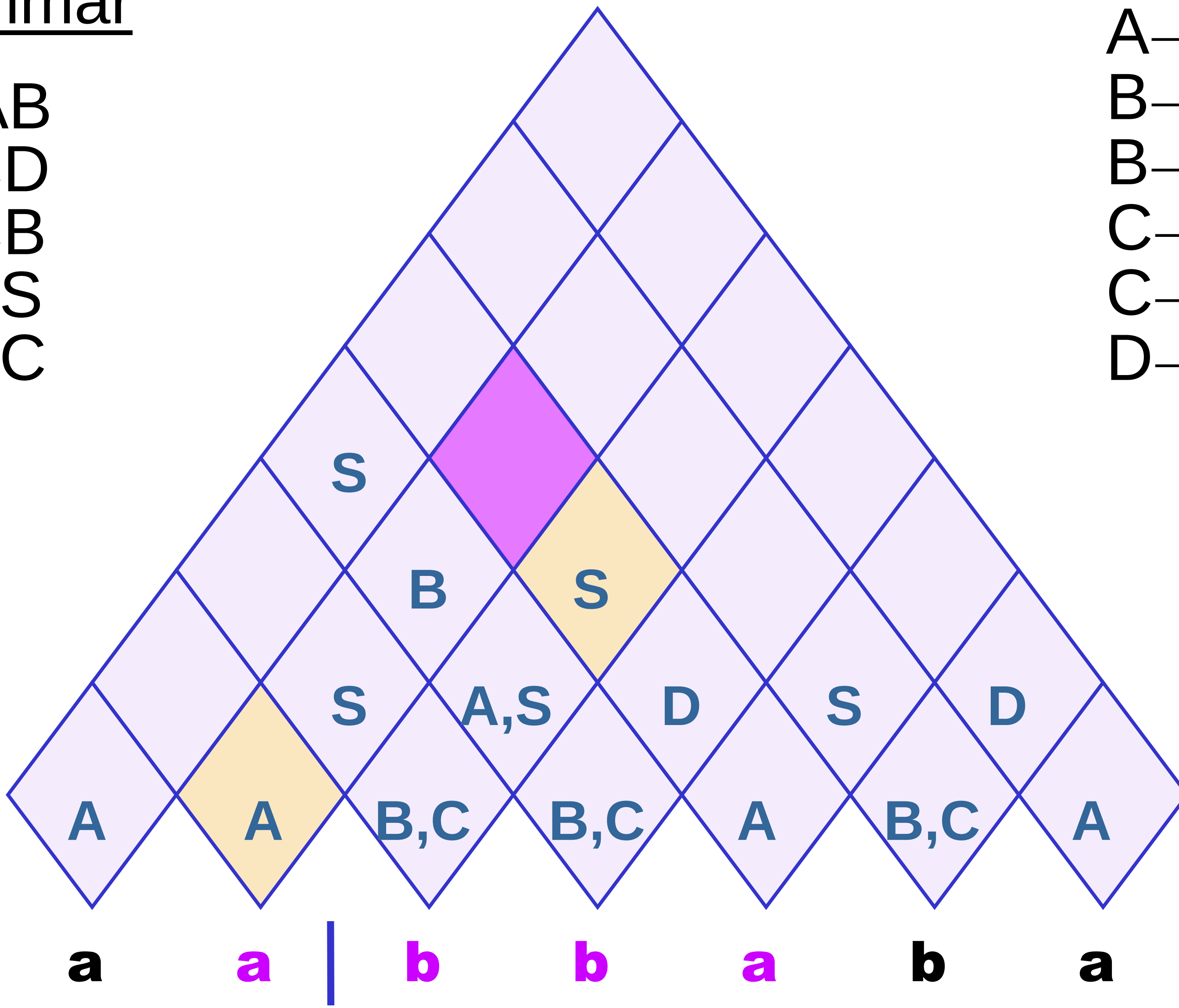
A→a
B→SC
B→b
C→DD
C→b
D→BA



Grammar

S→AB
S→CD
S→CB
S→SS
A→BC

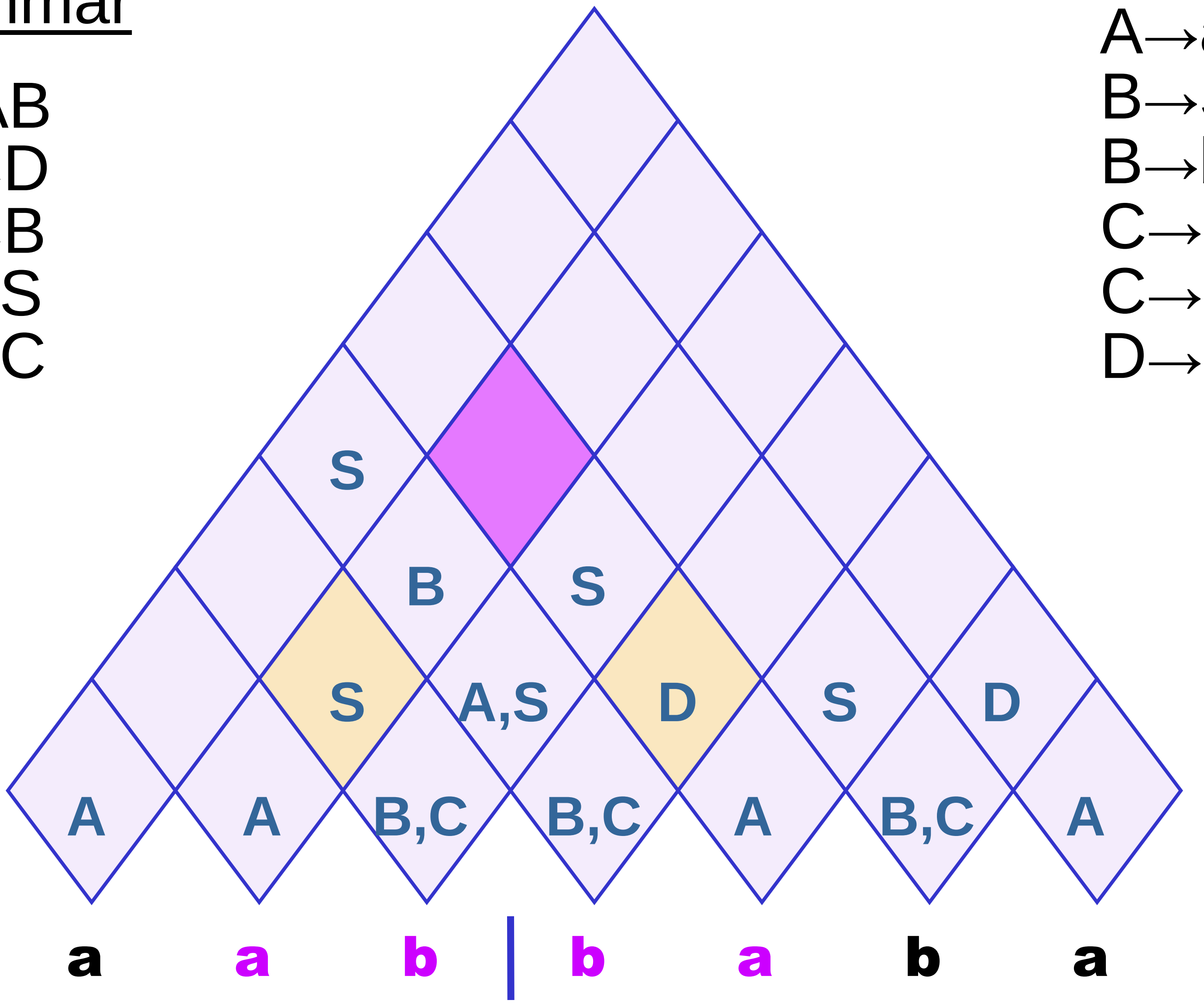
A→a
B→SC
B→b
C→DD
C→b
D→BA



Grammar

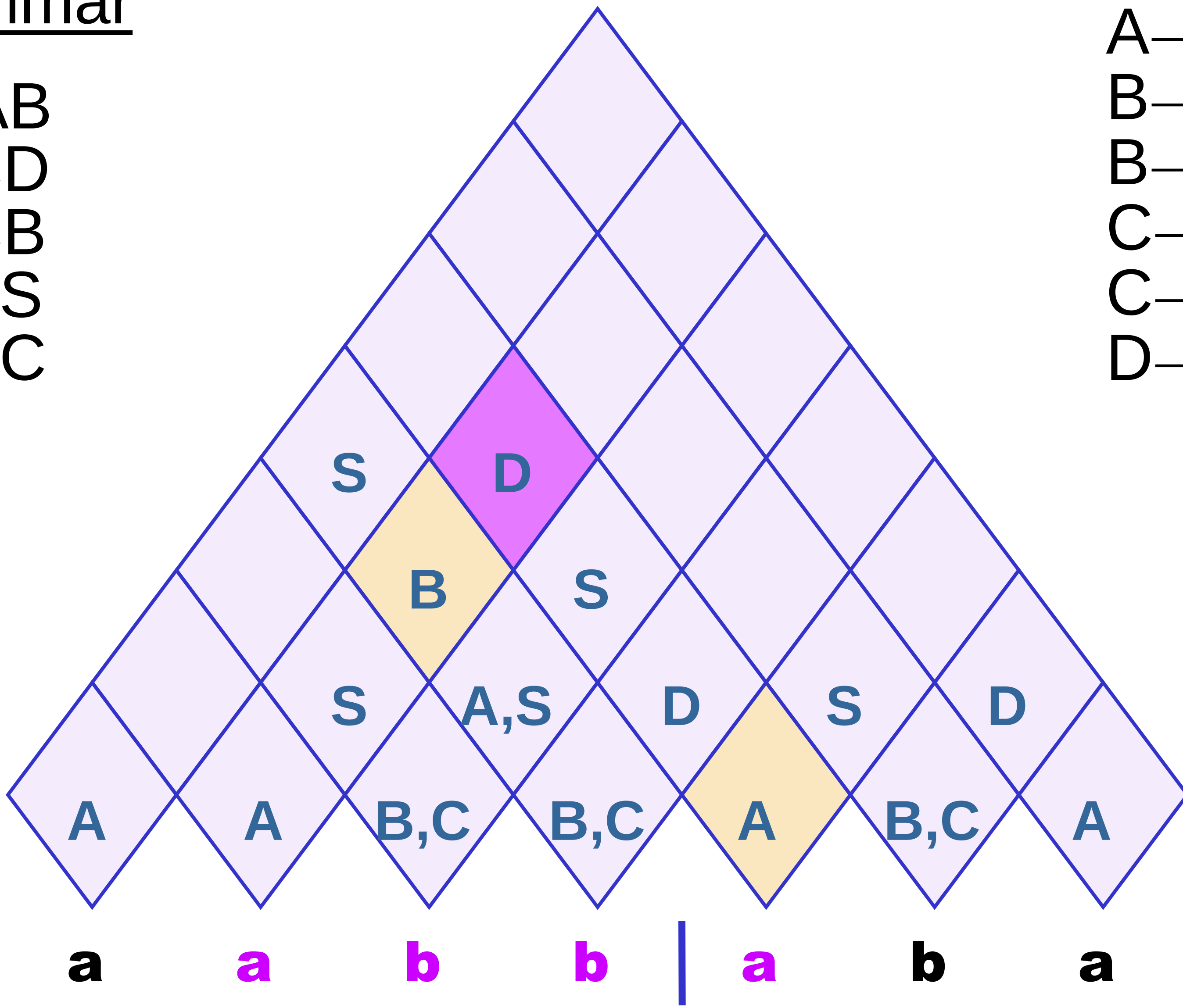
S→AB
S→CD
S→CB
S→SS
A→BC

A→a
B→SC
B→b
C→DD
C→b
D→BA



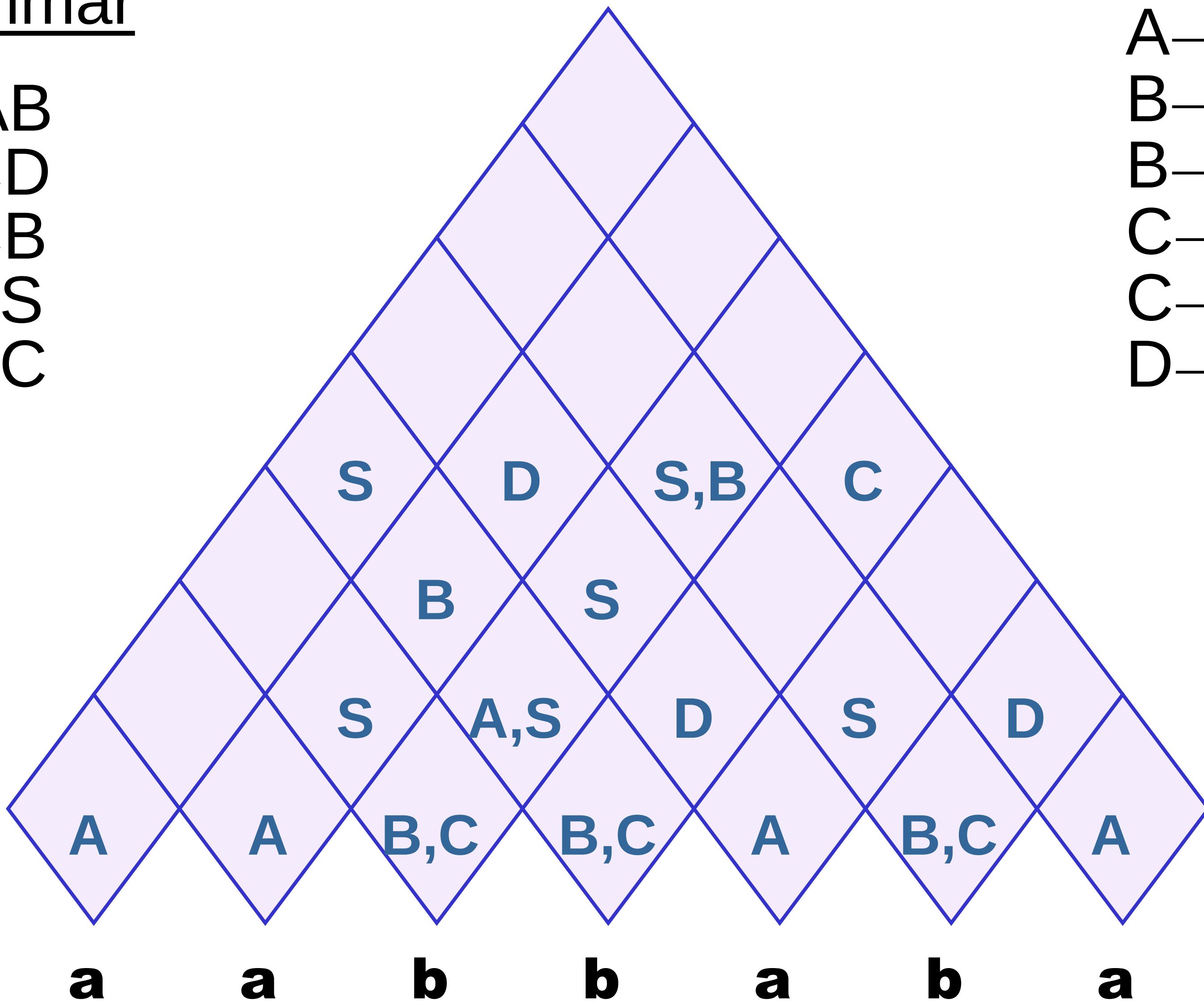
Grammar

$S \rightarrow AB$
$S \rightarrow CD$
$S \rightarrow CB$
$S \rightarrow SS$
$A \rightarrow BC$
$A \rightarrow a$
$B \rightarrow SC$
$B \rightarrow b$
$C \rightarrow DD$
$C \rightarrow b$
$D \rightarrow BA$



Grammar

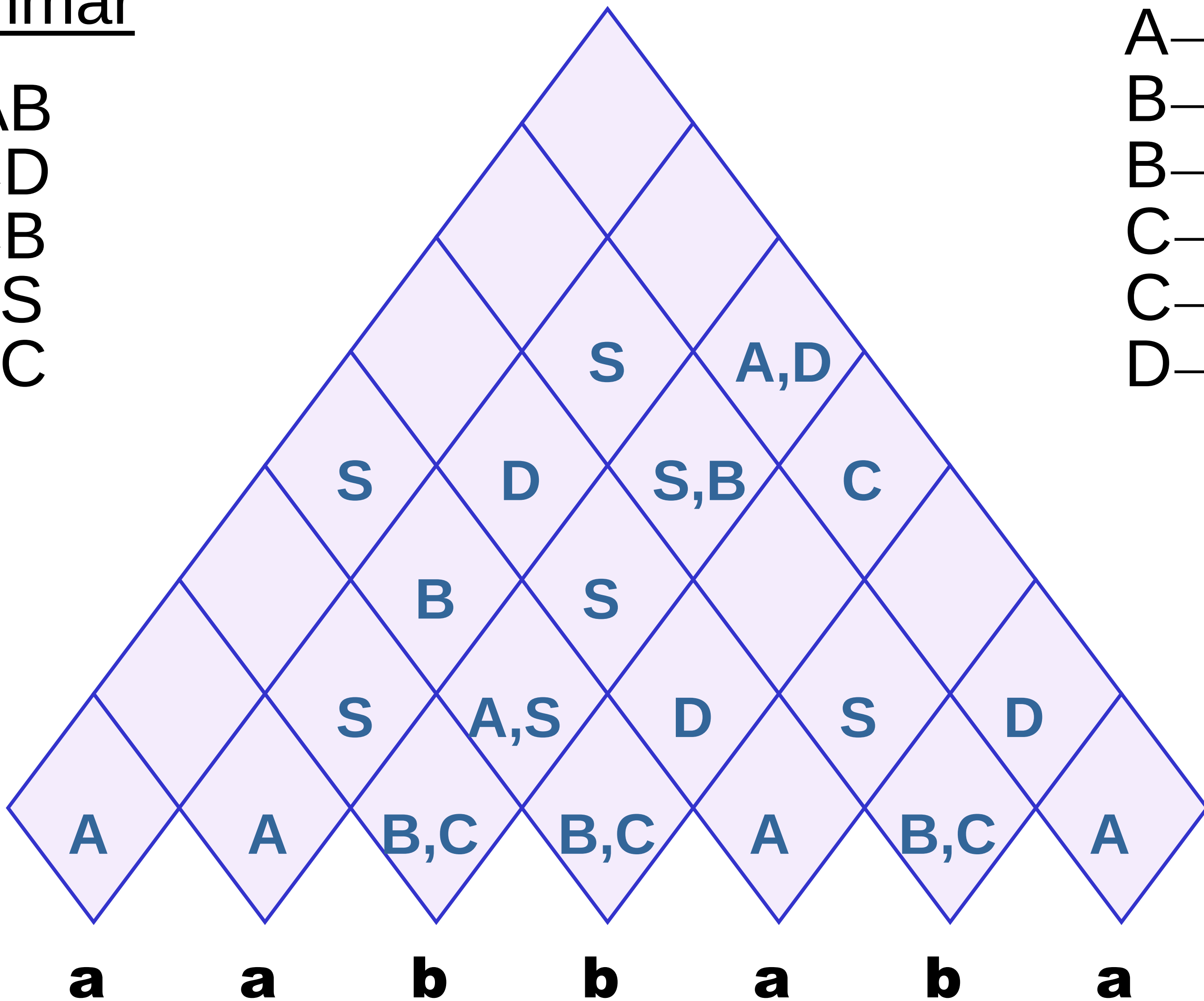
$S \rightarrow AB$
$S \rightarrow CD$
$S \rightarrow CB$
$S \rightarrow SS$
$A \rightarrow BC$
$A \rightarrow a$
$B \rightarrow SC$
$B \rightarrow b$
$C \rightarrow DD$
$C \rightarrow b$
$D \rightarrow BA$



Grammar

S→AB
S→CD
S→CB
S→SS
A→BC

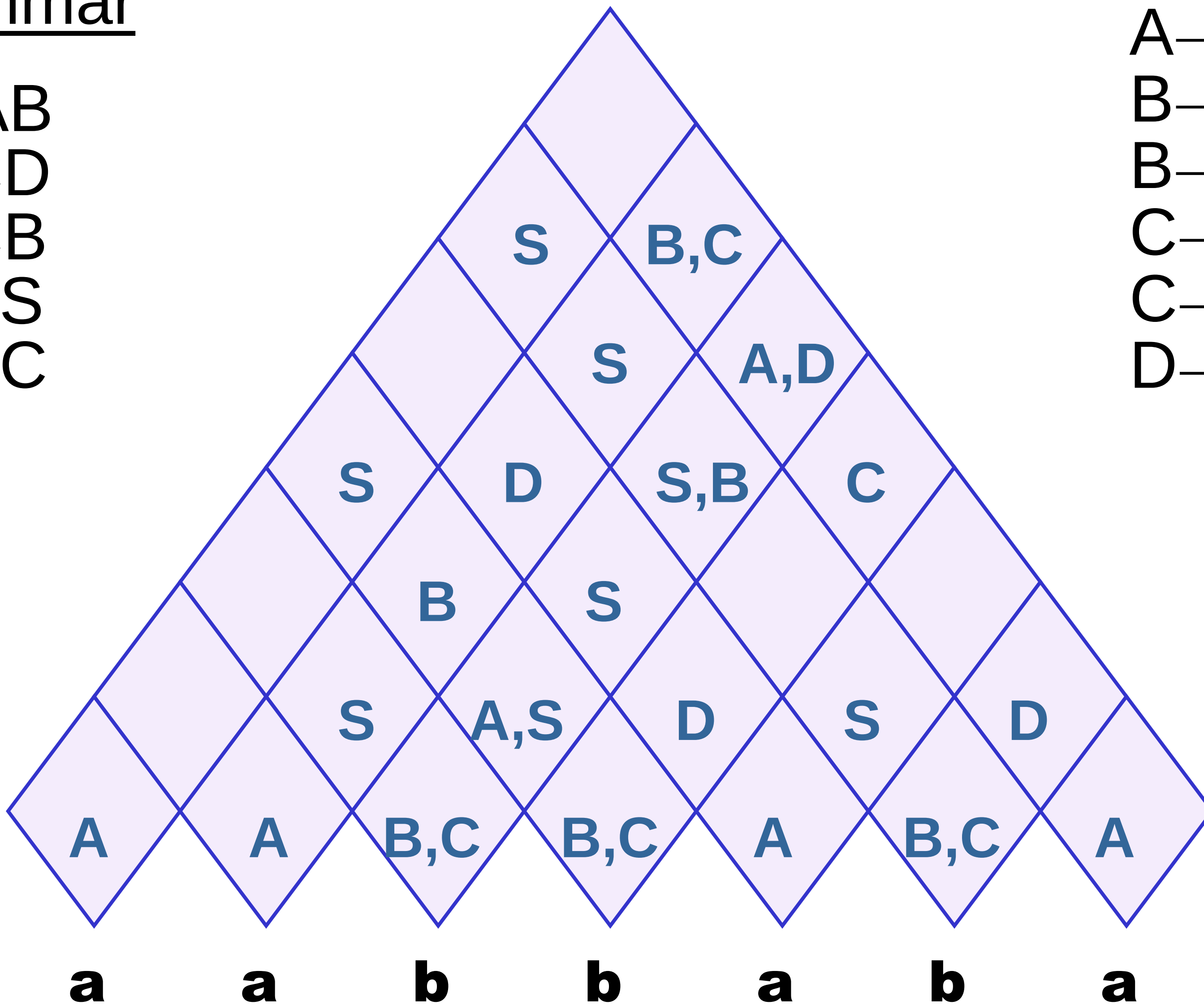
A→a
B→SC
B→b
C→DD
C→b
D→BA



Grammar

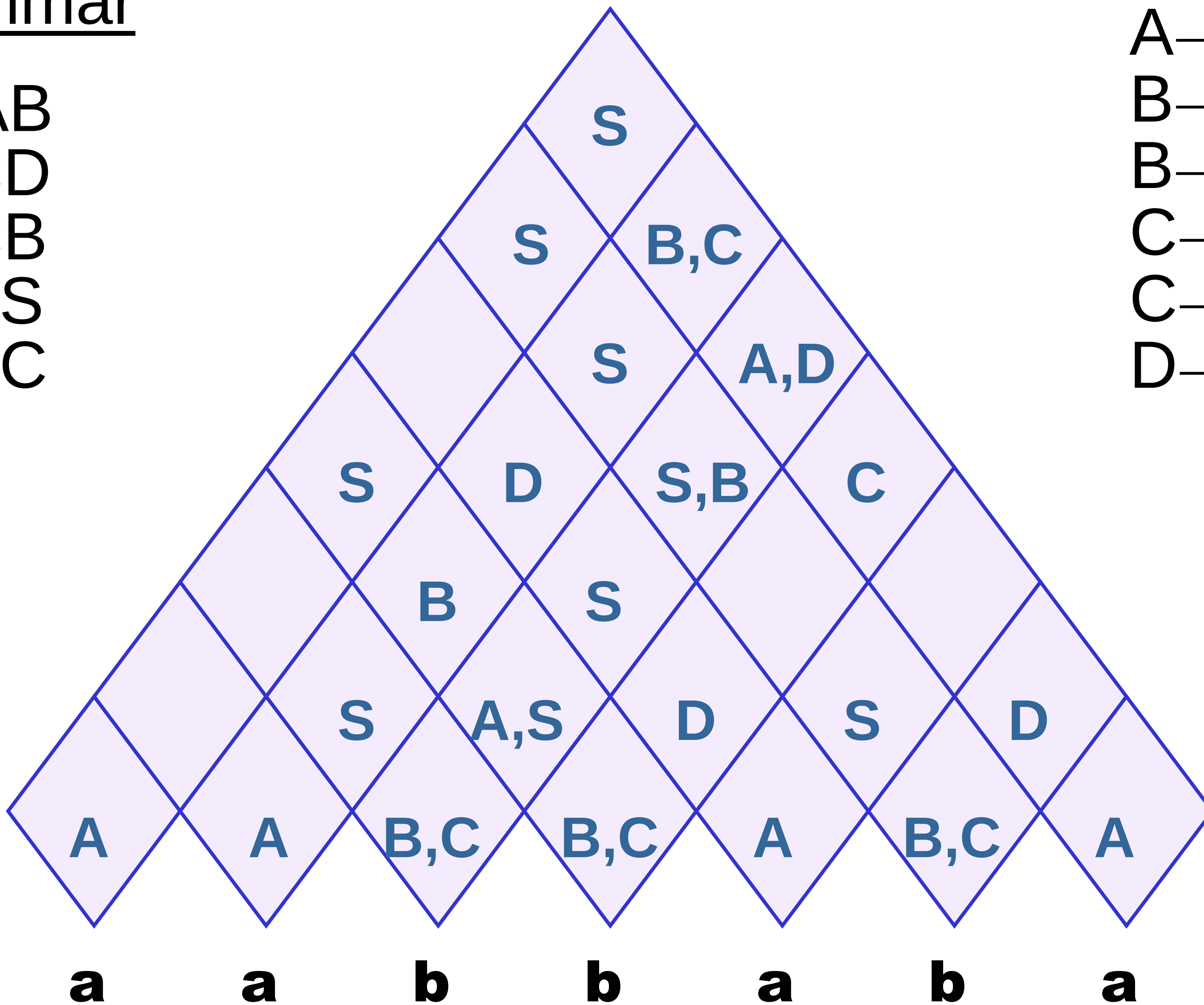
$S \rightarrow AB$
$S \rightarrow CD$
$S \rightarrow CB$
$S \rightarrow SS$
$A \rightarrow BC$

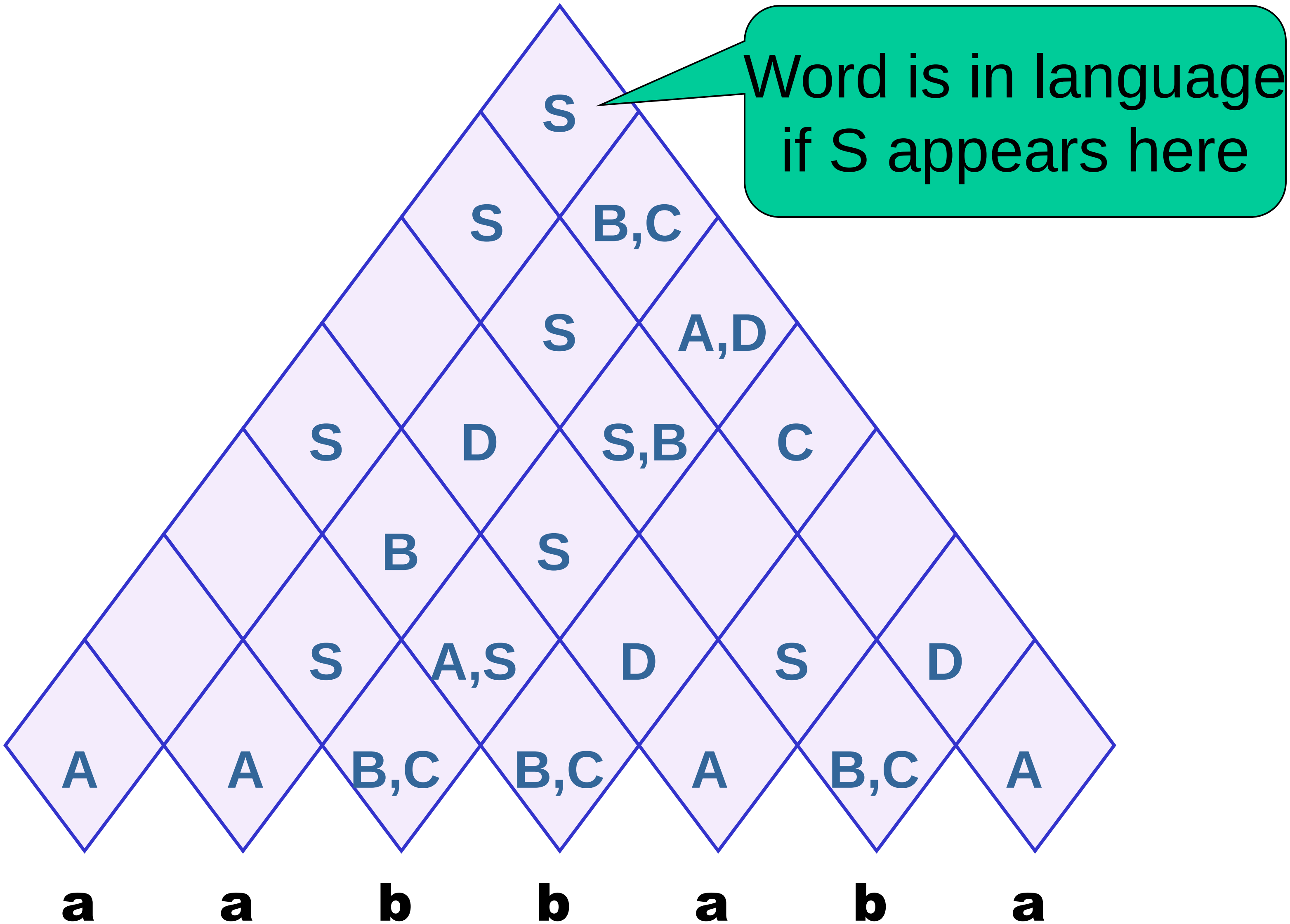
$A \rightarrow a$
$B \rightarrow SC$
$B \rightarrow b$
$C \rightarrow DD$
$C \rightarrow b$
$D \rightarrow BA$



Grammar

$S \rightarrow AB$
$S \rightarrow CD$
$S \rightarrow CB$
$S \rightarrow SS$
$A \rightarrow BC$
$A \rightarrow a$
$B \rightarrow SC$
$B \rightarrow b$
$C \rightarrow DD$
$C \rightarrow b$
$D \rightarrow BA$




Word is in language
if S appears here
S
S
B,C
S
A,D
S
D
S,B
C
B
S
S
A,S
D
S
D
A
A
B,C
B,C
A
B,C
A
a a b b a b a

exercise

Parse “baaba” for this grammar

S -> AB | BC
A -> BA | a
B -> CC | b
C -> AB | a

(Hopcroft, Motwani, Ullman, p301)