Wrocław University of Science and Technology

Iterative method of generating artificial context-free grammars

Olgierd Unold
Agnieszka Kaczmarek
Łukasz Culer

HR EXCELLENCE IN RESEARCH

Agenda

1. Motivation
2. Proposed solution
3. Grammar generator
4. Grammar Complexity Index
5. Positive test set generator
6. Negative test set generator
7. Live presentation

Motivation

1. Grammar-based Classifier System (GCS) - Empirical Grammatical Inference
2. Learning and testing sets - crucial part of every system
   a. Testing performance
   b. Developing improvements

Motivation

Available learning sets

1. Artificial sets (known structure)
   a. L1, L2, A_nB_n…
   b. Limited number, need of manual crafting
2. Real-life sets
   a. Amyloid database
   b. Described by unknown grammar of unknown complexity - which could not even exist
   c. Unrepresentative test set

Learning issues

- low complexity of sets (for simple sets full performance was obtained anyway)
- high complexity of sets (method was unable to learn grammar regardless of new features)
- a lack of specific features of grammar needed to test problematic issues
- smooth complexity incrementation - rarely possible (allow to notice subtle changes in performance)

Wroclaw University of Science and Technology

Proposed solution

Requirements:

- automatic procedure
- creation of consistent, context-free grammars of a given complexity
- positive and negative learning sets

Proposed solution

Grammar consistency

We consider grammar consistent when all the rules and symbols contained by the examined grammar are achievable and productive.

Creating a consistent grammar is the goal of our generation process.

Proposed solution

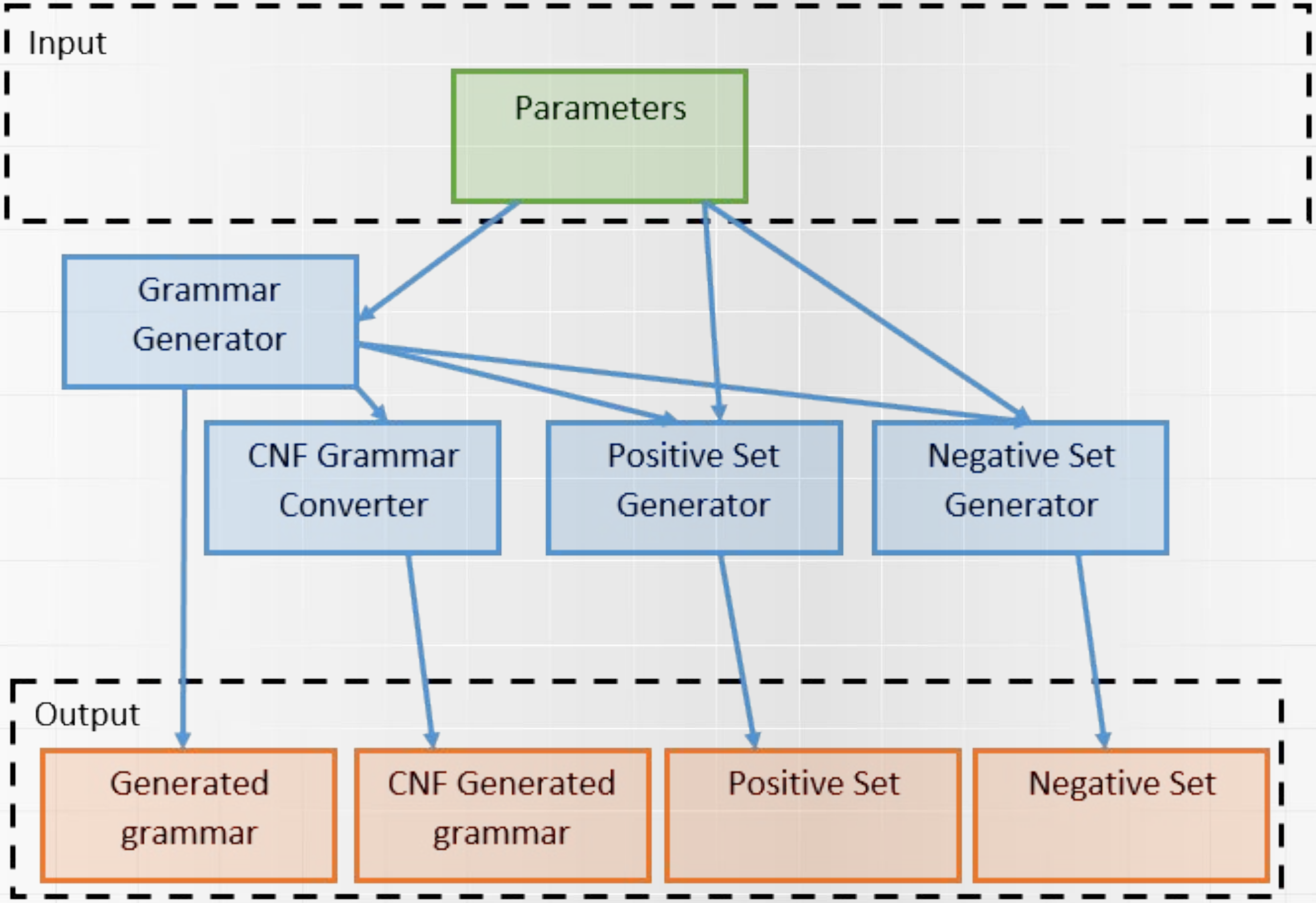
Assumptions

1. Context-free grammars
2. Rule forms:
    a. Parenthesis rules
        - **A→aBb**
        - **A→ab**
    b. Branch rules
        - **A→CD**
    c. Iteration rules
        - **A→cE**
        - **A→Ec**

SAKAKIBARA, Yasubumi; KONDO, Mitsuhiro. GA-based learning of context-free grammars using tabular representations. In: ICML. 1999. p. 354-360

Proposed solution

Algorithm flow



Proposed solution

Parameters

1. Number of parenthesis rules with non-terminal symbol
2. Number of parenthesis rules without non-terminal symbol
3. Number of branch rules
4. Number of iterative rules
5. Maximum number of terminal symbols
6. Maximum number of non-terminal symbols

Grammar generator

1. Rules and symbols are added iteratively during generation until the conditions are met
2. Rules are added according to given principles

Adding principles

1. At first, all terminal rules are added - in that case parenthesis rules with no non-terminal symbols
2. Then - all remaining rules randomly
3. Rules are connected only to productive symbols (only productive symbols have to be on the right side of the rule)
4. Rules can be added using the existing non-terminal symbols on their left side or by creating a new one

Adding principles

5. By creating a new non-terminal symbol on the left side, one of the non-terminal symbols from the right side must be a recently added non-terminal symbol
6. During the creation process number of rules that are left to add should be not lower than the number of remaining unconnected non-terminals
7. After completing the adding process, one of the symbols that would make all other symbols achievable, is converted into a start symbol

Grammar Complexity Index

1. The Grammar Complexity Index (**GCI**) is a simple indicator of grammar complexity.
   It is a sum of all rules that describes a given grammar.
2. A grammar that is described by the rule set P = **{A⟶AB, B⟶a, A⟶b}** has a GCI value of **3**.
3. The introduction of GCI was justified by the need to group generated grammars.

Example

GCI - 5

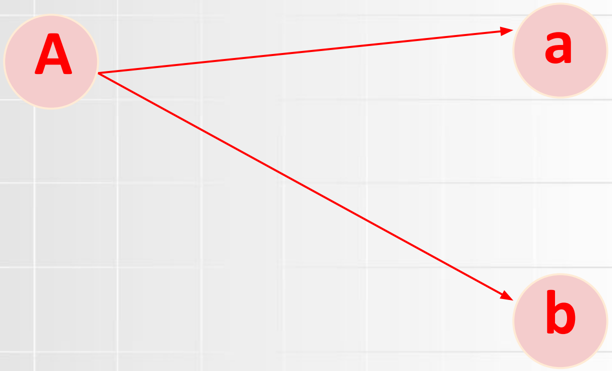
Parenthesis rules without non-terminal symbol - **2**
Parenthesis rules with non-terminal symbol - **0**
Branch rules - **1**
Iterative rules - **2**

Maximal number of terminal symbols - **3**
Maximal number of non-terminal symbols - **4**

Grammar Generator

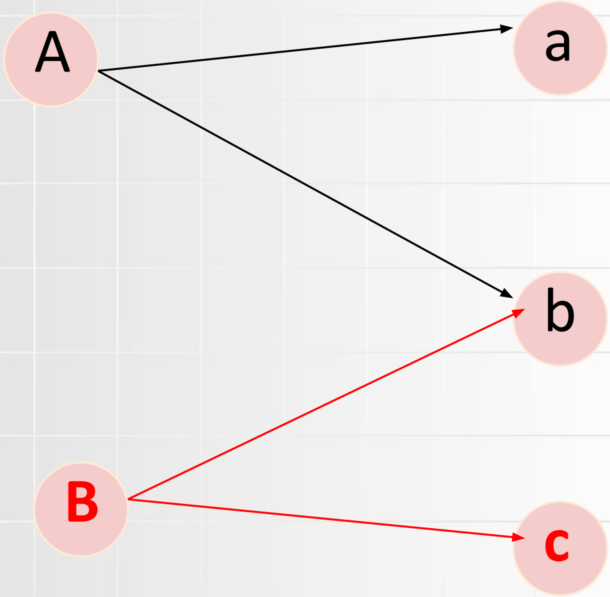
Example

S	R
A	A→ab
a	
b	



Example

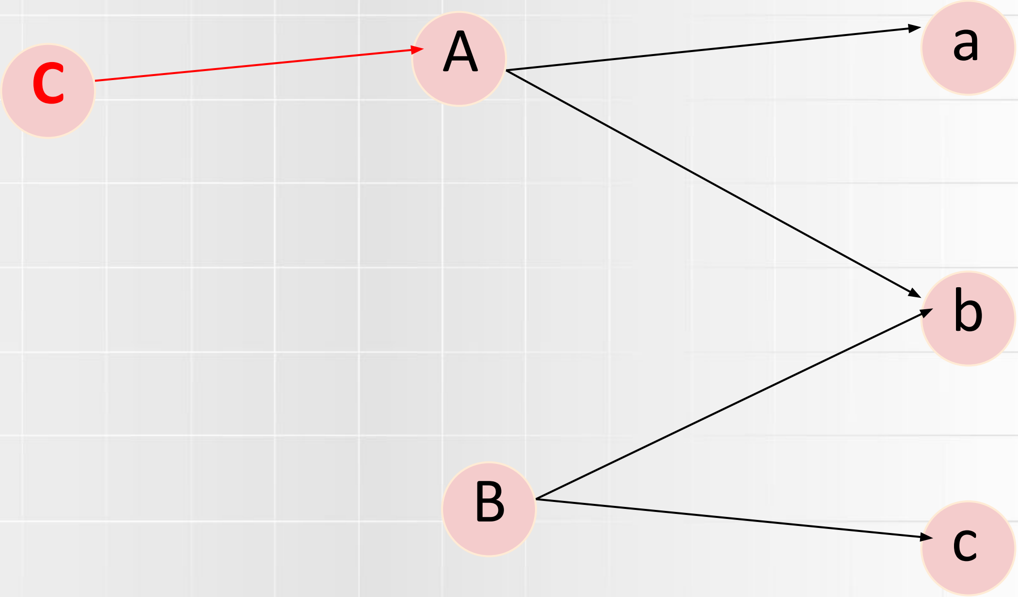
S	R
A	A→ab
B	**B→bc**
a	
b	
c	



Grammar Generator

Example

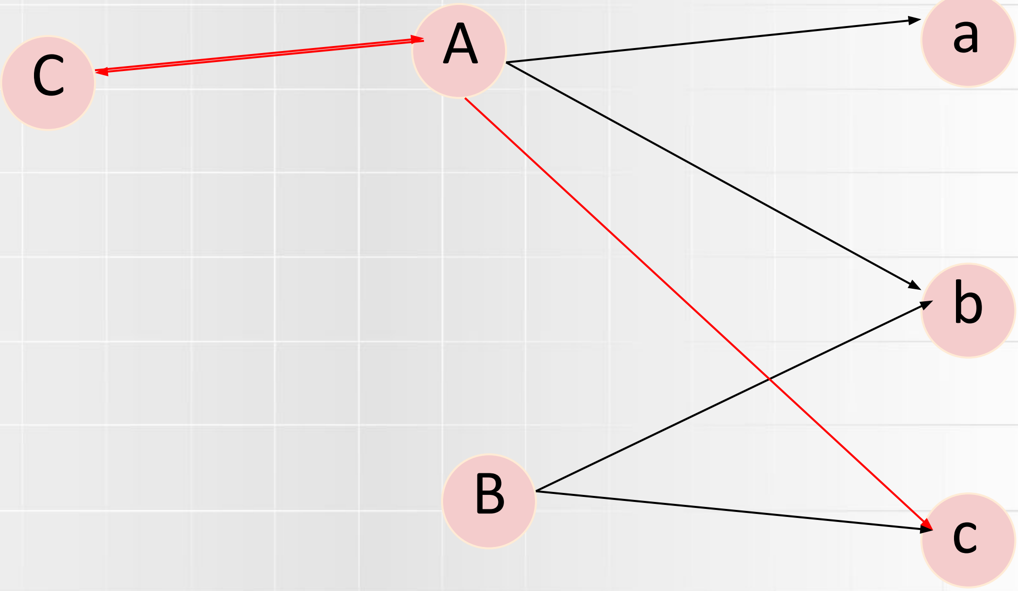
S	R
A	A→ab
B	B→bc
C	**C→AA**
a	
b	
c	



Grammar Generator

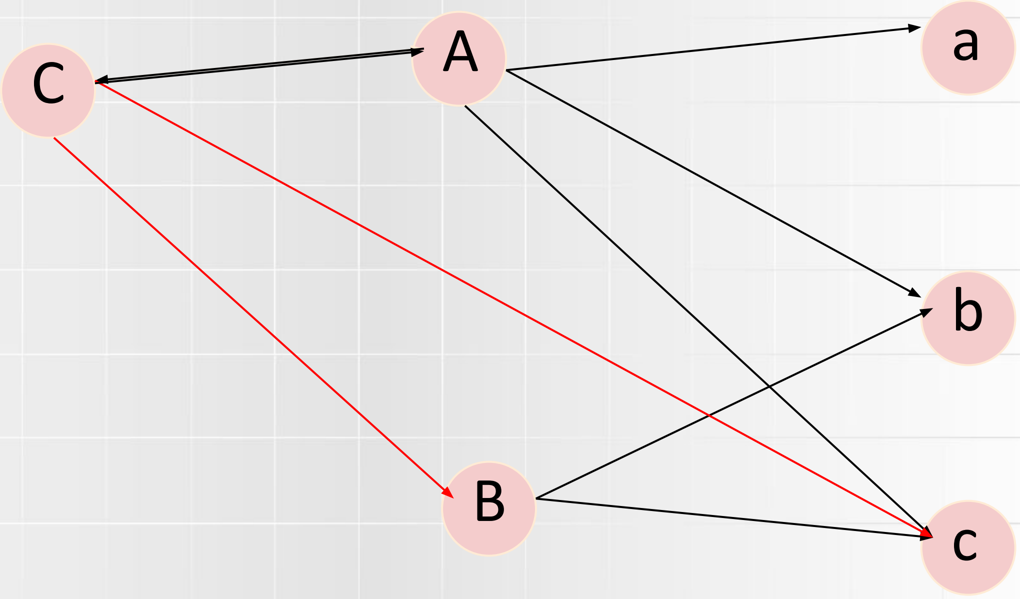
Example

S	R
A	A→ab
B	B→bc
C	C→AA
a	**A→Cc**
b	
c	



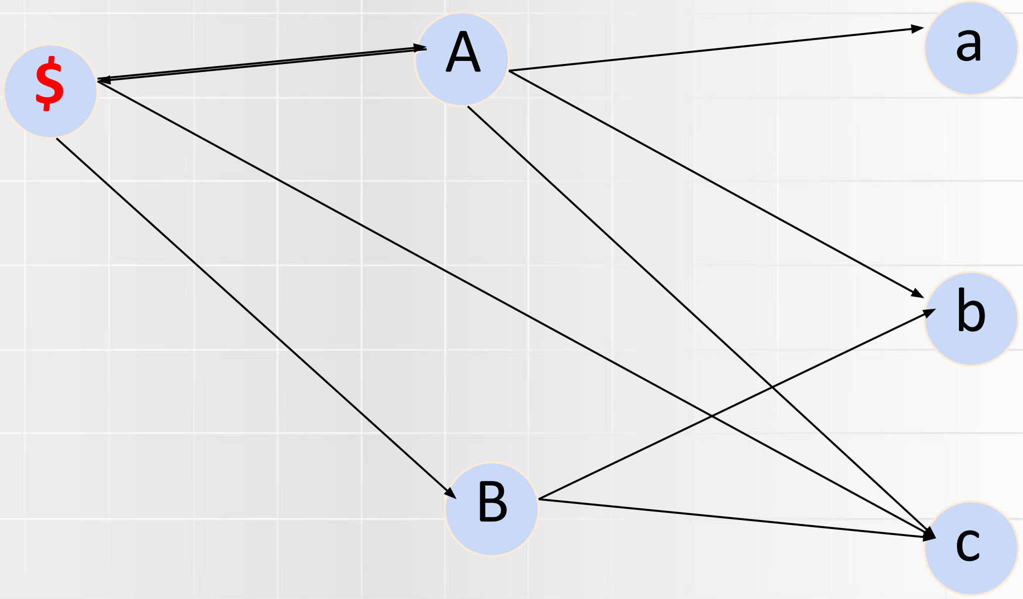
Example

S	R
A	A→ab
B	B→bc
C	C→AA
a	A→Cc
b	**C→Bc**
c	



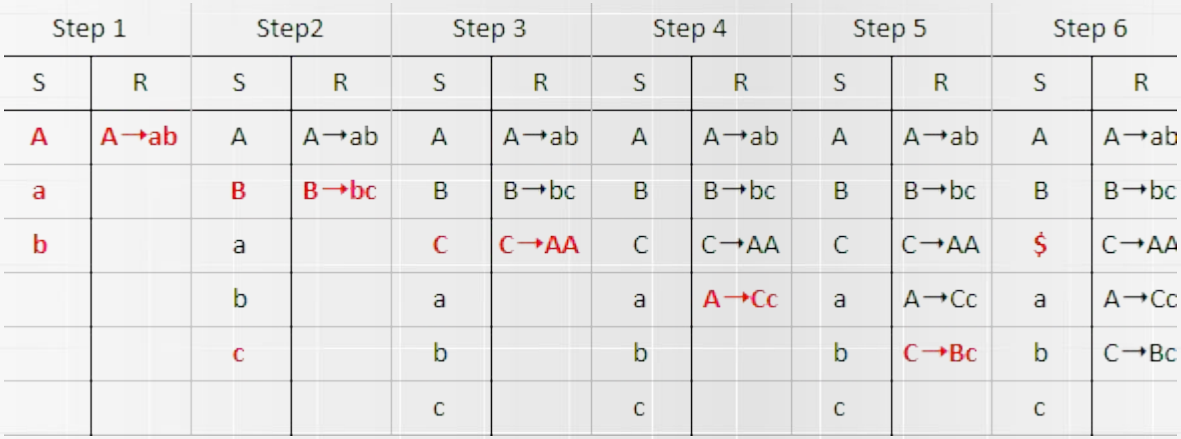
Example

S	R
A	A→ab
B	B→bc
$	C→AA
a	A→Cc
b	C→Bc
c	

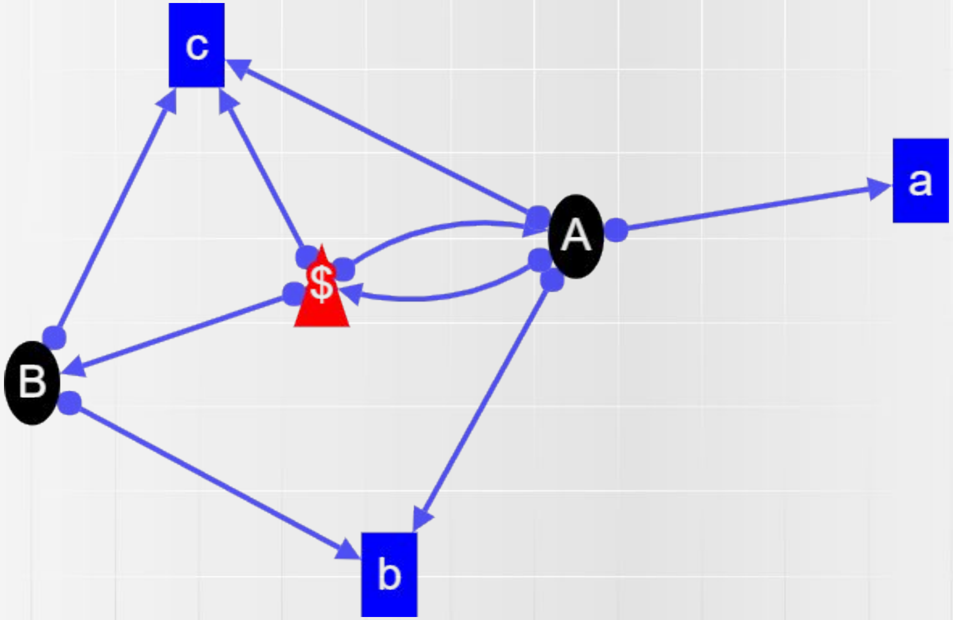


Grammar Generator

Example



Step 1		Step2		Step 3		Step 4		Step 5		Step 6	
S	R	S	R	S	R	S	R	S	R	S	R
A	A→ab	A	A→ab	A	A→ab	A	A→ab	A	A→ab	A	A→ab
a		B	B→bc	B	B→bc	B	B→bc	B	B→bc	B	B→bc
b		a		C	C→AA	C	C→AA	C	C→AA	$	C→AA
		b		a		a	A→Cc	a	A→Cc	a	A→Cc
		c		b		b		b	C→Bc	b	C→Bc
				c		c		c		c	



Theoretical analysis

S_T - maximum number of terminal symbols, $\in \mathbb{N}_+$
S_{NT} - maximum number of non-terminal symbols, $\in \mathbb{N}_+$
R^+_P - parenthesis rules with a non-terminal symbol, $\in \mathbb{N}_0$
R^-_P - parenthesis rules without a non-terminal symbol, $\in \mathbb{N}_+$
R_I - required number of iterative rules, $\in \mathbb{N}_0$
R_B - required number of branch rules, $\in \mathbb{N}_0$

$$\left\{ \begin{array}{c} \dfrac{|R_P^-|}{S_T^2} \leqslant S_{NT} \\ \dfrac{|R_P^-|}{S_T^2} \leqslant |R_P^+| + |R_I| + 2|R_B| + 1 \\ \sqrt{\dfrac{|R_P^+|}{S_T^2}} \leqslant S_{NT} \\ \sqrt{\dfrac{|R_I|}{2S_T}} \leqslant S_{NT} \\ \sqrt[3]{|R_B|} \leqslant S_{NT} \end{array} \right. \qquad where \begin{cases} S_{NT}, S_T, |R_P^-| \in \mathbb{N}_+ \\ |R_P^+|, |R_I|, |R_B| \in \mathbb{N}_0 \end{cases}$$

Theoretical analysis

$$S_T \in \left\langle \sqrt{\frac{|R_P^-|}{|R_P^+| + |R_I| + 2|R_B| + 1}}, 2|R_P^-| + 2|R_P^+| \right\rangle$$

$$\frac{\frac{|R_P^-|}{S_T^2} + \sqrt{\frac{|R_P^+|}{S_T^2}} + \sqrt{\frac{|R_I|}{2S_T}} + \sqrt[3]{|R_B|}}{4} \leqslant S_{NT}$$

$$\begin{cases} S_{NT} \leqslant |R_P^-| + |R_P^+| + |R_I| + |R_B| \quad for \quad |R_B| \geq |R_P^-| - 1 \\ S_{NT} \leqslant |R_P^+| + |R_I| + |2R_B| + 1 \quad for \quad |R_B| < |R_P^-| - 1 \end{cases}$$

Positive test set

1. Generation of all examples - potentially impossible (infinite number?)
2. Expression of the grammar structure with the lowest number of examples
3. Optimal set generation described in [2] (Maximum number of examples - $2|R|^3$)

[2] MAYER, Mikaël; HAMZA, Jad. Optimal Test Sets for Context-Free Languages. arXiv preprint arXiv:1611.06703, 2016.

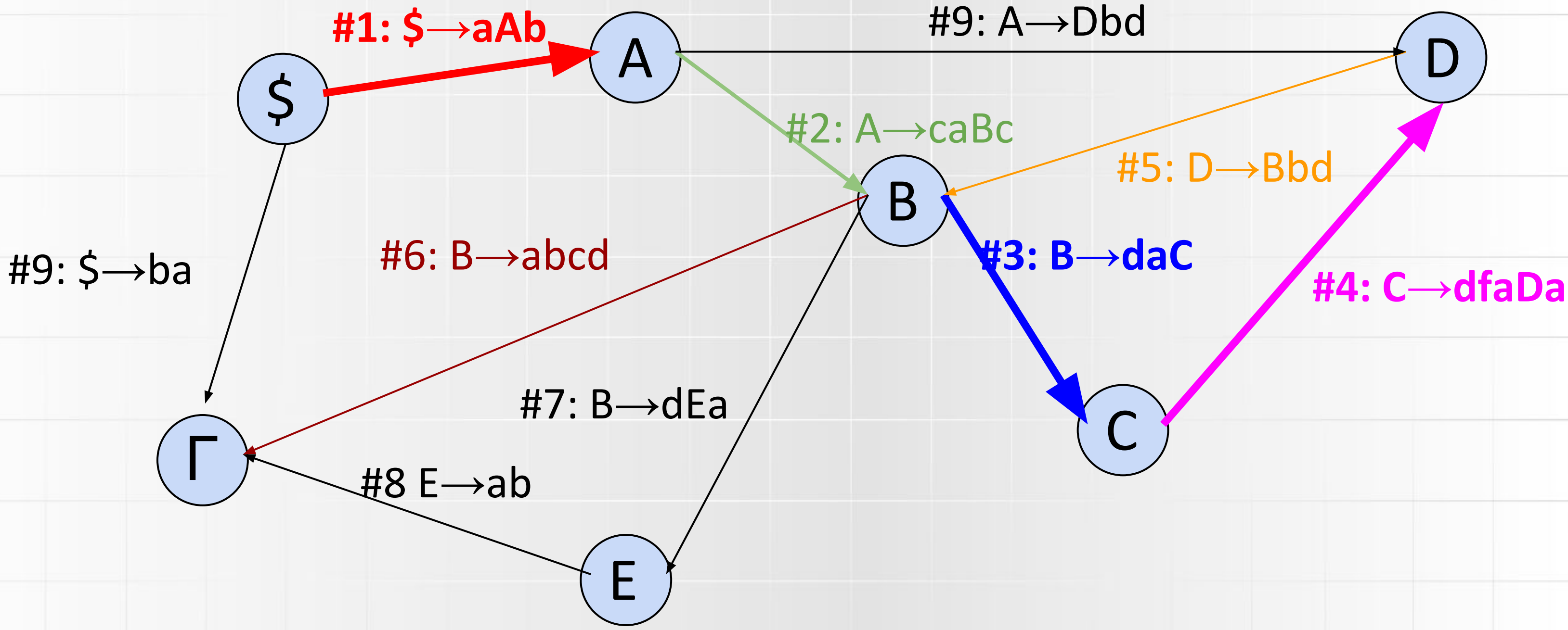
Positive test set

1. Input - any context-free grammar
2. Conversion to linear grammar
3. Graph generation
4. Test set generation based on the graph
5. Output - test set

Positive test set

#1#3#4

$ ⟶ #1 ⟶ #3 ⟶ #4 ⟶ Γ



acaddfaabcdbdacb

Wroclaw University of Science and Technology

Negative test set

1. Conversion to Chomsky Normal Form
2. Generation of a random terminal symbol string with a length from the given interval
3. CYK verification
4. Parsed - dismiss string
5. Not parsed - add to negative test set
6. Repeat until the desired number of strings obtained

Live presentation

http://lukasz.culer.staff.iiar.pwr.edu.pl/gencreator.php