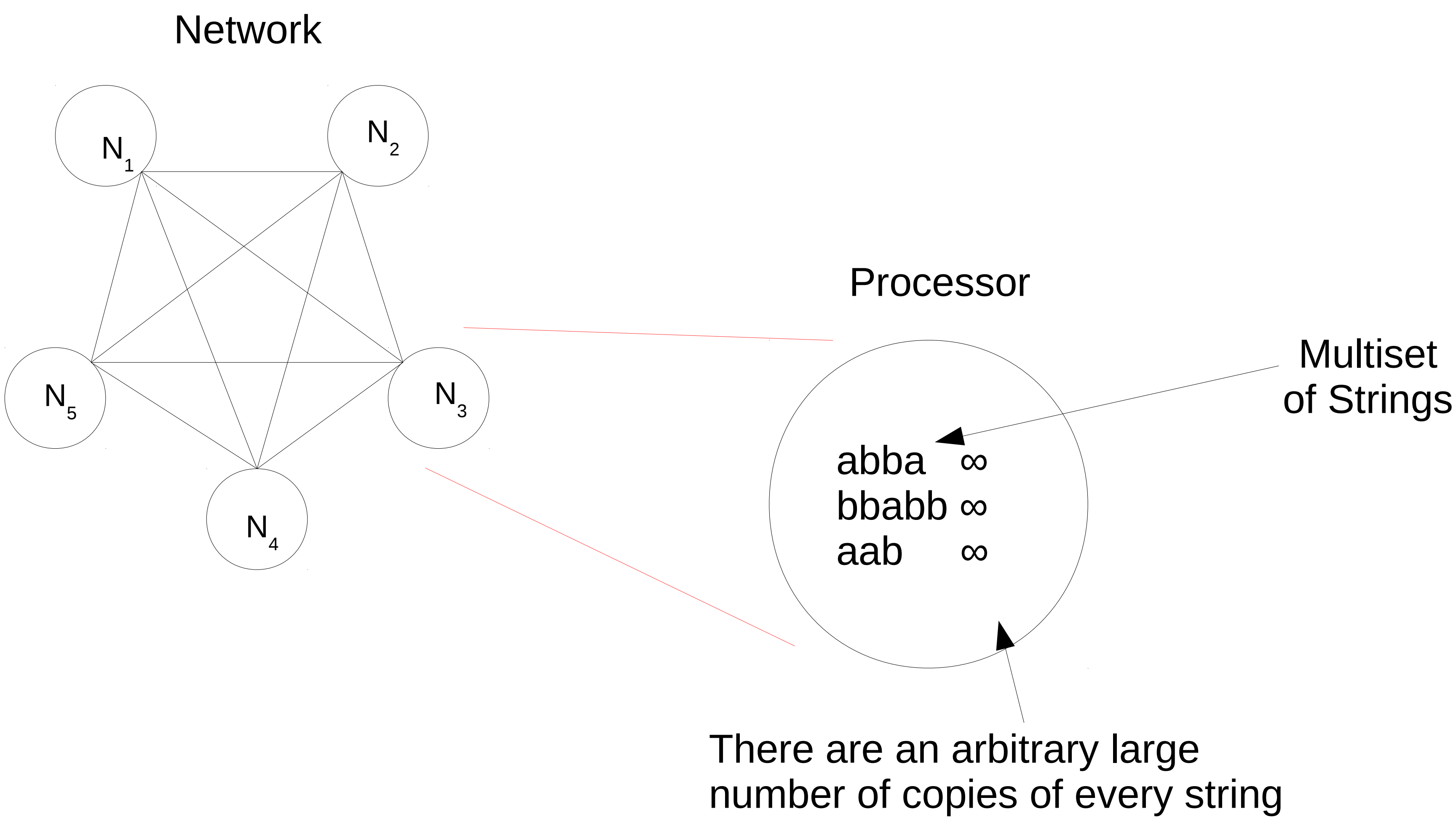
Networks of Genetic Processors as language generators

Marcelino Campos Francés

Departamento de Sistemas Informáticos y computación

Universidad Politécnica de Valencia

Networks of Genetic Processors



Mutation

Given the alphabet V , a **mutation** rule $a \rightarrow b$, with $a, b \in V$, can be applied over the string xay to produce the new string xby (observe that a mutation rule can be viewed as a substitution rule).

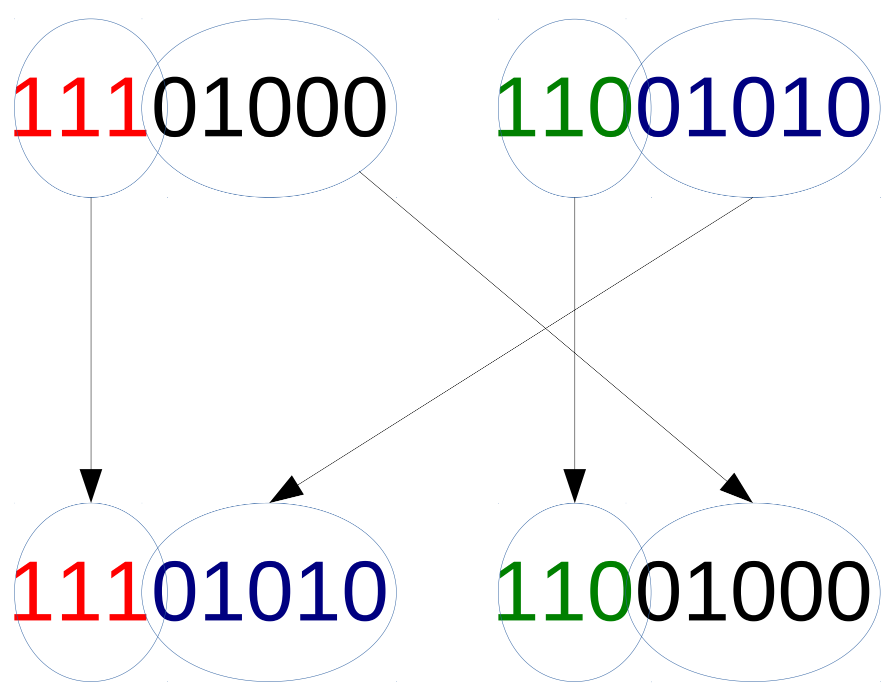
11101000

↓

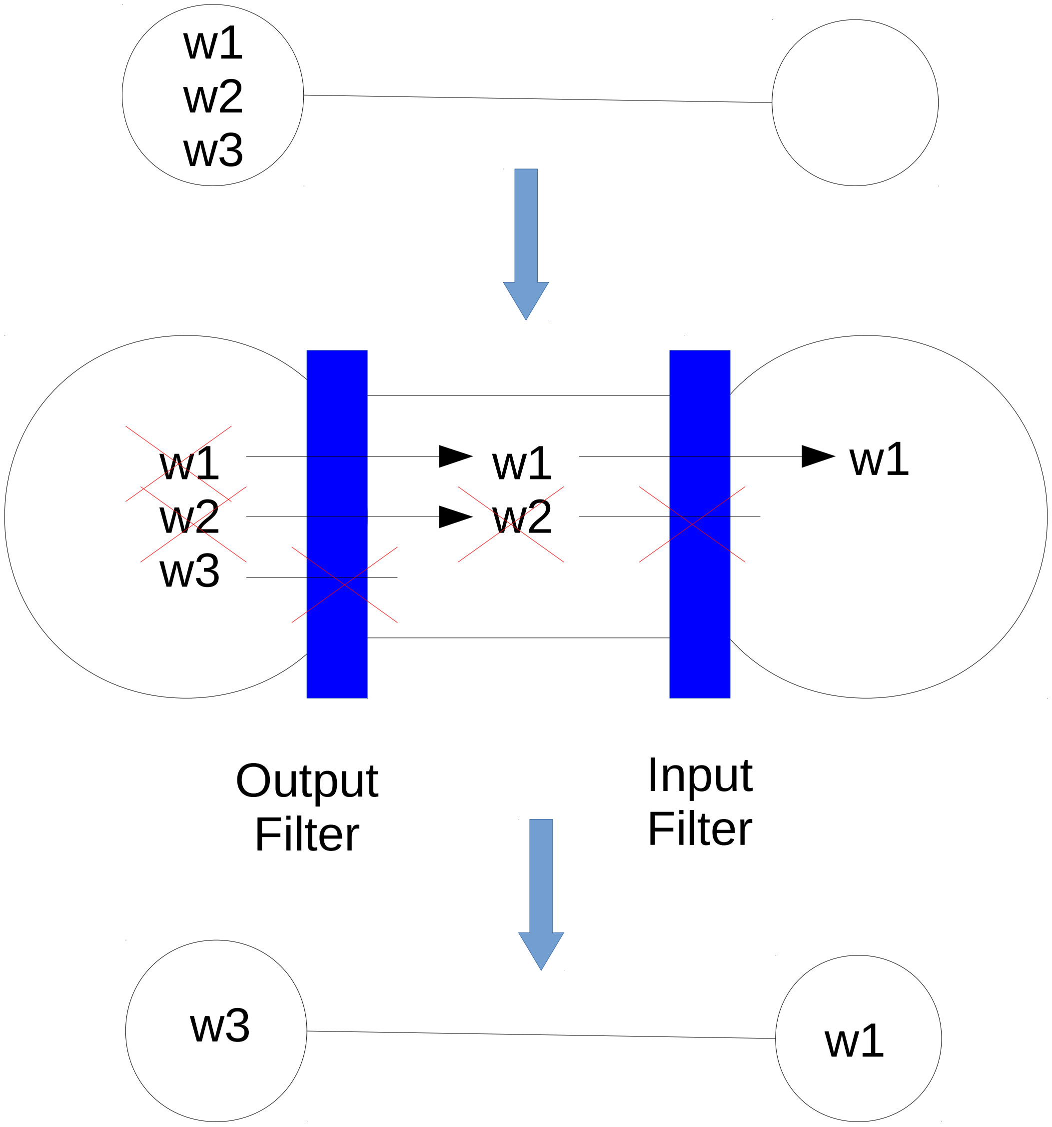
11001000

Crossover

A **crossover** operation is an operation over strings defined as follows: Let x and y be two strings, then $x \triangleright \triangleleft y = \{x_1 y_2 , y_1 x_2 : x = x_1 x_2 \text{ and } y = y_1 y_2\}$.



Filters



Predicates

Let P and F be two disjoint subsets of an alphabet V, and let $w \in V^*$. We define the predicates $\varphi^{(1)}$ and $\varphi^{(2)}$ as follows:

1. $\varphi^{(1)}(w, P, F) \equiv (P \subseteq alph(w)) \wedge (F \cap alph(w) = \emptyset)$ (strong predicate)
2. $\varphi^{(2)}(w, P, F) \equiv (alph(w) \cap P = \emptyset) \wedge (F \cap alph(w) = \emptyset)$ (weak predicate)

We can extend the previous predicates to act over segments instead of symbols. Let P and F be two disjoint sets of finite strings over V , and let $w \in V^*$. We extend the predicates $\varphi^{(1)}$ and $\varphi^{(2)}$ as follows:

1. $\varphi^{(1)}(w, P, F) \equiv (P \subseteq seg(w)) \wedge (F \cap seg(w) = \emptyset)$ (strong predicate)
2. $\varphi^{(2)}(w, P, F) \equiv (seg(w) \cap P = \emptyset) \wedge (F \cap seg(w) = \emptyset)$ (weak predicate)

Genetic Processor

Let V be an alphabet. A genetic processor over V is defined by the tuple $(\mathbf{MR},\mathbf{A},\mathbf{PI},\mathbf{FI},\mathbf{PO},\mathbf{FO},\boldsymbol{\alpha},\boldsymbol{\beta})$, where:

- **MR** is a finite set of mutation rules over V.
- **A** is a multiset of strings over V with a finite support and an arbitrary large number of copies of every string.
- **PI**, **FI** $\subseteq V^*$ are finite sets with the input permitting/forbidding contexts
- **PO**, **FO** $\subseteq V^*$ are finite sets with the output permitting/forbidding contexts
- $\boldsymbol{\alpha} \in \{1, 2\}$ defines the function mode with the following values:
  - If $\alpha = 1$ the processor applies mutation rules
  - If $\alpha = 2$ the processor applies crossover operations and $MR = \emptyset$
- $\boldsymbol{\beta} \in \{(1), (2)\}$ defines the type of the input/output filters of the processor. More precisely, for any word $w \in V^*$ we define an input filter $\rho(w) = \varphi^{\beta}(w,PI,FI)$ and an output filer $\tau(w) = \varphi^{\beta}(w,PO,FO)$. That is, $\rho(w)$ (resp. $\tau(w)$) indicates whether or not the word w passes the input (resp. the output) filter of the processor. We can extend the filters to act over languages. So, $\rho(L)$ (resp. $\tau(L)$) is the set of words of L that can pass the input (resp. output) filter of the processor.

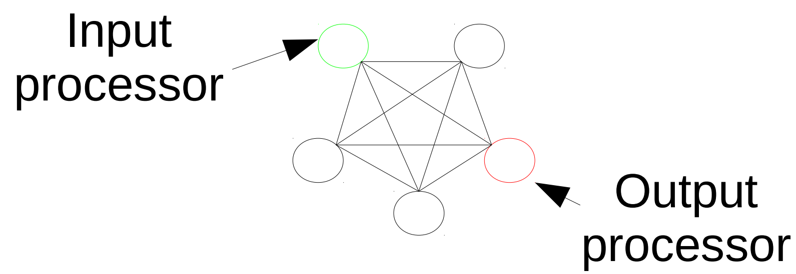
Networks of Genetic Processors

A Generating Network of Genetic Processors (GNGP) is defined by the tuple $\Pi = (\mathbf{V}, \mathbf{V}_{out}, \mathbf{N}_1, \mathbf{N}_2, \ldots, \mathbf{N}_n, \mathbf{G}, \mathbf{N}, \mathbf{N}_{out})$, where:

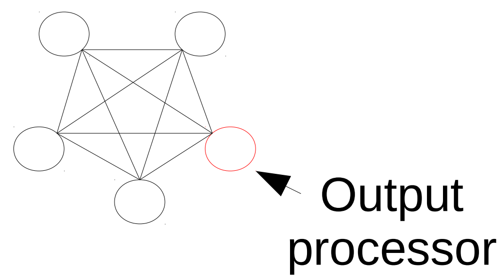
- $\mathbf{V}$ is an alphabet.
- $\mathbf{V}_{out} \subseteq V$ is an output alphabet.
- $\mathbf{N}_i$ $(1 \leq i \leq n)$ is a genetic processor over V.
- $\mathbf{G} = (X_G, E_G)$ is a graph.
- $\mathbf{N} : X_G \rightarrow \{N_1, N_2, \ldots, N_n\}$ is a mapping that associates the genetic processor N_i to the node $i \in X_G$
- $\mathbf{N}_{out} \in \{N_1, \cdots, N_n\}$ is the output processor.

Types of Networks of Genetic Processors

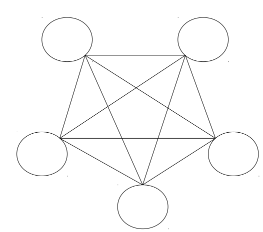
Accepting Networks:



Generating Network:



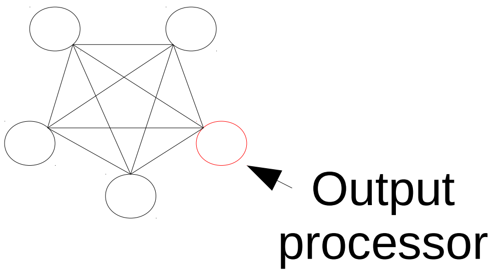
Networks as Genetic algorithms:



The filters implements the restrictions and the optimization function.

Types of Generating Networks of Genetic Processors

Generating Network:



There are two types of generating networks depending of the accepting criteria:

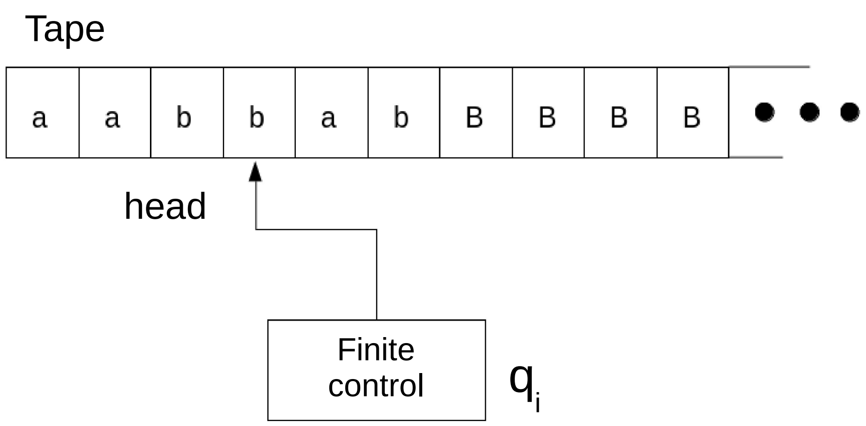
- Output node.
- Output node and output alphabet

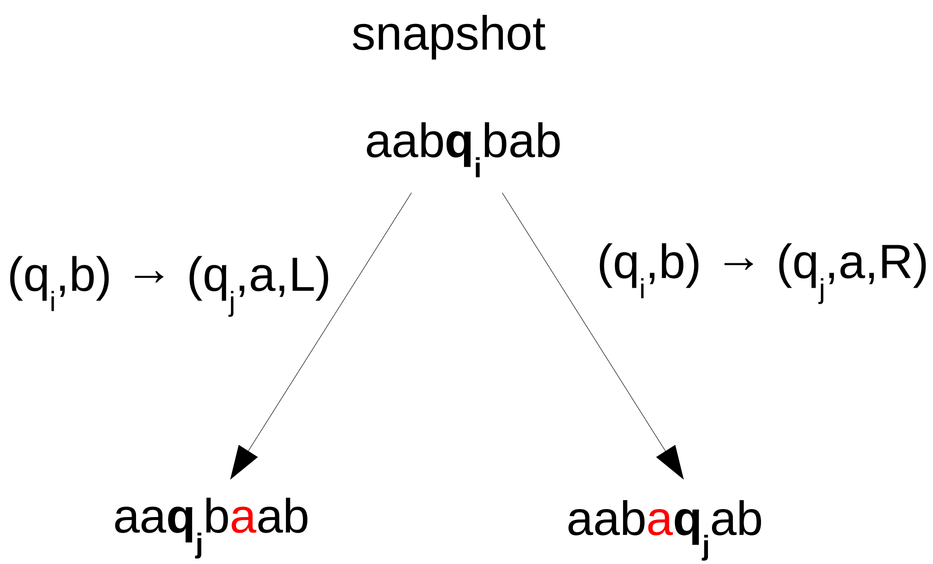
Networks of Genetic Processors are Computationally Complete

Theorem: Accepting Networks of Genetic Processors are computationally complete.

Networks of Genetic Processors are Computationally Complete

The proof will be based on the simulation of any arbitrary deterministic Turing machine during the computation of any input string.



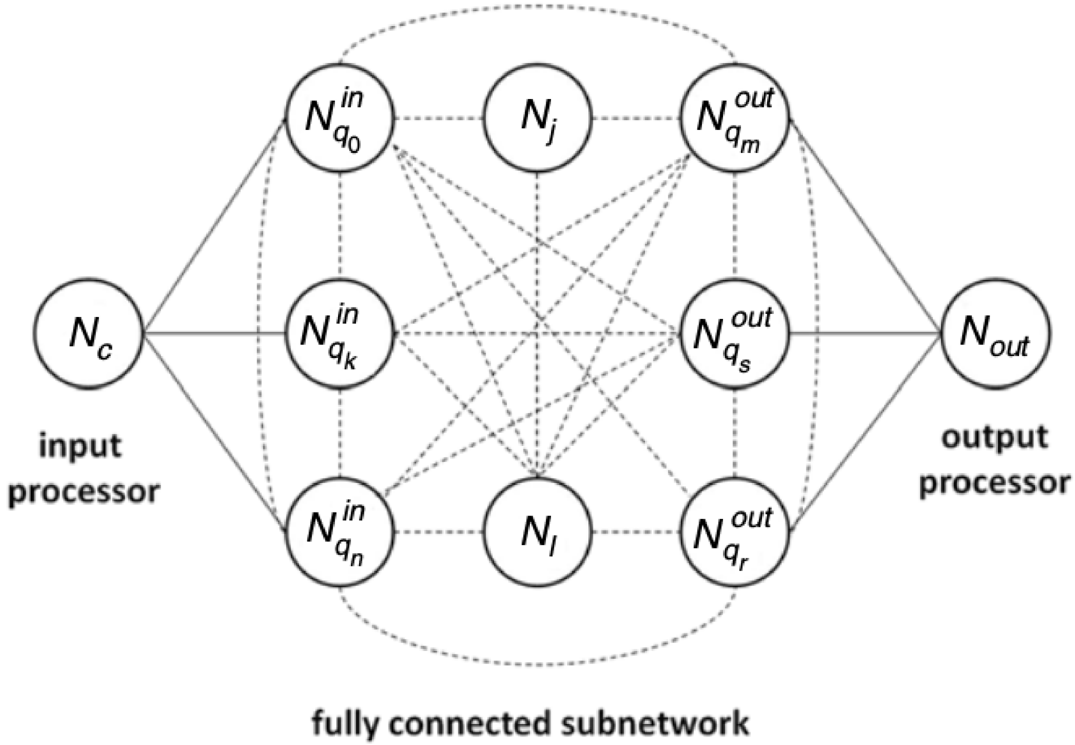


Networks of Genetic Processors are Computationally Complete

Snapshot codification.

$aab\mathbf{q}_i bab \longrightarrow \mathbf{q}_i aab\$babF$

Network:



Networks of Genetic Processors are Computationally Complete

Network Behavior:

- Acceptance criteria:

When a snapshot with a final state appears, the string will enter into the corresponding N_q^{out} processor, this processor will send the string to N_{out} and the computation halts in an acceptance mode.

- Rejects:

There are two situations in wich the computation rejects: when it doesn't exist defined movement and when the head is at the first cell of the tape and the machine tries to do a left movement. In both cases the snapshot does not get into any processor, so the process is interrupted and in a finite number of steps we will have two consecutive steps with the same chains in the same processors, this will stop the computation and the initial string will be rejected.

- Infinite computation:

The network also performs an infinite computation, and the input string will never be accepted.

Networks of Genetic Processors are Computationally Complete

Theorem: Every nondeterministic Turing machine can be simulated by an ANGP.

The process is the same that in the deterministic way, but in this case a snapshot can enter more than one processor at a time. On the other hand if two snapshots enter the same processor, the rules will be applied independently.

Acceptance criteria:

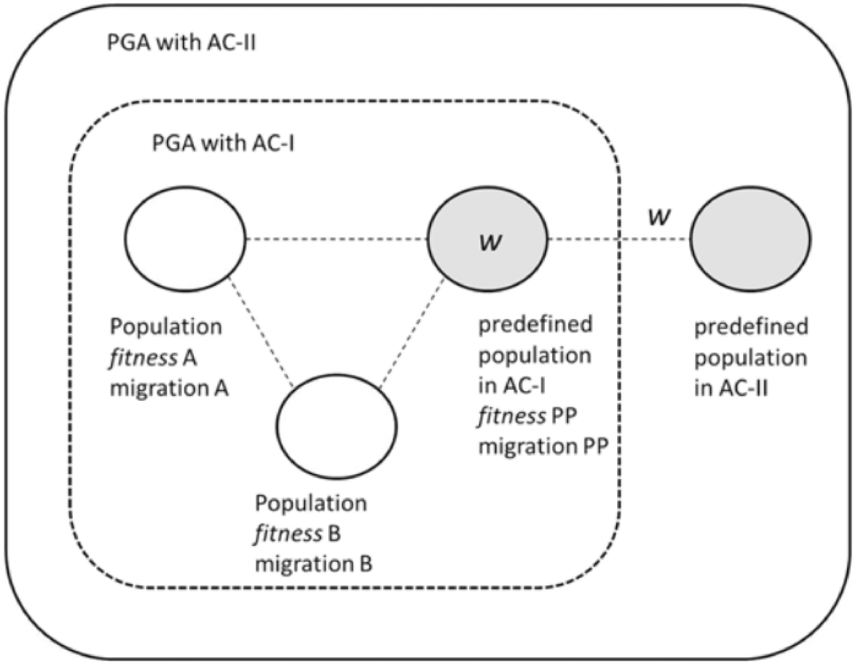
- Acceptance criterion I (AC-I):

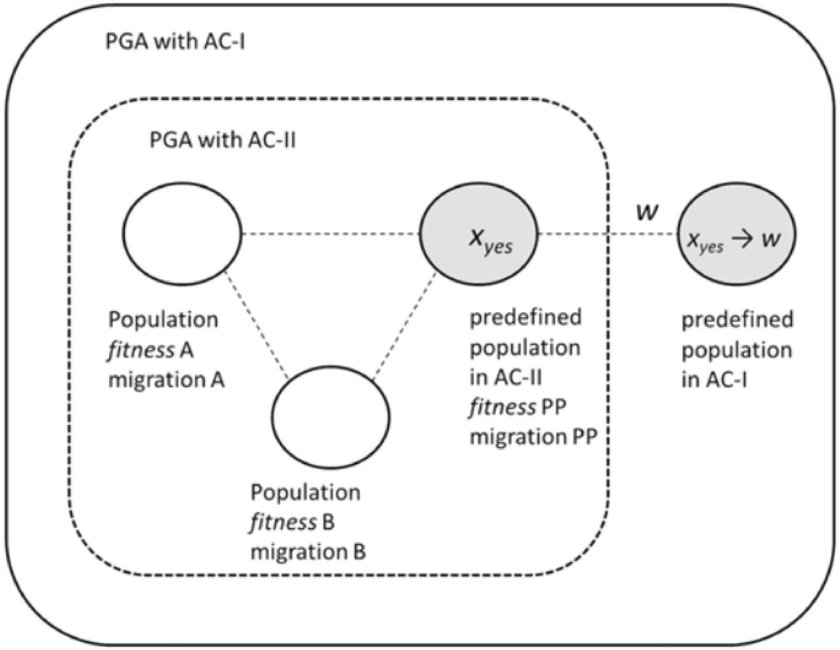
Let w be an input string. We say that a PGA accepts w if w appears in a predefined survival population after a finite number of iterations (operators applications, fitness selection, and individuals migration).

- Acceptance criterion II (AC-II):

Let w be an input string. We say that a PGA accepts w if a distinguished individual x_{yes} appears in a predefined survival population after a finite number of iterations (operators applications, fitness selection, and individual migration). We say that the PGA rejects the input string if a distinguished individual x_{not} appears in a predefined survival population after a finite number of iterations (operators applications, fitness selection, and individual migration).

Both acceptance criteria are equivalent:





Multiple Populations: The crossover operations in one population are made with string of the same population.

Synchronicity and Full Migration Phenomena: In one step all the solutions are transmitted at the same time.

We can define consider a ANGP like a PGA with multiple populations, synchronicity, and full migration phenomena.

Theorem: Parallel Genetic Algorithms with multiple populations, synchronicity, and full migration phenomena are computationally complete.

The Chomsky's Hierarchy

$$REG \subset CF \subset CS \subset RE$$

REG: Regular grammars

CF: Context-free grammars

CS: Context-sensitive grammars

RE: Phrase structure grammars

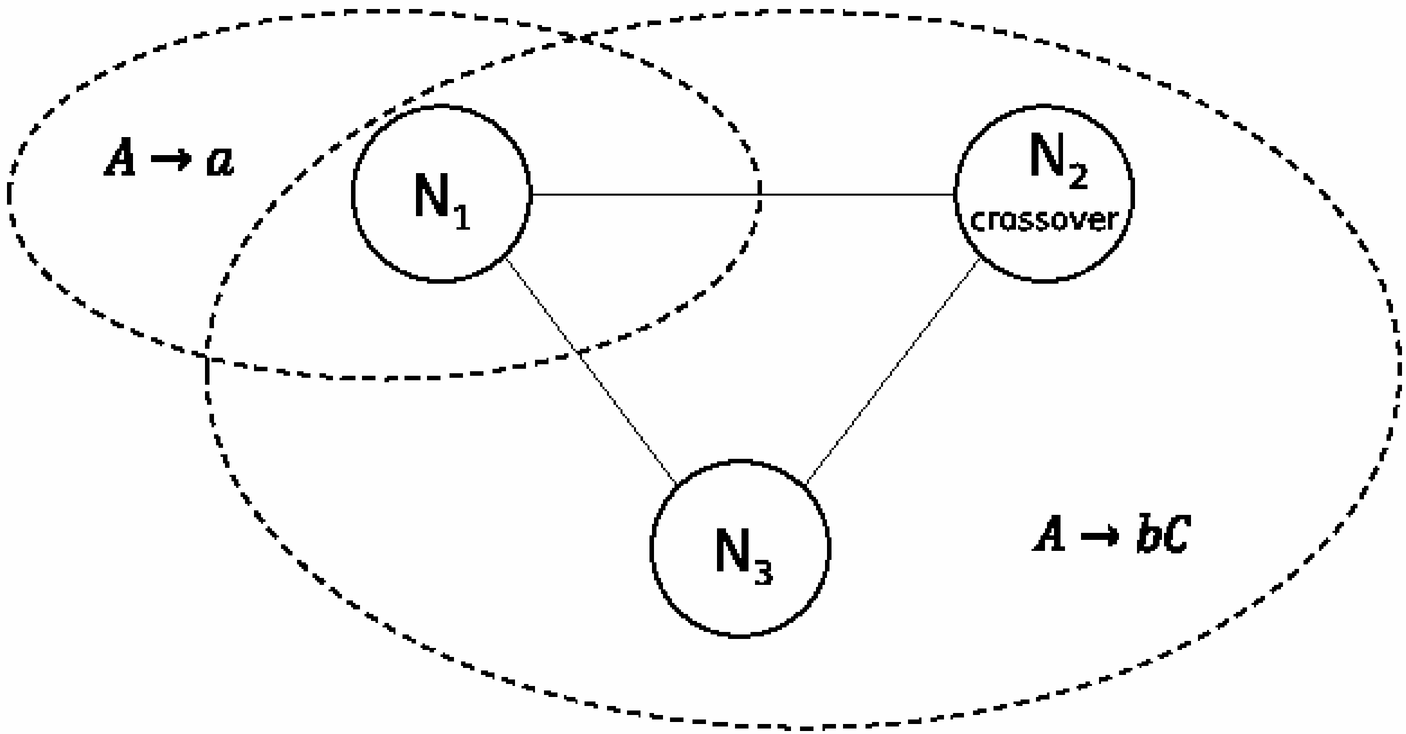
Regular Grammars

Regular grammars (right linear grammars):

- $A \to aB$, with $A, B \in N$ and $a \in T$
- $A \to a$, with $A \in N$ and $a \in T \cup \{\varepsilon\}$

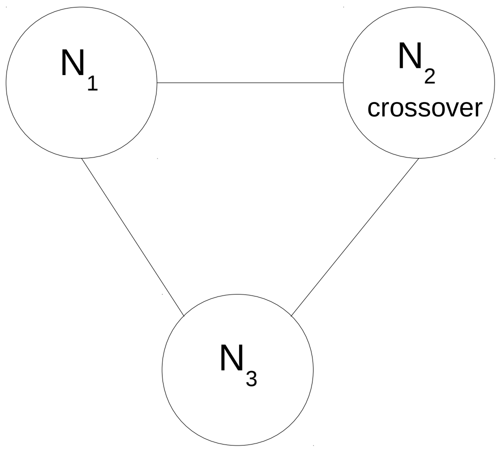
Theorem: Every regular language can be generated by a GNGP with 3 processors.

Topology for Regular Grammars



Topology for Regular Grammars

$A \rightarrow a$
$A \rightarrow bC$



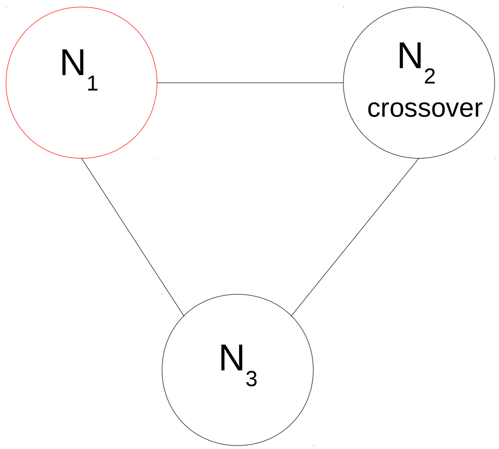
Topology for Regular Grammars

A → a
A → bC

abbaA

↓

abbaa



Topology for Regular Grammars

A → a
A → bC

N_1

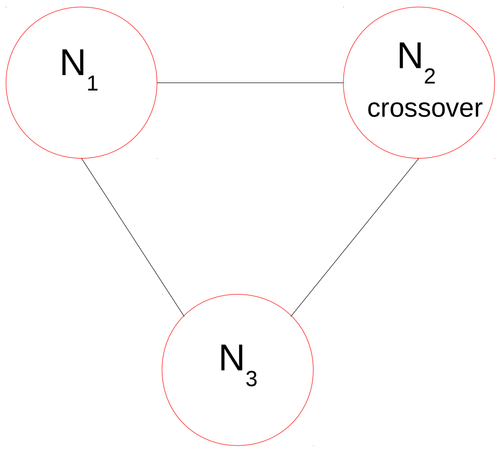
abbaA
↓
abba[bC]

N_2

abba[bC] Ĉ
↘ ↙ crossover
abba[bC]Ĉ

N_3

abba[bC]Ĉ
↓
abbabC

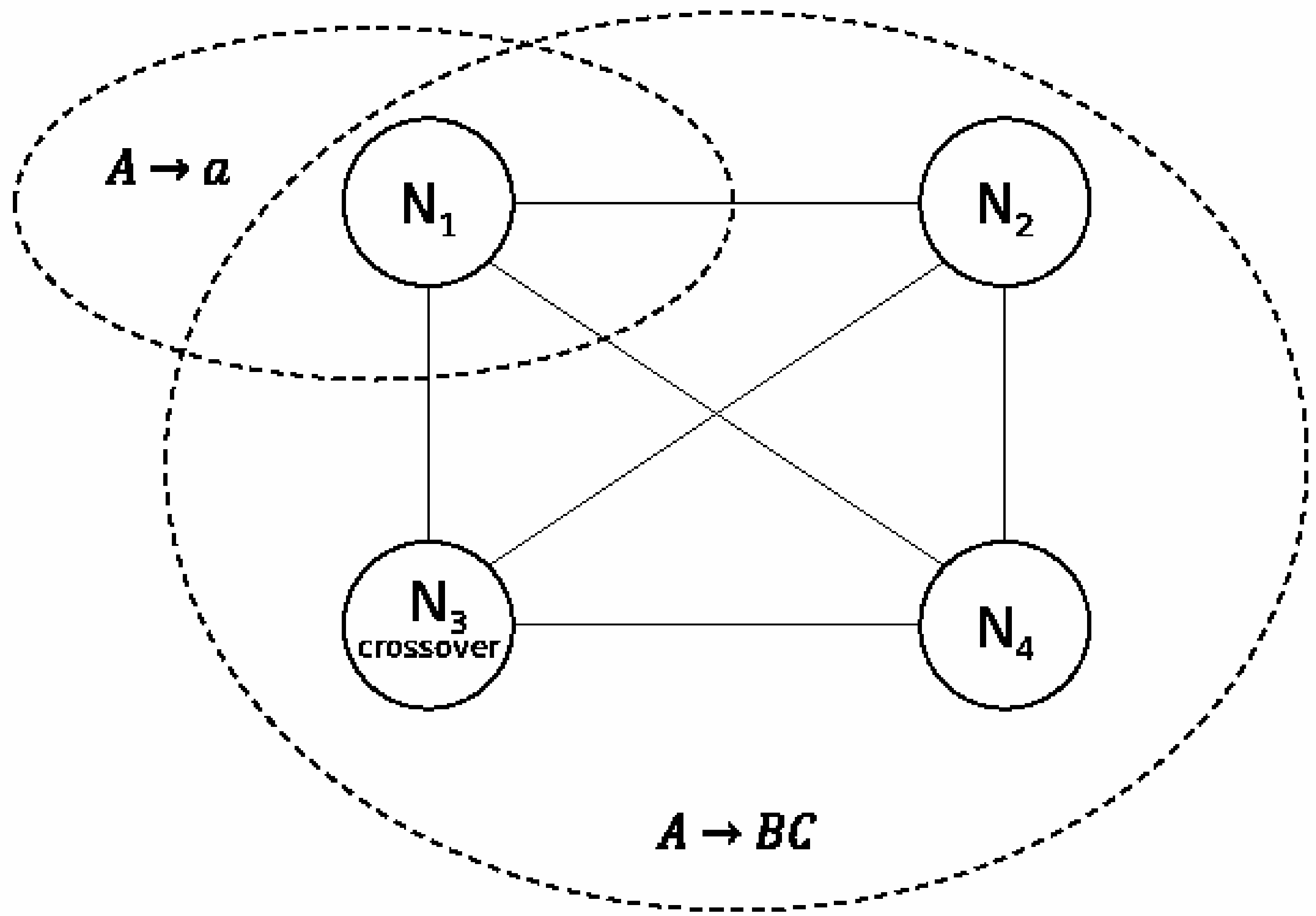


Context-free grammars (Chomsky Normal Form):

- $A \rightarrow BC$, with $A, B, C \in N$
- $A \rightarrow a$, with $A \in N$ and $a \in T$

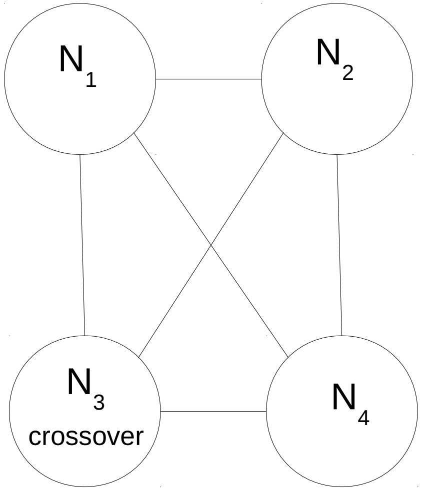
Theorem: Every context-free language can be generated by a GNGP with 4 processors.

Topology for Context-free Grammars



Topology for Context-free Grammars

$A \rightarrow a$
$A \rightarrow BC$



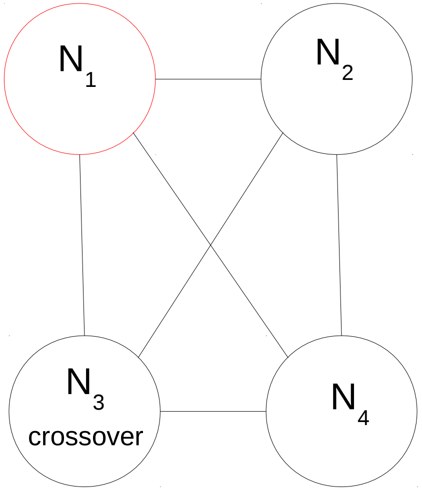
Topology for Context-free Grammars

$A \rightarrow a$
$A \rightarrow BC$

aBbaABa

↓

aBbaaBa



Topology for Context-free Grammars

A → a

A → BC

N_1

aAbBa

↓

a[[BC]]bBa

N_2

a[[BC]]bBa

↓

a[[BC]][Cb][bB][Ba]'

N_3

a[[BC]][Cb][bB][Ba]'   â

crossover

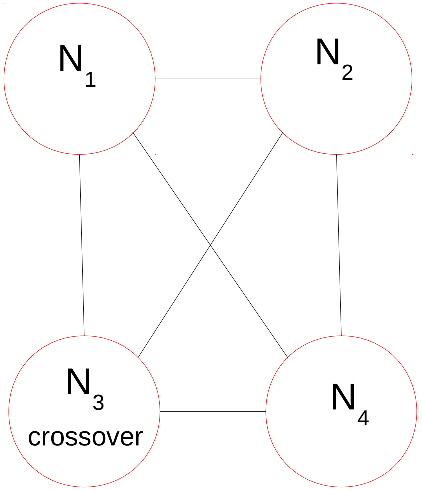
a[[BC]][Cb][bB][Ba]'â

N_4

a[[BC]][Cb][bB][Ba]'â

↓

aBCbBa



Context-sensitive grammars (Kuroda Normal Form):

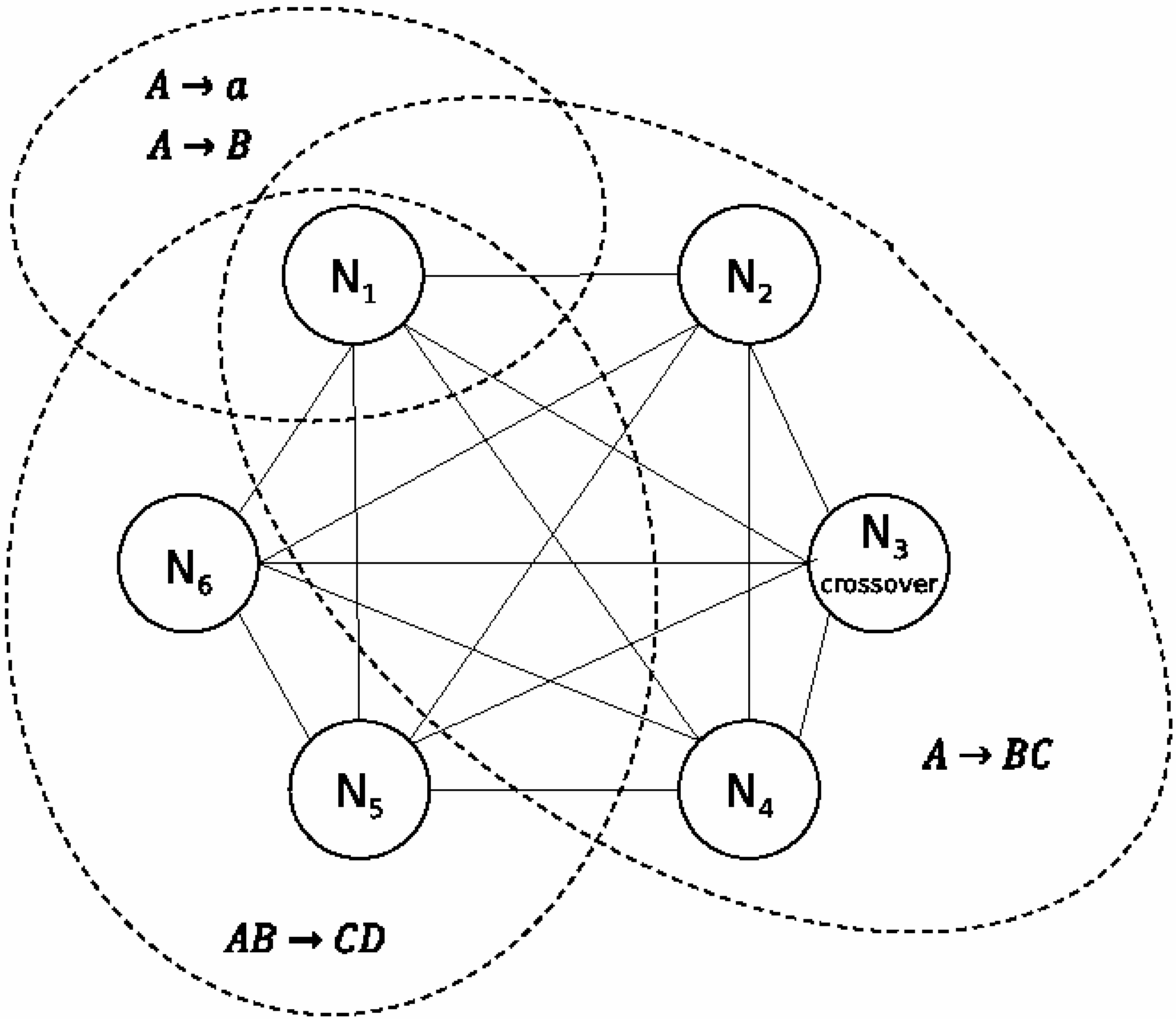
- $A \rightarrow a$, with $A \in N$ and $a \in T$
- $A \rightarrow B$, with $A, B \in N$
- $A \rightarrow BC$ with $A, B, C \in N$
- $AB \rightarrow CD$ with $A, B, C, D \in N$

In addition, we can add the production $S \rightarrow \varepsilon$, whenever S does not appear to the right side of any production. In such a case, the grammar can generate the empty string.

Context-sensitive Grammars

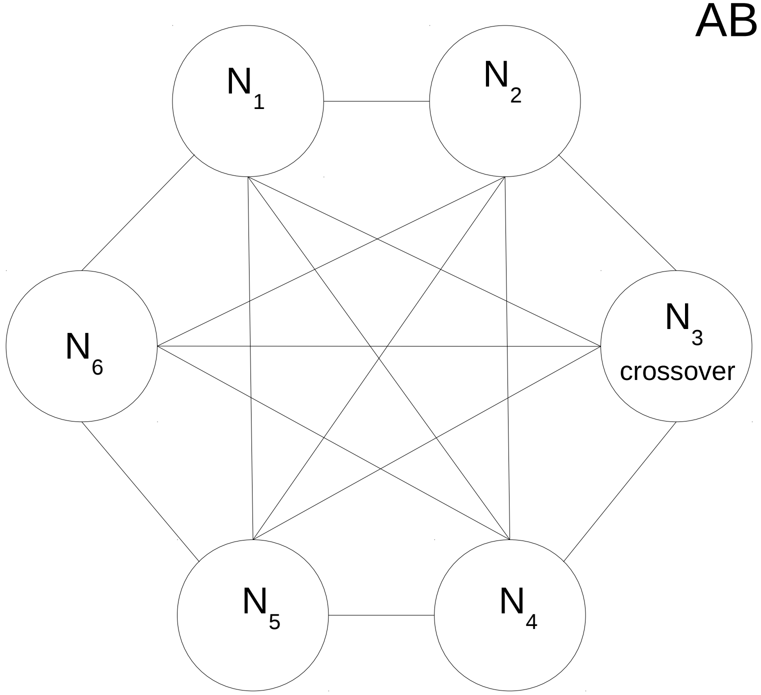
Theorem: Every context-sensitive language can be generated by a GNGP with 6 processors.

Topology for Context-sensitive Grammars



Topology for Context-sensitive Grammars

A → a
A → B
A → BC
AB → CD



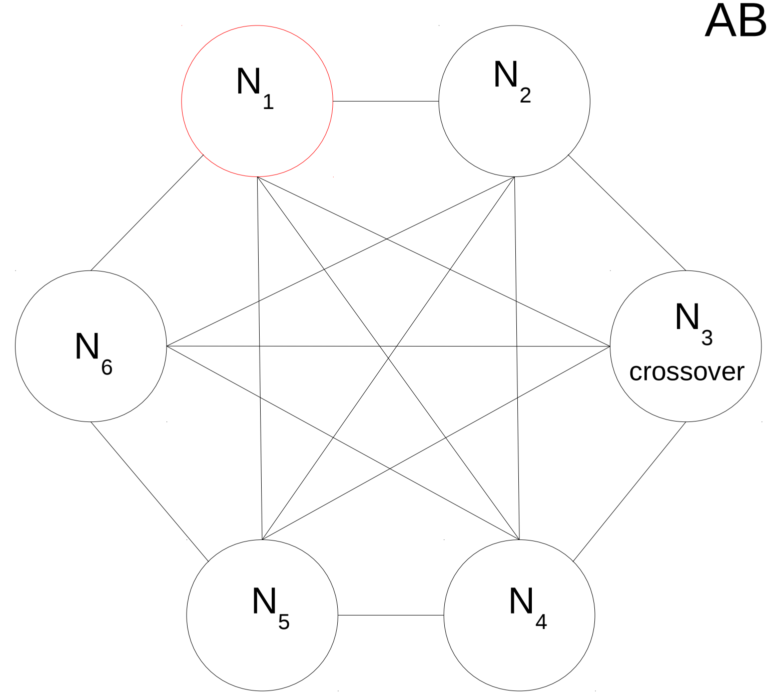
Topology for Context-sensitive Grammars

$A \rightarrow a$
$A \rightarrow B$
$A \rightarrow BC$
$AB \rightarrow CD$

aBbaABa

↓

aBbaaBa



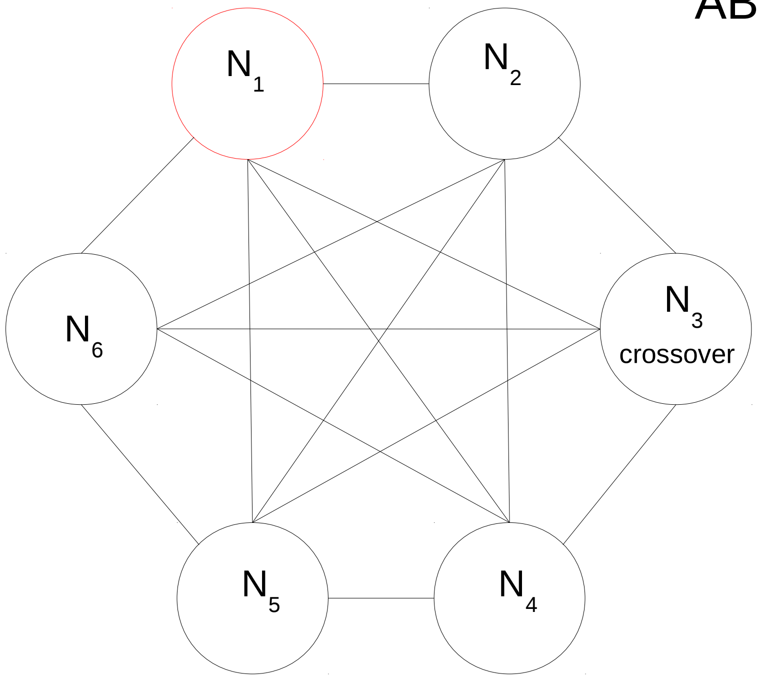
Topology for Context-sensitive Grammars

A → a
A → B
A → BC
AB → CD

aBbaABa

↓

aBbaBBa



Topology for Context-sensitive Grammars

A → a
A → B
A → BC
AB → CD

N_1

aAbBa
↓
a[[BC]]bBa

N_2

a[[BC]]bBa
↓
a[[BC]][Cb][bB][Ba]'

N_3

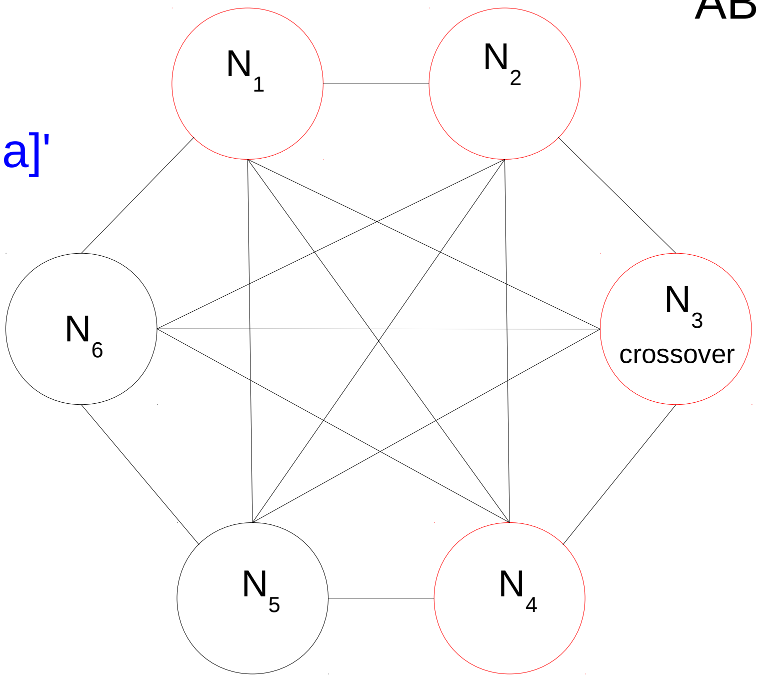
a[[BC]][Cb][bB][Ba]'   â

crossover

a[[BC]][Cb][bB][Ba]'â

N_4

a[[BC]][Cb][bB][Ba]'â
↓
aBCbBa



Topology for Context-sensitive Grammars

A → a
A → B
A → BC
AB → CD

N_1

aCABaC
↓
aC[[ACD]BaC

---

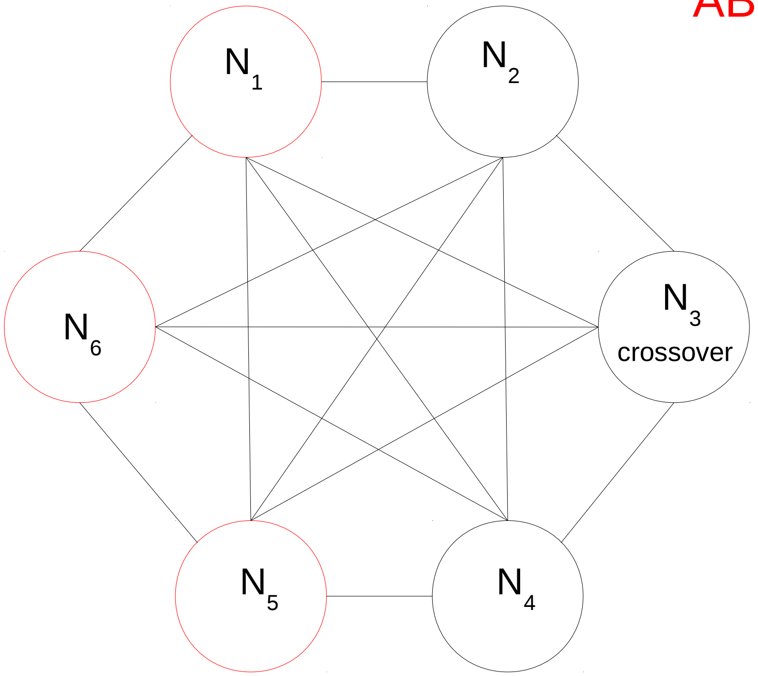
aC[[ACD]BaC

N_5
↓
aC[[ACD][BCD]]aC

---

aC[[ACD][BCD]]aC

N_6
↓
aCCDaC

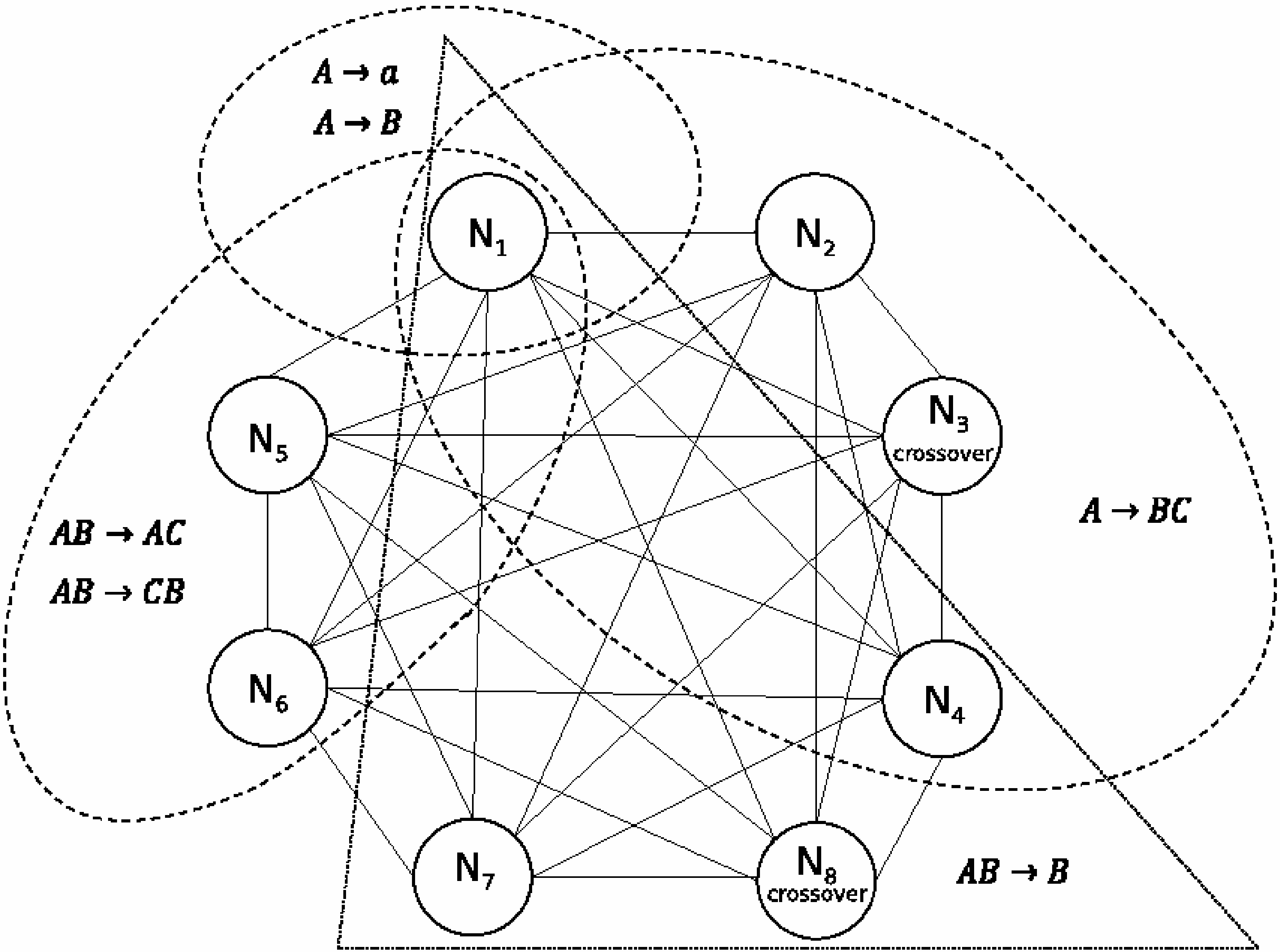


Phrase structure grammars (extended Kuroda Normal Form):

- $S \rightarrow \varepsilon$
- $A \rightarrow a$, with $A \in N$ and $a \in T$
- $A \rightarrow B$, with $A, B \in N$
- $A \rightarrow BC$ with $A, B, C \in N$
- $AB \rightarrow AC$ with $A, B, C \in N$
- $AB \rightarrow CB$, with $A, B, C \in N$
- $AB \rightarrow B$, with $A, B \in N$

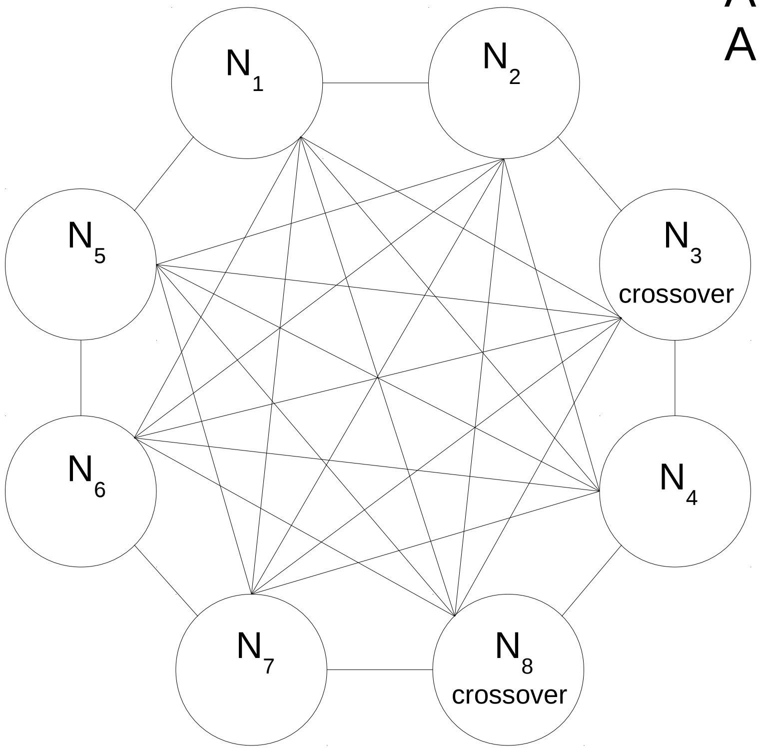
Theorem: Every recursively enumerable language can be generated by a GNGP with 8 processors.

Topology for Phrase Structure Grammars



Topology for Phrase Structure Grammars

$A \rightarrow a$
$A \rightarrow B$
$A \rightarrow BC$
$AB \rightarrow AC$
$AB \rightarrow CB$
$AB \rightarrow B$



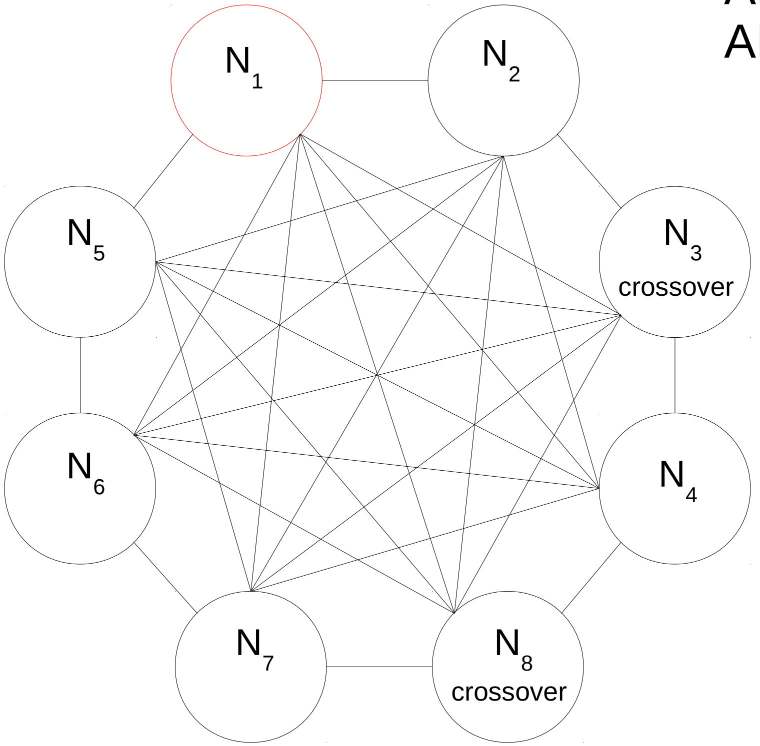
Topology for Phrase Structure Grammars

A → a
A → B
A → BC
AB → AC
AB → CB
AB → B

aBbaABa

↓

aBbaaBa



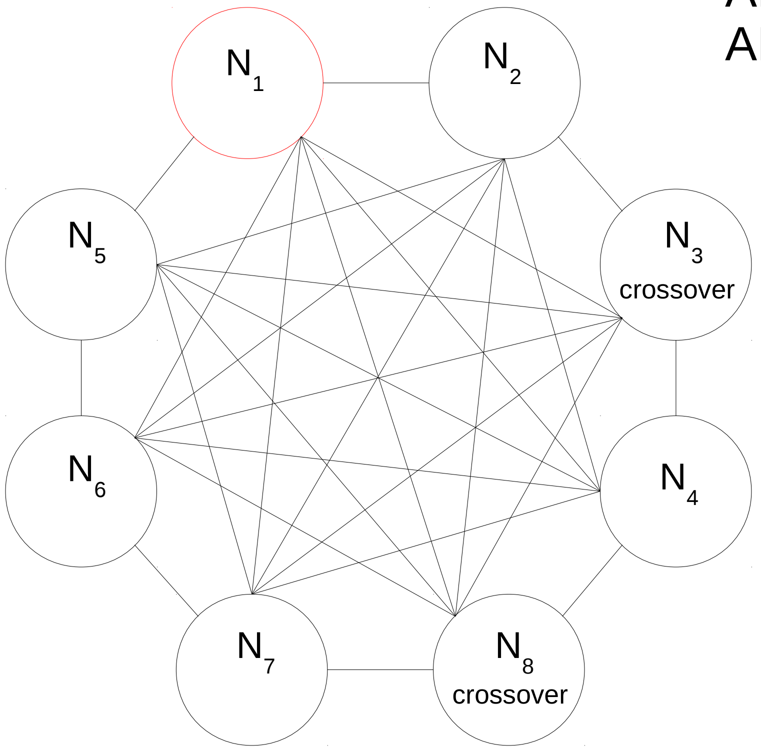
Topology for Phrase Structure Grammars

A → a
A → B
A → BC
AB → AC
AB → CB
AB → B

aBbaABa

↓

aBbaBBa



Topology for Phrase Structure Grammars

N_1

aAbBa

↓

a[[BC]]bBa

N_2

a[[BC]]bBa

↓

N_3

a[[BC]][Cb][bB][Ba]' â

crossover

a[[BC]][Cb][bB][Ba]'â

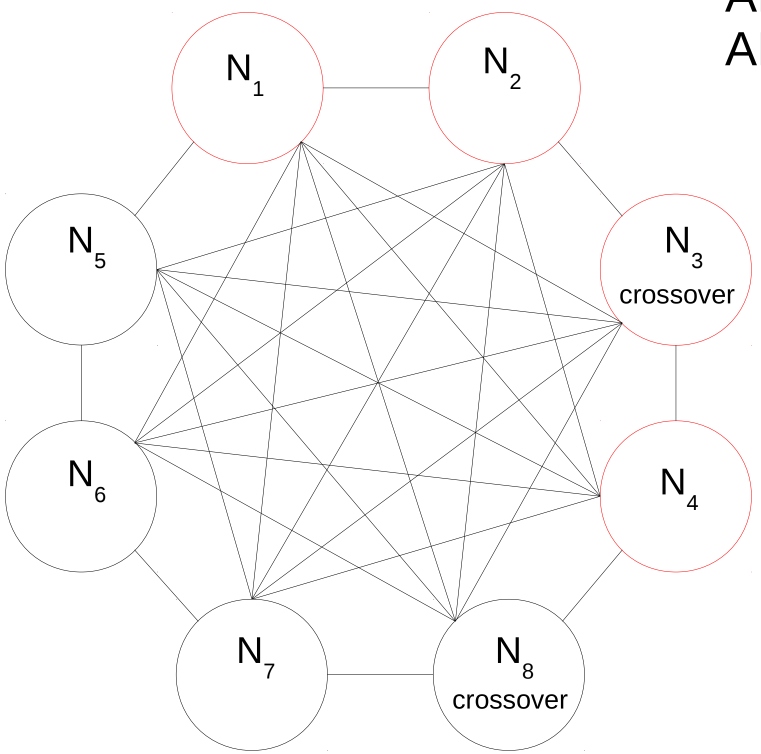
N_4

a[[BC]][Cb][bB][Ba]'â

↓

aBCbBa

$A \rightarrow a$
$A \rightarrow B$
$A \rightarrow BC$
$AB \rightarrow AC$
$AB \rightarrow CB$
$AB \rightarrow B$



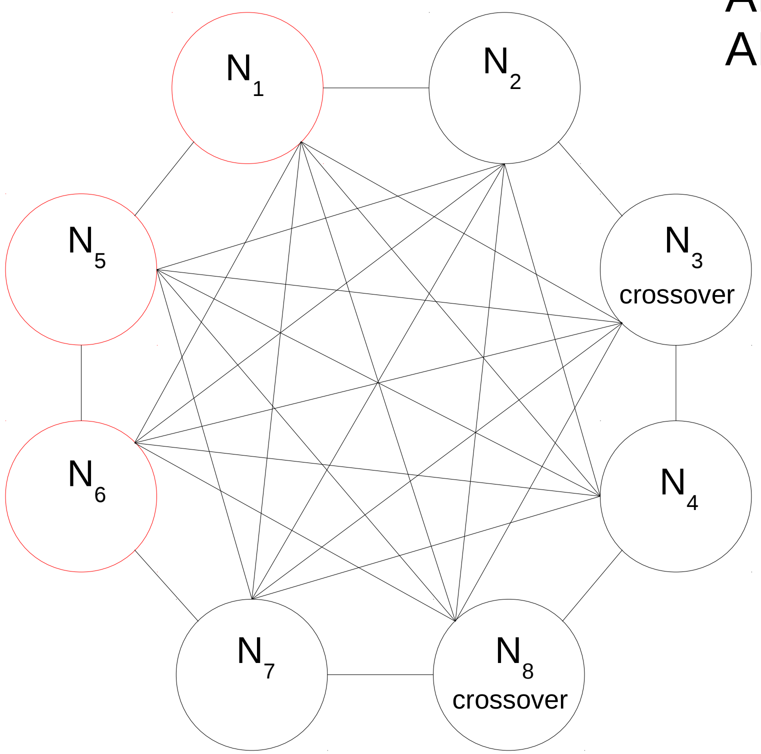
Topology for Phrase Structure Grammars

A → a
A → B
A → BC
AB → AC
AB → CB
AB → B

N_1 aCABaC
↓
aC[[AAC]]BaC

N_5 aC[[AAC]]BaC
↓
aC[[AAC]][BAC]]aC

N_6 aC[[AAC]][BAC]]aC
↓
aCACaC



Topology for Phrase Structure Grammars

A → a
A → B
A → BC
AB → AC
AB → CB
AB → B

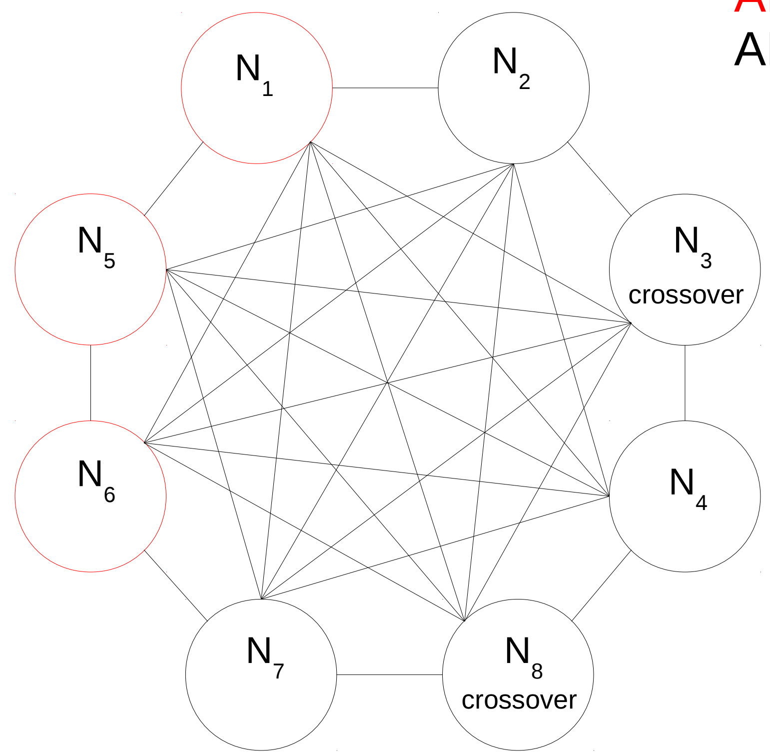
N_1 aCABaC
↓
aC[[ACB]]BaC

---

aC[[ACB]]BaC
N_5 ↓
aC[[ACB]][BCB]]aC

---

aC[[ACB]][BCB]]aC
N_6 ↓
aCCBaC



Topology for Phrase Structure Grammars

A → a
A → B
A → BC
AB → AC
AB → CB
AB → B

aABbC

↓

N_1 a<<AB>>BbC

N_7 a<<AB>>BbC

↓

a<<AB>><Bb><bC><CX>'

N_8 a<<AB>><Bb><bC><CX>'

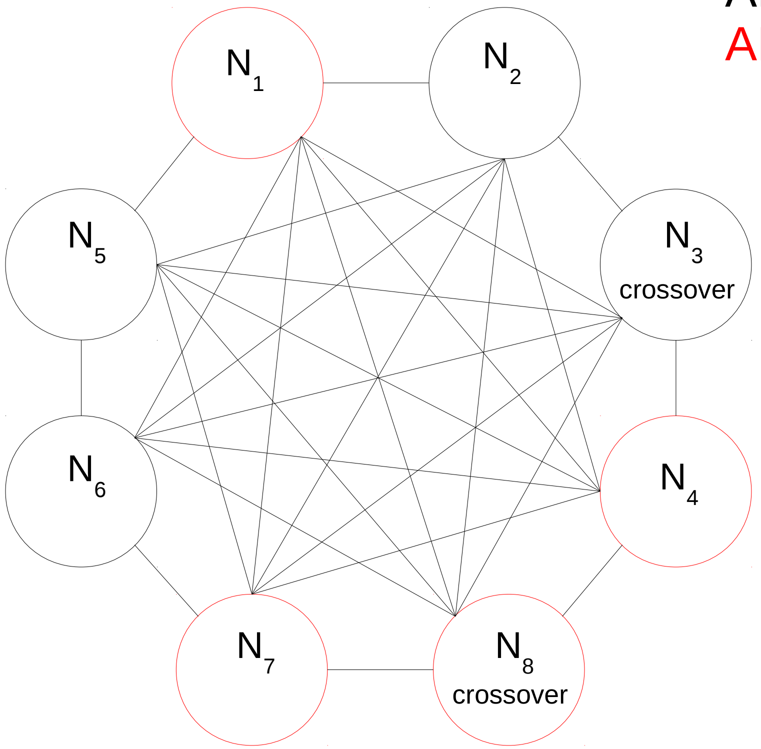
↓ crossover

a<<AB>><Bb><bC>

N_4 a<<AB>><Bb><bC>

↓

aBbC



Publications

●Marcelino Campos, José M. Sempere.
A characterization of formal languages through Networks of Genetic Processors.
(submitted)

●Solving Combinatorial Problems with Networks of Genetic Processors.
International Journal “Information Technologies and Knowledge”
Vol.7 No. 1, pp 65-71. 2013

●Marcelino Campos, José M. Sempere.
Accepting Networks of Genetic Processors are computationally complete.
Theoretical Computer Science Vol. 456, pp 18-29. 2012

●M. Campos, J. González, T.A. Pérez, J. M. Sempere.
Implementing Evolutionary Processors in JAVA: A case study.
13th International Symposium on Artificial Life and Robotics (AROB 2008)
(Beppu, Japan) January 31 - February 2.
2008 Proceedings edited by M. Sugisaka and H. Tanaka pp 510-515.
2008 ISBN: 978-4-9902880-2-0

Thank you

¿Questions?