Real Life Applications of Bio-inspired Computing Models: EAP and NEPs

PhD Thesis

Emilio del Rosal García

Supervisor: Alfonso Ortega de la Puente

Escuela Politécnica Superior
Universidad Autónoma de Madrid
C/ Francisco Tomás y Valiente 11
-28049- Madrid, Spain

A todos aquellos que cometen la insensatez de quererme tal y como soy.

A mi familia y amigos.

"There is only one good, knowledge, and one evil, ignorance."

Socrates

Contents

I Introduction 12

1 Preamble 13
1.1 Recuerdos y agradecimientos 13
1.2 Resumen 14
1.3 Context and motivation 16
1.4 Brief description of the contents 16

2 State of the art 18
2.1 Introduction to natural computing 18
2.1.1 Algorithms inspired by nature 20
2.1.1.1 Evolutionary computing 20
2.1.1.2 Neurocomputing 25
2.1.2 Computing models inspired by nature 28
2.1.2.1 L-systems 29
2.1.2.2 Cellular automata 31
2.1.2.3 P-Systems 32
2.1.3 Nature as hardware 34
2.1.3.1 DNA computing 34
2.1.3.2 Quantum computing 36
2.2 Evolutionary Automatic Programming and Grammatical Evolution 37
2.2.1 Genetic Programming and its weaknesses 37
2.2.1.1 How GP works 38
2.2.2 Grammatical Evolution 39
2.2.2.1 The Grammatical Evolution functioning 41
2.2.2.2 Discussion and comments 44
2.2.3 Advances in the Grammatical Evolution framework 45
2.2.3.1 Attribute Grammar Evolution 45
2.2.3.2 Christiansen Grammar Evolution 47
2.3 NEP 48
2.3.1 NEPs in practice 52
2.4 Two scientific challenges 53
2.4.1 Modelling associative learning in psychology 53
2.4.1.1 Different types of learning 53
2.4.1.2 Current models and their problems 55
2.4.1.3 Characteristics of habituation 55
2.4.1.4 Review of current models 56
2.4.1.5 Other models related to habituation 57
2.4.1.6 Summing up 58
2.4.2 Language Processing 58

II Advances on bio-inspired computing models 60

3 Simulating and programming NEPs 61
3.1 jNEP . . . 61
3.1.1 jNEP design . . . 61
3.1.2 jNEP in practice . . . 64
3.1.2.1 An example . . . 66
3.1.3 jNEPView . . . 68
3.1.3.1 jNEPView design . . . 69
3.1.3.2 jNEPView example . . . 69
3.1.4 A Visual Language for Modelling and Simulation of NEPs . . 70
3.1.4.1 Introduction to AToM^3 . . . 70
3.1.4.2 NEPs visual language . . . 72
3.1.4.3 User point of view - How domain experts use it . . . 73
3.1.5 NEPs-Lingua . . . 75
3.1.5.1 The NEPs-Lingua syntax . . . 75
3.1.5.2 Examples . . . 77
3.1.5.3 NEPs Lingua semantics . . . 78
3.1.6 Final comments . . . 78

4 NEP's applications 80
4.1 Solving NP-complete problems with jNEP . . . 80
4.1.0.1 Solving the SAT problem with linear resources . . . 80
4.1.0.2 Hamiltonian path problem . . . 83
4.1.0.3 Coloring problems . . . 85
4.1.0.4 Final comments . . . 88
4.2 NEPs for parsing . . . 89
4.2.1 Efficiency improvements . . . 94
4.2.1.1 Formal description . . . 95
4.2.1.2 jNEP description of PNEPs . . . 96
4.2.2 On PNEP temporal complexity . . . 97
4.3 Natural language parsing with PNEPs . . . 98
4.3.1 An example . . . 99
4.4 PNEP and shallow parsing: PNEP in a real natural language context 107
4.4.1 Introduction to FreeLing and shallow parsing . . . 107
4.4.2 PNEP extension for shallow parsing . . . 108
4.4.3 Our PNEP for the FreeLing's Spanish grammar . . . 110
4.4.4 Final comments . . . 111

5 Automatic modelling 114
5.1 Automatic modelling of habituation . . . 114
5.1.0.1 General prior assumptions and constraints . . . 115
5.1.0.2 Proposed solution through Grammatical Evolution 116
5.1.0.3 Experiments . . . 123
5.1.0.4 Experiments program . . . 124
5.1.0.5 Results . . . 124
5.1.0.6 A look on two of the models . . . 124
5.2 Automatic programming of NEPs . . . 125
5.2.1 Motivation . . . 125
5.2.2 Automatic programming of NEPs . . . 131
5.2.3 The NEPs to search for . . . 132
5.2.4 Testing the framework . . . 133
5.2.5 The complete solution. . . . 135
5.2.5.1 Introduction to NEPs to rotate strings . . . 135

5.2.5.2 Our solution 136
5.2.5.3 Experiments and results 137
5.2.5.4 Conclusions and further research lines 144
5.3 A methodology for automatic modelling 145

6 Conclussions and final comments 147

A Configuration file for the 3 variables SAT 150

B Config file for the hamiltonian problem 154

C Config file for the coloring problem 156

D Config file for the shallow parsing PNEP 168

List of Tables

1.1 Publications sorted by date and type 17

2.1 Eight-Queens genetic algorithm general description 23
2.2 Output of the L-system at each iteration. 29
2.3 Transition function of the Cellular Automaton 31
2.4 Habituation models comparison 57

List of Figures

2.1 General scheme of an evolutionary algorithm. Based on Eiben and Smith [2003] . . . 22
2.2 The construction of the classical Sierpinski fractal. The actual fractal is the result of an infinite number of iterations. The image was taken from *Wikimedia Commons* and is in the public domain. . . . 30
2.3 The Peano curve's formation after three iterations. The actual fractal is the result of an infinite number of iterations and its limit is a space-filling curve. . . . 30
2.4 Eight first steps of our example Cellular Automata. The initial state is "1" for the center cell and "0" for the rest. . . . 32
2.5 An example P-System. The image was taken from *Wikimedia Commons* and is in the public domain. . . . 33
2.6 Example trees for two simple LISP expressions. . . . 38
2.7 Closure problem in GP . . . 40
2.8 Derivation tree for the expression "1 + sen (X)" . . . 49

3.1 Simplified class diagram of jNEP . . . 62
3.2 Window that shows the layout of the simulated NEP . . . 69
3.3 Initial simulation step . . . 70
3.4 Next simulation step . . . 71
3.5 Second simulation step . . . 72
3.6 End of simulation . . . 73
3.7 The meta-model UML class diagram . . . 74
3.8 The visual language in action. . . . 74

4.1 Graph studied by Adleman . . . 84
4.2 Example of a map and its adjacency graph. In this case, there is no solution for the 3-colorability problem . . . 85
4.3 Sequence of steps in the solution of a 3-coloring problem by jNEP . 87
4.4 FreeLing output for "Aquel chico es un gran ingeniero" (*That guy is a great engineer*) . . . 109
4.5 jNEP output for "Él es ingeniero". . . . 112
4.6 Shallow parsing tree for "Él es ingeniero" . . . 113

5.1 Derivation tree for the expression "l,sumb,+,Ab,*,Tib,Di," . . . 122
5.2 Empiric data against Evol. Model A . . . 126
5.3 Alonso et al., 2005 against Evol. Model A . . . 127
5.4 Empiric data against Evol. Model B . . . 128
5.5 del Rosal et al., 2005 against Evol. Model B . . . 129
5.6 Blocks of a general way to program natural computers . . . 131
5.7 Simplified scheme of the rotation NEP presented in Csuhaj-Varju et al. [2005] . . . 136

5.8 The Christiansen Grammar 138

Part I

Introduction

Chapter 1

Preamble

1.1 Recuerdos y agradecimientos

Han pasado ya muchos años desde que comencé mis estudios de doctorado. A lo largo de este tiempo he vivido diferentes cambios y vicisitudes, al mismo tiempo que conocía y trabajaba con muchas personas distintas. Agradecer el tiempo compartido a todas ellas sin que la memoria me juegue malas pasadas no va a ser fácil. Voy a intentarlo, si he olvidado a alguien, ruego me disculpe.

Es difícil de creer, pero todo esto comienza en un laboratorio de la facultad de psicología de la UAM. Allí trabajé un tiempo con mi primer director de tesis, José Santacreu, del que guardo un grato recuerdo, un profesor muy cercano y que, en la medida de sus posibilidades, hizo todo lo que pudo por ayudarme como doctorando. Aquel laboratorio era de lo más divertido y variopinto. Allí conocí al torbellino de Lola, a las aplicadísimas Montse y Ana, al bueno de Agustín, a Laura, a María y tantos otros que pasaban por allí de cuando en cuando, sin olvidar a nuestra rata CTRL-Z. Tuve la oportunidad de publicar algún artículo con Manuel, *el de la NASA*, Rafa y los ya mencionados Pepe y Lola.

En aquellos tiempos creo que era todavía bastante ingenuo, tenía una visión romántica e idealizada de la investigación. A lo largo de los años aprendí que un doctorando joven muy probablemente tenga que pasar por penurias económicas y todo tipo de desencantos durante su trabajo. Es por ello que, ya siendo formalmente ingeniero informático, me decidí a buscar otro lugar con mejores perspectivas profesionales y dar el salto a la Escuela Politécnica para realizar el doctorado allí. Me entristece pensar que dejé toda aquel mundo de la psicología y su gente, pues me encantaba. Fueron años ilusionantes y alegres.

El caso es que acabé recalando en un laboratorio de la Escuela Politécnica del que no recuerdo el número. Aquel laboratorio infinito estaba hasta los topes de gente. Allí conocí a muchos compañeros, es díficil enumerar a todos. Recuerdo especialmente a los gemelos Daniel y José Miguel, los magos de Linux, a Ignacio y su afición a las películas, a Ana, la sufrida única mujer, y a Javi Molina, divertido donde los haya. Un tiempo después me mudé de laboratorio a la planta dos, allí conocí a más gente, especialmente a Fer, un salmantino entrañable. Enfrente teníamos un laboratorio de "telecos" a cada cual más pirado, liderados por Iñaki, viejo amigo de la infancia.

Mientras tanto, echaba una mano a *los Manueles* para conseguir la solución definitiva a las copias en programación, problema importante en las prácticas de programación, donde el número de alumnos era enorme. Recuerdo con cariño el programa anti-copias: AC. También tuve la oportunidad de conocer a las dos alegres chicas rumanas que vinieron para quedarse: Alex y Cris. Los últimos años llegué a

vivir un tiempo en el campus, en una habituación de la residencia de estudiantes. Fueron los últimos momentos en los que pude dedicarme de forma casi plena a mi tesis, después vendrían diferentes trabajos fuera de la UAM y comencé a perder el contacto cotidiano con todo aquello. No puedo terminar de hablar de mis tiempos en la UAM, casi una vida entera, sin recordar a los amigos que hice durante mi tiempo de estudiante de grado. La mayoría de ellos, de una forma u otra también me acompañaron durante mi trabajo predoctoral. La gente de psicología, viejos amigos: Noe, Natalia, Abel, Alfredo y tantos otros. Y, también, aquella bendita locura de los físicos que fui conociendo, entre otros: Paula, Dani, Ana, Ángela, Gonzalo y mi tocayo Emilio.

Como ya he dicho, había que ganarse la vida y comencé a trabajar para, como se suele decir, pagar las facturas. Hice diferentes cosas, pero en relación a la tesis y el mundo académico, recuerdo con gran cariño el año que estuve trabajando en el CSIC en un proyecto de Ignacio Ahumada para la construcción de un corpus lingüístico del español como lengua científica. Aprendí una barbaridad de cosas con él y con Jordi, compañero y también informático. El resultado de todo aquello aparece en esta memoria. Trabajé en dos despachos diferentes. En el primero, recuerdo a mi única compañera: Marisol, encantadora, siempre pendiente de sus retoños. Luego recalé en el nuevo y enorme edificio del Centro de Ciencias Humanas y Sociales. Allí, pasaron por delante mía muchas personas en muy poco tiempo, pero con Virtudes pude entablar una relación de amistad que todavía perdura. Fue de agradecer su amabilidad al acogerme en ese extraño lugar, lleno de gente de humanidades mientras yo me comunicaba en unos y ceros. Todavía me admira la dedicación y entusiasmo que imprime a sus investigaciones de antropología.

Desde entonces, no le pude dedicar mucho tiempo a esta tesis que tienes entre manos. Más aun cuando comencé a trabajar como profesor en la universidad privada CEU San Pablo. Durante aquellos cuatro años aprendí una profesión, la de profesor, y maduré personal y profesionalmente. Por desgracia, todo aquello acabó con agrios encontronazos con mis jefes y el consiguiente despido. Y es que tuve la *extravagante* actitud de protestar al descubrir que mis jefes habían cambiado mis calificaciones a mis espaldas. Cosa que hicieron con la intención de otorgar aprobados espurios. Pero, también me quedé con el grato recuerdo de grandes compañeros entre los que me gustaría destacar a José Rojo, un hombre de integridad a prueba de bombas, y, también, a Gonzalo Cañadas y Marta Gómez junto a otros buenos compañeros que tuve allí.

Evidentemente, no voy a terminar esta pequeña carta de agradecimiento sin mencionar a todos los compañeros con los que he podido trabajar y publicar artículos, a todos ellos los homenageo de la mejor forma posible: citándolos a lo largo de la memoria. Por supuesto, entre ellos destacan Manuel Alfonseca, profesor de la UAM que comenzó co-dirigiendo mis primeros trabajos, y Alfonso Ortega, como no, el director de esta tesis. Alfonso siempre me ha acompañado en los altos y bajos de este arduo trabajo, ayudándome en todo lo que podía a lo largo de siete años.

Y a todos mis seres queridos que, sin pertenecer al mundo académico, han estado cerca todo este tiempo.

Gracias a todos.

1.2 Resumen

En la actualidad, se están realizando una gran cantidad de esfuerzos en el área de la Computación Natural. En general, sus objetivos principales son la definición, descripción formal, analisis, simulación y programación de nuevos modelos de cómputo inspirados en la naturaleza, habitualmente con el mismo poder computacional que la máquina de Turing. Este área también comprende algoritmos inspirados en procesos

naturales que, por sus características, son especialmente adecuados para abordar problemas relacionados con sistemas complejos o soluciones aproximadas.

Estos nuevos modelos despiertan múltiples intereses. Uno de los más importantes es que sus propiedades principales, como su carácter paralelo y distribuido, los hacen especialmente adecuados para la simulación de sistemas complejos. Además, su estudio nos puede llevar a concebir un nuevo paradigma de computadores; lo cual es especialmente importante en estos momentos en los que la clásica arquitectura de von Neumann, y sus implementaciones actuales sobre tecnologías basadas en silicio, está alcanzado sus límites teóricos. Por último, su naturaleza masivamente paralela les permite tratar problemas NP de manera eficiente.

Sin embargo, su uso presenta también algunas desventajas. La mayoría de ellas relacionadas con el hecho de que estos modelos se componen de gran cantidad de elementos actuando de manera coordinada, generando comportamientos muy complejos globalmente, pero partiendo de acciones locales muy simples de cada elemento. Esta particularidad provoca que las bases de su funcionamiento sean difíciles de entender. Por la misma razón, el diseño y la programación de este tipo de dispositivos suele enfrentarse a grandes dificultades.

A lo largo del presente trabajo, hemos intentado contribuir a esta área de investigación principalmente de tres maneras. Hemos desarrollado y estudiado el alcance de un entorno de trabajo para la simulación y programación de Networks of Evolutionary Processors (NEPs) [Castellanos et al., 2001], un novedoso modelo de computación que es masivamente paralelo y está inspirado en ciertos mecanismos biológicos. También hemos investigado el uso de NEPs en el campo del procesamiento de lenguaje natural y para la resolución de problemas NP. Otra de nuestras aportaciones ha sido una metodología general que permita el diseño o programación automática de sistemas complejos como los NEPs. Tal metodología nos parece de gran interés dada la complejidad de diseñar o programar este tipo de sistemas. Para ello, nos hemos apoyado en el algoritmo de Grammatical Evolution (GE) y sus variantes más avanzadas como Christiansen Grammar Evolution or Attribute Grammar Evolution [Echeandia et al., 2005], todos ellos algoritmos de programación automática evolutiva. Estos algoritmos son también parte del área de Computación Natural, puesto que se inspiran en la evolución de las especies por selección natural.

En cuanto a la estructura y contenido, el presente documento consta de dos partes. La primera contiene el presente preámbulo cuyo contenido es una introducción general a la tesis doctoral, tanto en español como en inglés. Además, en su segundo capítulo, ofrece una introducción al área de la Computación Natural en términos generales (sección 2.1) y, tras esto, presenta una explicación detalla de nuestras dos herramientas de trabajo principales: Networks of Evolutionary Processors (NEPs) y Grammatical Evolution junto a sus variantes y mejoras más importantes (sección 2.2). Por último, se presenta una descripción detallada de los dos problemas principales que se abordan en la tesis: el modelado del aprendizaje asociativo animal y el procesamiento del lenguaje natural (sección 2.4).

La segunda parte del documento contiene los principales desarrollos y avances realizados durante nuestro trabajo. Primeramente, en el Capítulo 3, mostramos el amplio entorno de trabajo para simular y programar NEPs que nosotros mismos hemos desarrollado. Éste incluye *jNEP* y sus módulos y añadidos. En segundo lugar, el Capítulo 4 presenta nuestros trabajos sobre simulación de problemas NP con el entorno de trabajo ya señalado. En él se muestran también las posibilidades de los NEPs para resolver problemas NP de manera eficiente. Seguidamente, se detalla el uso de los NEPs para el análisis sintáctico y se discute en detalle nuestros algoritmos de análisis sintáctico para NEPs y sus ventajas en el caso de análisis de lenguaje natural. Por último, el Capítulo 5 versa sobre nuestros trabajos relacionados con la aplicación de los algoritmos de Grammatical Evolution para el modelado

automático de sistemas complejos presentes en la naturaleza. En concreto, se explica su aplicación en el caso de la programación automática de NEPs y el modelado automático de procesos de aprendizaje asociativo animal. Para cerrar el capítulo, discutimos un borrador de una posible metodología general de modelado automático inspirada en nuestra experiencia a los largo de los diferentes trabajos.

Finalmente, el Capítulo 6 contiene unas breves conclusiones globales, junto con comentarios finales y líneas de investigación futuras. El documento termina, como es de esperar, con sus correspondientes apéndices y la bibliografía utilizada.

Todo el trabajo aquí presentado deriva en mayor o menor medida de las publicaciones listadas en la tabla 1.1. Ellas representan el trabajo realizado por el autor a lo largo de sus años de doctorado.

1.3 Context and motivation

A great deal of research effort is currently being made in the realm of *natural computing*. Natural computing mainly focuses on the definition, formal description, analysis, simulation and programming of new models of computation (usually with the same expressive power as Turing Machines) inspired by Nature. It also concerns algorithms inspired by natural processes, which are especially accurate for problems dealing with complex systems or approximate solutions.

These new models have different interests. Firstly, their main features, like intrinsic parallelism/distributivity, makes them particularly suitable for the simulation of complex systems. Secondly, they could lead to a new paradigm of computers, which is particularly important, as the von Neumann architecture and its conventional implementation with silicon-based technologies is reaching its theoretical limits. Last but not least, their parallel nature makes them capable of treating *NP* problems efficiently.

However, they also have some counterparts. Most of them consist of many elements behaving in a coordinated fashion, creating a global complex behaviour from very simple local decisions. For this reason, the fundamentals of their functioning are often difficult to understand. In the same manner, the task of designing or programming these kinds of devices faces many difficulties.

During our work, we have tried to contribute to this field of research by developing and studying a framework for the simulation and programming of a bio-inspired computing model called Networks of Evolutionary Processors (NEPs) [Castellanos et al., 2001]. We have also investigated its application to NP problems and to the field of Natural Language Processing. In addition, since designing and programming NEPs and other natural systems is a complex task, we have proposed and studied a general methodology that permits the automatic design or programming of complex systems. This methodology is based on the Grammatical Evolution (GE) algorithm and its modern variants like Christiansen Grammar Evolution or Attribute Grammar Evolution [Echeandia et al., 2005]. GE is an algorithm inspired by the natural process of evolution and natural selection.

1.4 Brief description of the contents

The present dissertation is divided into two parts. The first one contains this preamble with a general introduction to the document (chapter 1). Later on, it introduces the field of natural computing in a general way (section 2.1). Those readers familiar with natural computing can skip this section. After that, it explains in detail our two main objects of research: Networks of Evolutionary Processors and Grammatical Evolutions with its improvements (section 2.2). Finally, it describes

the problems we are dealing with and their state of the art: modelling animal associative learning and language processing (section 2.4).

The second part presents the main developments and advances achieved during our work. Firstly, Chapter 3 shows our wide framework to simulate and program NEPs, which includes the simulator *jNEP* and other extensions. Secondly, Chapter 4 presents the simulation of NP problems with the aforementioned simulator and the power of NEPs to efficiently solve these kind of problems. Later on, it explains the application of NEPs to language parsing. In this section, we discuss in detail our NEP parsing algorithm and its suitability for natural language parsing. Finally, Chapter 5 shows how Grammatical Evolution can be used to automatically model or program natural systems, especially NEPs and process of animal associative learning. At the end, we outlined a possible general methodology for automatic modelling built from our experience.

Lastly, Chapter 6 contains the conclusions and final comments of the present work together with some future research lines. It is followed by a few appendices which present a detailed description, in terms of their jNEP configuration file, of some complex NEPs discussed along the document. An extend bibliography is listed at the end of the document.

It is worth noticing that most of the content is directly derived from the following publications. They represent in some way the work of the author throughout his pre-doctoral studies.

Publications

Chapter in book		
2	Ortega et al. [2011]	Sections 4.2, 4.3 and 4.4.
1	del Rosal et al. [2009a]	Section 3.1.
Article in journal or LNCS proceeding		
8	Ortega de la Puente et al. [2012]	Sections 3.1, 3.1.3, 3.1.4 and 3.1.5.
7	del Rosal et al. [2011]	Section 5.2.
6	Porta et al. [2011]	Sections 4.3 and 4.4.
5	del Rosal and Cuéllar [2009]	Section 3.1.3.
4	Ortega et al. [2009]	Section 4.2.
3	del Rosal et al. [2008]	Section 3.1.
2	del Rosal et al. [2006]	Sections 2.4.1 and 5.1.
1	Alonso et al. [2005a]	Sections 2.4.1 and 5.1.
Conference proceeding		
6	del Rosal et al. [2012]	Section 5.2.
5	de la Cruz et al. [2011]	Section 3.1.5.
4	Jimenez et al. [2010]	Section 3.1.4.
3	del Rosal et al. [2010]	Section 4.4.
2	del Rosal et al. [2009b]	Section 4.1.
1	del Rosal et al. [2007]	Section 5.1.

Table 1.1: Publications sorted by date and type

Chapter 2

State of the art

2.1 Introduction to natural computing

Bio-inspired computing or natural computing can be defined as the field of research that takes inspiration from nature to develop new computing tools, algorithms or paradigms. This quite new field is very diverse and has proposed a vast amount of ideas during the last 20 years. Along this section, we will try to give a general introduction to the field, emphasizing aspects and research lines related to our work. Nevertheless, this introduction is intended to give the reader a general insight on natural computing, focusing on the interesting aspects common to most natural computing research. We think this introduction is important for clarifying the context of our contributions and deeply understand the implications of our work. However, those readers with some knowledge on natural computing can skip this section and continue reading the next sections, where we will present the two main components of our work in detail; Grammatical Evolution and Networks of Evolutionary Processors.

Although it is difficult to find a fixed set of features that every natural computing integrant has, some general aspects exist that most of them share. At least, there are some general concepts present in most natural computing ideas, following de Castro [2006], we will try to describe the most relevant below.

They are composed of individuals or agents. Many natural computing systems are made up of somehow independent/autonomous elements. They are situated within an environment and are also part of it. Actually, they can only interact with each other through the environment, thanks to their capability to perceive it and interact with it. The resulting behaviour of these agents interacting with each other through the environment constitutes the whole system. Moreover, the simple behaviour of each agent interacting with its partners creates synergies that create complex, intrincate and rich behaviour in the whole system.

Parallelism and distributivity. Parallelism and ditributivity have to do with being capable of processing various things at the same time. There are different examples of parallel processing in nature, such as ant colonies or the human nervous system (neural networks). The former system contains different individuals (ants) and each of them is responsible for one task: harvesting, cleaning, etc. They all work in parallel to carry out a general task: the colony's survival and reproduction. In the same manner, a large number of neurons form the nervous system. Each neuron receives and sends stimuli to other neurons in a deeply interconnected network. Therefore, many different computations are performed in the nervous system all at

once. In a broad sense, we can easily observe parallel behaviour carried out by the brain when we listen to the radio while running or when we talk and drive at the same time.

As mentioned before, every unit or element with assigned tasks in a parallel/distributed system is called an individual or an agent. The idea of a set of agents interacting easily leads to the ideas of parallelism and distributivity. Each agent carries out a different computation and, after that, shares or communicates its output to other agents. Those new outputs serve as new inputs and so on. This way of behaviour is inherent to many natural computing systems and implies parallel and distributed manners of solving the global computation.

Interactivity. Obviously, those individuals forming the system need to communicate and interact with each other, otherwise the set of individuals would not form a system. The agents' interaction can function at different levels, from a direct one-to-one interaction to a more complex indirect interaction. We could also see some interactions with the environment as an independent entity. For example, a neuron interacts with other neurons by sending to it excitatory or inhibitory signals. Insects can also interact with one another by placing pheromones around particular places, attracting other insects.

Adaptation. One of the most important features of natural systems is the ability to fit or adjust their behaviour to the environment's stimuli. Dynamically, as the environment changes, natural systems can gradually change in order to adapt themselves to the environment in a better way. A classic example is the change in the connections between neurons. An important learning rule says that those neurons that fire together tend to fire together in the future. This way the nervous system can change the neural connections and produce adaptations, as the environment's stimuli cause particular neural activation patterns. These kinds of changes are called "learning". However, the adaptation based on evolutionary biology is usually called "evolution" and is based on letting the strongest live and the weakest die. Mainly, those individuals which fit better in the environment or, in other words, produce better-quality outputs, prevail and produce the next offspring. The new generations contain slightly different individuals (by mutation or other evolutionary processes) and, again, some of them will prevail and others will die. This way, evolution is said to *search* for better individuals.

Feedback loops. They consist in a self-control process by which the system tends to repeat or inhibit a response as the response itself appears. Therefore, the loop can be positive or negative. The former reinforces the response, while the latter inhibits it. A positive feedback example is the population growth in any species: the more individuals reproduce, the bigger the population to reproduce is. On the other hand, a negative feedback appears when a large population consumes many resources and the lack of resources makes successful reproduction more difficult. Positive and negative loops usually regulate the system until an equilibrium is reached.

Self-organization. Many natural systems present amazing patterns and internal organization. The most interesting feature of this organization is that it seems to originate, somehow, from within each element of the system spontaneously, with no external instructions or global rules. In other words, global organization comes from a large number of individuals interacting or rises from local, independent decisions of the individuals. For example, the beehive's regular shapes are not created thanks to the guidance of a leader or external rules but to the agent's interaction and simple

individual behaviour. The opposite kind of behaviour is presented, for example, in a group of soldiers marching, since its behaviour is induced by global control.

Complexity and emergence. Complexity and emergence are very abstract concepts and difficult to define. However, both are involved in the previously defined concepts. We could define a complex system as a large amount of independent elements interacting and self-organized, whose global behaviour is not just the simple sum of its elements. We call emergence to the arising of that new behaviour which is not just a simple or intuitive addition of the elements. This emergence makes natural systems rich and interesting, however it also makes them difficult to understand. The behaviour of ant colonies are a good example of emergence patterns. A single ant follows very simple, local rules, however a group of ants interacting through those simple rules shows a much more complex and rich global behaviour.

With all these general concepts in mind, the natural computing field has presented many new ideas, tools, models and even paradigms. They are all diverse and sometimes difficult to categorize. In the following sections, we will try to divide them into three main branches following the proposal of the review made in de Castro [2006]. These branches represent the final objective of the natural inspiration when contributing to computer science. The three main areas of contribution are algorithms, computing models and new hardware paradigms. Nevertheless, it is worth noticing that these categories overlap to some extent in many cases, since some natural computing ideas can be studied, for example, from an algorithmic perspective and from a computing model one at the same time.

We can not offer a complete and detailed description of the vast natural computing field, since it would be outside the scope of this work and our possibilities. However, we will try to make a sufficient presentation, emphasizing those key ideas and tools relevant to our work. From now on, we assume that the reader has a basic knowledge on automata theory and formal languages [Martin, 2003].

2.1.1 Algorithms inspired by nature

In the following sections, we will present a variety of algorithms inspired by nature. Different research lines have observed the ways nature solves problems and have found new ideas to design innovative algorithmic methods.

2.1.1.1 Evolutionary computing

Evolutionary computing is a family of algorithms inspired by the natural selection of species, firstly described by Darwin's theory of evolution. As it is well-known, the process of evolution consists of the survival of the strongest individuals and the death of the weakest ones. The strongest individuals have more probability to survive and, thus, reproduce more often. Furthermore, the new offspring is not an exact copy of the progenitors, but it has small random variations which could permit better adaptation or fulfill future environmental needs. This way, the population of a given species evolves throughout time, producing individuals better adapted to the environment.

Taking this natural process as inspiration, a general evolutionary system has the following main features:

Population A set of individuals with a significant degree of diversity in their behaviour and features.

Environment The entity where the individuals are placed. The individuals interact with the environment and have to fulfill its demands so as to improve their probability of survival.

Reproduction process All the individuals die soon or later. Thus, a reproduction process has to take place periodically to maintain the population.

Source of variation Reproduction produces new individuals which are similar to their parents but not an exact copy. This source of variation comes from a variety of operations which have different features. The system tries to find better individuals by these small modifications.

With these four components in mind, the dynamic of the system consists of a constant renovation of the population, whereby the individuals increase the quality of their behaviour as the system varies the descendants progressively. The quality is measured in terms of its fitness to the environment. From a computational perspective, we could understand that the system is *searching* for the best quality individual. In other words, given a problem (fitting the environment), the system searches for an individual good enough by means of an iterative stochastic process.

This metaphor inspired evolutionary algorithms. In this field, any problem solving task is understood as a search problem that can be solved by evolutionary means. The search takes place in a solutions space where each point represents a solution/individual. The system explores the space following a path that, hopefully, leads to the best solution. A detailed formal characterization of an evolutionary search can be found in Eiben and Smith [2003], in this brief introduction we will try to explain how evolutionary algorithms function. The previous perspective makes evolutionary algorithms a versatile tool, since almost any problem can fit this scheme. We will understand this assertion in a deeper manner once we present the components an evolutionary algorithm should have:

Representation of individuals In an evolutionary algorithm, each individual is a candidate solution to the problem. In this manner, individuals have to be defined in a formal way. Since in most cases a solution to the problem is a complex expression in a formal language, a simpler encoding, called *genotype*, is designed for the solutions. The genotype is the representation of the individual, while the formal expression of the solution is called *phenotype*. The translation process from genotype to phenotype is called *mapping*.

Fitness function To measure the quality of individuals, that is, the quality of the solutions found, we need to define a function to evaluate them. It represents the requirements the solutions have to adapt to.

Population It is the set of candidate solutions. It changes every reproduction cycle where some individuals die and others are born.

Parent selection mechanism Before reproduction occurs, the parents of the new offspring have to be chosen. Mostly, the best adapted individuals (those individuals with higher fitness values) are chosen to reproduce, although other criteria can be used.

Variation operators As expected, the new individuals are similar to their progenitors, but some differences also appear. There are different sources of variation that make the offspring diverge slightly. They are mainly an analogy of natural evolution phenomena, as mutation or chromosomes crossover. In short, they all alter little pieces of the progenitors representation to create a new individual.

Figure 2.1: General scheme of an evolutionary algorithm. Based on Eiben and Smith [2003]

```
BEGIN
  INITIALISE 'population' with random candidate solutions;
  EVALUATE each candidate;
  REPEAT UNTIL ('termination condition' is satisfied)
    (1) SELECT parents;
    (2) REPRODUCE;
    (3) EVALUATE new candidates;
    (4) SELECT survivors;
  END
END
```

Survivor selection mechanism As in the case of parent selection, some evolutionary algorithms make a selection of the individuals to survive which, therefore, are eligible for reproduction. Again, higher fitness values have a larger probability of surviving.

Taking the previous elements, a general scheme of an evolutionary algorithm can be described. It consists of a loop where selection and reproduction take place many times until a stopping condition is satisfied. Usually, the termination condition is finding a fitness value higher than a predetermined threshold. In a more formal way, the algorithm in pseudocode is presented in Fig. 2.1:

To make a better explanation, we will give a complete example an of evolutionary algorithm. For that purpose, we will use the most well-known type of evolutionary algorithms: genetic algorithms. Genetic algorithms mostly use a string of integers or binary digits as representation, although there exists other variants [Eiben and Smith, 2003]. These strings are usually called chromosomes, as an analogy to natural genetics. In our example, we will design a genetic algorithm to solve the Eight-Queens problem: we are given a regular chessboard and we have to place eight queens in such a way that no queen can check any other.

We could imagine many different representations for a candidate solution to this problem. We will choose a string of eight integers in the range [1,64], where each integer represents the position of its corresponding queen on the chessboard (remember a chessboard has $8 \times 8 = 64$ positions). The fitness function can be defined as the number of possible checks in the board. Concerning reproduction, our algorithm will create two new children from two individuals in the following way: firstly, a splitting point is chosen randomly and, after that, the two parents are cut in two segments. Finally, segment 1 of parent 1 is merged with segment 2 of parent 2 and vice versa. This way of child creation is a very frequent variation operator which is called chromosome recombination or crossover, again as an analogy to natural genetics. After crossover, a new variation operation is executed which consists of changing one of the integers randomly. Since one can think of an integer as a gene of a chromosome, this kind of variation operation is called mutation. At this point, we only need to decide the selection mechanisms. There are plenty of alternatives, in our example we will just take the best 2 out of 5 random individuals each time we want to create two children and later on the two worst individuals of the population die. Thus, we can summarize our algorithm's parameters in table 2.1. They are just a proposal, any evolutionary algorithm needs a parameter tuning to work properly. In this case, not much work is needed to find the most efficient parameters, since the problem is easily solved by a genetic algorithm as the one presented or any other

Table 2.1: Eight-Queens genetic algorithm general description

Representation	String of eight integers
Fitness function	Number of possible checks
Variation operators	Mutation and one point crossover
Mutation probability	90%
Parent selection	Best 2 out of 5 random individuals
Survival selection	Replace worst
Population size	500
Offspring size	2
Initialization size	Random
Stopping condition	Fitness = 0 or 10000 cycles

similar.

During the present introduction, we have tried to give a clear explanation and basic insight into evolutionary algorithms. Nevertheless, there are many variants and details in all the components of evolutionary algorithms that we have omitted. For further details and a deeper explanation we recommend [Eiben and Smith, 2003]. It is worth noticing that we will describe in detail an evolutionary algorithm which is important for our work in section 2.2. Indeed, genetic algorithms are the most widely used and known algorithms of the family, however, there are other types of evolutionary algorithms. Below, we will briefly present the most important ones.

Evolution strategies Mainly, evolution strategies are used for continuous parameter optimization. Therefore, the construction of a fitness function is quite straightforward. A $f : \Re^2 \rightarrow \Re$ function where its domain represents the parameters to tune and its codomain the fitness values. Most times, the fitness function is the problem at hand itself, where the target is minimizing or maximizing the aforesaid function. In this context, choosing the representation is also obvious. In fact, the simplest representation is a real-valued vector of n elements $\bar{x} = (x_1, ..., x_n)$ which stands for the parameters to search for.

Moreover, the most important specialty of evolution strategies is the self-adaptation of the evolutionary parameters as the algorithm runs, mainly, mutation's parameters. In short, every x_i variable has a mutation parameter associated σ_i. Mutation operations depend on their σ parameter by a mechanism we will explain later. Furthermore, σ parameters also adapt or evolve as the rest of the individual. For this reason, it is said that evolutionary strategies perform self-adaptation. These kinds of evolutionary parameters are called strategy parameters. Putting these two components together, we obtain the following individual structure:

$$\langle \bar{x}, \bar{\sigma} \rangle = \langle x_1, ..., x_n, \sigma_1, ..., \sigma_n \rangle$$

Individuals can have an even more complex structure where strategy parameters can interact with each other. This interaction depends on a third type of component which is denoted as α_i and represents the degree of correlation between each pair of σ_i. These kinds of individuals permit correlated mutations. This way, two parameters highly correlated tend to mutate in the same direction together (increasing or decreasing) pointing to the fitness maximum, which can speed up the evolutionary search. More details about this more complex mutation system is in Eiben and Smith [2003]. Thus, the most general case of individual has the following structure.

$$\langle \bar{x}, \bar{\sigma}, \bar{\alpha} \rangle = \langle x_1, ..., x_n, \sigma_1, ..., \sigma_n, \alpha_1, ..., \alpha_{n(n-1)/2} \rangle$$

Mutation in evolution strategies Mutation is the main variation operation in this type of evolutionary algorithm. As mentioned, an individual is a real-valued vector, thus a mutation consists in changing one of the floating-point values. This change is determined by a random addition to the original value as follows: $x_i^{'} = x_i + N(0, \sigma)$, where $N(0, \sigma)$ stands for a normal distribution with zero mean and standard deviation σ. Therefore, the mutation is said to have a step size which depends on the σ strategy parameter. Self-adaptation appears when the σ parameter also can mutate. Mutation in that case also follows a probability distribution. We will omit more details, since a complete explanation of evolution strategies' mutation scheme is outside the scope of this brief explanation.

Recombination in evolution strategies Since the genes are floating-point values in this case, sometimes a special crossover scheme is used in which the two parents' (x, y) genes are averaged to produce the new child. More formally, for each value i in the vector, the child value is computed in the following manner: $z_i = (x_i + y_i)/2$. This special crossover is called intermediate recombination.

Selection in evolution strategies In this case, parent selection is not fitness biased as usual. Whenever a new child has to be created two parents are chosen randomly. However, survivor selection is very severe with bad quality individuals; only the n best individuals survive. The set considered to select survivors is the offspring only, denoted (μ, λ) selection, or the offspring together with the parents, denoted $(\mu + \lambda)$, where μ stands for the parents and λ for the offspring.

Evolutionary programming Originally, they were developed as a designing tool for artificial intelligent [Fogel et al., 1966]. They regarded AI as a field that tries to create agents that perceive the environment and respond to it so as to achieve a particular goal.

In this context, the algorithm was conceived to treat with finite state machines and other similar machines. The main features of evolutionary programming appeared to fulfill the needs of evolving these kind of machines: 1) representation is flexible, 2) crossover is not used for such representations and 3) mutation, in many different manners, is the main variation operator. The most important mutation operators were:

- Changing an output symbol in a state transition.
- Changing state transition, precisely changing the next state.
- Adding a new state to the machine.
- Deleting an existing state.
- Changing the initial state.

However, the evolutionary programming family developed and changed to a quite standard algorithm that is very similar to evolution strategies. This variant also uses real-valued vectors as the main representation and progressively incorporated self-adapted σ strategy parameters. Therefore, a general widely used form of individuals is as follows:

$$\langle \bar{x}, \bar{\sigma} \rangle = \langle x_1, ..., x_n, \sigma_1, ..., \sigma_n \rangle$$

which is equivalent to those of some evolution strategies. In this case, mutation works in a similar way as in evolution strategies.

Nevertheless, parent and survivor selection differs from evolution strategies. Parent selection is almost not used, since every individual in the population creates a new one during reproduction. Note that this is possible thanks to the lack of crossover operations. It also implies that fitness does not play any role in parent selection which is a special feature of evolutionary programming. On the other hand, survivor selection is fitness biased due to a mechanism called tournament selection which grades every member of the population and the offspring in the following way: for an individual i, n competitors are chosen randomly and the fitness of i is compared to each competitor's, finally the number of times i wins is the actual grade. Those individuals with the greatest grades are selected.

Genetic programming It was developed as an automatic programming tool for modern computer languages. At its first stage, the representation were programs in the LISP language that tried to solve a particular task. Given the special representation, specific variation operators were created and the fitness function was a measure of the program requirements. Later on, many variants and improvements have appeared which use other languages. We will discuss this algorithm family in detail later in section 2.2, since it is the precursor of Grammatical Evolution which is a central tool in this work.

As shown above, evolutionary computing is a big family of algorithms which share the same searching scheme and their inspiration in the biological process of evolution.

2.1.1.2 Neurocomputing

Neurocomputing is the oldest research area of those presented in this introduction. The important paper of McCulloch and Pitts [1943] represents the start of the area. It describes an explanation for the nervous activity based on simple computational units which model the biological neurons and their connections. Those neurons, though simple, present the most important features of biological neurons. Each unit has an activation state which represents if the neuron is firing or not. Their functioning is similar to binary logical operators, since the activation state of the unit can only be “1” or “0”. Furthermore, a neuron receives input from other neurons' activation state and these inputs determine its activation. A neuron changes its state to “1” if the sum of its inputs is greater or equal to its threshold value. For example, if a neuron has two inputs and a threshold of “1” it will behave as the logical operator OR. If the threshold is two, we have got the AND logical operator. Neurons are connected in a network and the complex interaction through their links carries out the computation. Some neurons' activation state is considered the output of the network. Although this model is old and in some ways obsolete, it has had a great influence and still has the main features of most artificial neural networks.

Artificial neural networks Following de Castro [2006], artificial neural networks (ANN) shares the following features with the nervous systems. These features are also a good list of common elements to all variants of ANN:

- The basic computation process is made by simple artificial neurons.
- Neurons are connected to other neurons, they receive and send stimuli from/to each other.

- These stimuli are sent via links called synapses. They are varied in their principles and functioning.
- Each synapse has a strength or weight assigned. It determines the efficiency of the link, in other words, how strong the influence of the link is in the activation of the receiving neuron.
- Knowledge is implemented in the synapse's' strength. Therefore, learning consists in changing the synapses' strength.

A general artificial neuron can be defined formally as a set of weight values, a summing junction and an activation function. The weight values represent the strength of each input to the neuron. The stronger the connection, the bigger the influence of that input in the neuron activation. All those inputs are added by the summing function. It determines the final input value received by the neuron. Finally, the final input is used by the activation function which calculates the final output of the neuron or, in other words, its activation value. Usually, the activation value is limited to a range.

Learning methods As any other computational device, a neural network receives an input and tries to send the right output. In this case, the input is a vector of activation values sent to a group of so-called input neurons. In the same way, the output is the vector of activation values of output neurons. The rest of the neurons should compute its activation values in a proper way so as to, at the end, produce the right output. Then, the objective of any ANN is producing the correct output vector for each corresponding input vector.

In a neural network, the only way to define the computation to be done is by assigning the weight values of each connection. Mostly, the task of *programming* this kind of devices is not hand-made, but developed by automatic learning methods. This is one of the main interests of ANN. There are two main approaches to ANN learning: supervised and unsupervised learning.

Supervised learning In this case, learning occurs during a big amount of consecutive cycles consisting of: 1) presenting the input to the network, 2) observing the output, 3) comparing the given output with the desired output and 4) changing the weights so as to produce an output slightly closer to the desired one. The most famous ANN learning algorithm belongs to this category and is called *backpropagation*[de Castro, 2006, Bechtel and Abrahamsen, 2002].

Unsupervised learning Also called self-organized learning, it has no desired output and, thus, there is no input-output adaptation. These kind of algorithms is focused on detecting regularities in the input patterns. As the main principle, it follows the idea of competitive learning. In short, each neuron competes for being the one activated when an input is presented. Once a neuron wins a competition, its weights are modified to be more strongly influenced by the corresponding input pattern. This way, a positive feedback loop [1] comes into play which makes the winner neuron closely related to the input. Usually, neighboring neurons are also positively influenced by this relation. This way, each neuron or group of neurons becomes a detector of a particular input or group of inputs that share features. Thus, unsupervised learning techniques are able to find clusters or categories in input patterns. The widely used self-organizing maps [Kohonen, 1990] belong to this kind of learning.

[1] In the sense explained in 2.1

There exists plenty of mathematical methods to perform both kinds of learning. Note that we have also ignored other less important sorts of learning. We will not give a deeper explanation of them in this brief introduction. We refer to Haykin [2009].

Artificial neural networks can be considered a complete computational model. In other words, they are able to solve any computable function. Nevertheless, they are suited for clustering, classification and function approximation problems. They are a good alternative for those problems where hard knowledge-based rules can not be applied. They have been widely used in many industrial applications like patter recognition, robotics, data mining, etc. For a deeper theoretical and technical introduction we refer to Haykin [2009]. For a broad discussion on the interplay between ANN, biology and psychology we recommend Bechtel and Abrahamsen [2002].

Other algorithms inspired by nature There are two more broad families of algorithms inspired by nature: *swarm intelligence* and *immunocomputing*. The former was inspired by the behaviour of ant colonies, bird flocking and fish schooling, on the other hand, the latter was inspired by the functioning and structure of the human immune system.

The most famous algorithm of the swarm intelligence is the Ant Colony Optimization algorithm (ACO). It follows the principles of the collective foraging performed by ant colonies. Ants create pheromone trails from the nest to food sources. The trail is created when an ant finds a food source by accident and starts leasing pheromones at the source and the path to the nest. Pheromones attract more ants to the source, reinforcing the trail. This behaviour has interesting features as coming from independent local decisions and surprisingly finding the shortest path to the source. ACO is a mathematical formalization of this collective behaviour. It can easily be applied to a classical NP problem like the traveling salesman problem, although its application can be generalized to any discrete combinatorial optimization problem [de Castro, 2006].

The second main Swarm Intelligence branch is `particle swarm`. It is based on the idea of social adaptation of knowledge and, in the same manner, on bird flocking and fish schooling. Any particle swarm algorithm has a set of individuals conforming a neigbourhood and they try to adapt to the environment. This learning/adaptation relies on three main principles: 1) individuals realize their degree of adaptation to the environment, 2) individuals are able to compare their adaptation with their neighbours' and 3) individuals imitate others behaviour when it seems more appropriate. This way, researches have implemented this kind of behaviour by representing individuals as parameter vectors and describing the improvement of an individual at each step $x_i(t)$ with the formula:

$$x_i(t+1) = x_i(t) + \Delta x_i(t+1),$$

where $\Delta x_i(t+1)$ is composed of the knowledge observed in the best neighbours and the individual's best values. There are many variants of this general scheme [de Castro, 2006] and they are mostly used for parameter optimization, as in the case of evolution strategies. This is not surprising if we note the similarities with the evolution strategies' individuals representation.

Concerning immunocomputing, the human immune system is extremely complex. Many organs and blood cells take part in the system functioning. Computer scientists have created many different algorithms inspired by this reach system. It is the youngest research area in bio-inspired computing [Timmis and Bentley, 2002]. We will try to give just a brief insight into the two most important families of algorithms.

`Negative selection algorithms` is a search algorithm that tries to find elements different to a familiar pattern. Note that this target is the opposite to other algorithms already commented. This idea is inspired by immune T-cells whose main objective is to find pathogens or, in other words, molecules external to the human body. This negative or anomaly detection procedure fits the needs of some problems like computer *network intrusion detection* or cancer diagnosis [de Castro, 2006]. The algorithm follows a mechanism similar to the one in the thymus, where T-cells are selected according to their ability to detect external elements. The algorithm initializes a large amount of patterns randomly which are compared to the self patterns. Those patterns similar to the self patterns are discarded and those which are different prevail. After that, the set of surviving patterns, also called detectors, can be used to monitor changes or track novelties in the system.

`Clonal selection` is a immunocomputing algorithm similar to genetic algorithms. However, in this case, individuals are antibodies of the immune system and they must be able to fit the antigens of the pathogens. In biology, both antibodies and antigens are proteins that fit together, this way, the immune system can locate and catch the pathogens. Thus, under this metaphor computational antibodies must evolve to fit the antigens as much as possible. Clonal selection algorithm proposes a procedure to evolve the antibodies/individuals [Forrest et al., 1993].It can be summarized as follows:

1. Create a random population of antibodies.
2. Evaluate the affinity of antibodies to the antigens.
3. Evolve the antibodies using a sort of genetic algorithm.

In most cases, clonal selection only uses mutation variation operators and reproduction rates of antibodies which are proportional to their affinities. This way, the algorithm is slightly different in concept to regular genetic algorithms. Finally, not surprisingly, clonal selection has been used for patter recognition with success.

During the last paragraphs, we have presented a brief overview of bio-inspired algorithms. Needless to say, we have omitted many variants and families of algorithms. However, we hope the reader has received a good insight on the main features, applications and nature of these kinds of algorithms. Along the section, we have seen examples of the main features enumerated at the beginning: parallelism and ditributivity, interactivity, adaptation, feedback loops, self-organization, complexity and emergence. All these features make bio-inspired algorithms different in methods and potentials to those classical algorithms developed under the von Neumann architecture's mindset. They are specially suited for problems that are difficult to solve with hard knowledge-based rules or do not have a clear analytical solution. Furthermore, they are a good alternative when dealing with complex systems or a hard temporal complexity (NP problems).

2.1.2 Computing models inspired by nature

The so-called bio-inspired devices or natural computing devices are formal complex systems that are able to compute and could, therefore, be used as computers. All of them share two main characteristics: their inspiration in the way in which Nature efficiently solves complex tasks and an intrinsic parallelism that makes it possible to develop algorithms which improve the temporal performance of classic von Neumann architectures.

In this section, we present the most relevant bio-inspired devices. Some of them were conceived for simulating a specific natural phenomenon, others are general

Iteration	Word
0	c
1	b
2	ac
3	cb
4	bac
5	accb
6	cbbac
7	bacaccb
8	accbcbbac

Table 2.2: Output of the L-system at each iteration.

abstract devices inspired by how nature works. However, all of them are promising models for a future implementation of highly parallel devices as we will discuss in section 2.1.3. Note that, at this point, we will ignore Networks of Evolutionary Processors which are the most important model for our work. Later on, in section 2.3 a detailed introduction will be given.

2.1.2.1 L-systems

Lindenmayer systems [Lindenmayer, 1968], or L-systems in sort, are a mathematical formalism firstly developed to simulate the growth of multicellular organisms, especially plants. They have been broadly studied due to their relation to formal languages and automata theory. In fact, they can be considered a special type of formal grammar and share many aspects with Chomsky grammars.

Informally, an example L-system taken from de Castro [2006] consists of an alphabet, $A = \{a, b, c\}$, and a set of production rules:

- $a \rightarrow c$
- $b \rightarrow ac$
- $c \rightarrow b$

Note that there is no distinction between terminals and non-terminals. Furthermore, a rule's right side is composed by only one symbol and every symbol in the alphabet must have at least one rule where it appears on the right side. The last element we need is the axiom which is a non-empty word, in our example $x = c$. If there is only one rule per symbol, the L-system is called deterministic. In table 2.2, we present an example of the re-writing process in this kind of grammar. It starts at the axiom and applies every possible rule to every symbol presented in an iterative process.

Even though it is difficult to imagine at first glance, L-systems has been widely used to model fractals. For those not familiar with fractals, we could define a fractal as a geometric shape that is, somehow, self-contained or, in other words, presents self-similarity. Put differently, it is a shape where if one zoom in it will find the same global figure again and again. Fractals are a very intuitive idea when giving a graphical example; see Fig. 2.2. L-systems can model initiator-generator fractals, we will focus in this kind of fractal during this section.

A deep understanding of fractal mathematics is outside the scope of this brief introduction to L-systems' applications. The interested reader is referred to Mandelbrot [1983]. In short, some of the typical properties of fractals are self-similarity,

Figure 2.2: The construction of the classical Sierpinski fractal. The actual fractal is the result of an infinite number of iterations. The image was taken from *Wikimedia Commons* and is in the public domain.

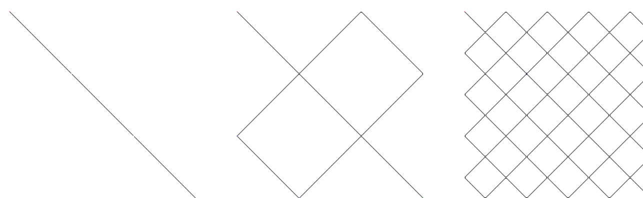


Figure 2.3: The Peano curve's formation after three iterations. The actual fractal is the result of an infinite number of iterations and its limit is a space-filling curve.

fractal dimension and non-differentiability. The fractal dimension idea is a generalization of the classical topological dimension. While topological dimension is an integer (lines and curves are one-dimensional, planes and surfaces are two-dimensional and so on), a fractal dimension can take any value between integers. Thus, a fractal shape can have a dimension of, for example, 1.24, which means that such a fractal shape is neither a curve nor a surface. Informally, the fractal dimension property is the consequence of infinite pattern iterations at any scale which cause that the length between two points on a curve is infinite. We can even find fractal curves that can fill a two-dimensional space, as the Peano curve (see Fig. 2.3).

Fractal shapes are present in nature and describe many natural systems' development; the formation of trees or river canyons can be modeled as the construction of a fractal shape. They can model other complex natural systems like clouds. In fact, one of the L-systems main applications is the modeling of plants.

In order to link L-systems and fractals, we need to do a graphical interpretation of the L-system's symbols. The word created by an L-system has to describe the painting method for the defined shape. A very common one, called turtle graphics [Papert, 1980], is considering the symbol F a fix-length forward movement of the drawing tool and the symbols $+$ and $-$ an angle increment of 90 degrees to the right or left respectively. This way, we could create a very simple L-system like the following:

- $x : F$
- $F \rightarrow F - F + F + F + F - F - F - F + F$

The previous example draws the Peano curve as we can see in Fig. 2.3. The axiom works as the fractal initiator and the only rule as the generator.

As we have shown, the expressive power of L-systems is big, since it can describe complex systems as fractals. However, they have many other applications, the L-system formalism has been used for the modeling and simulation of many biological or natural systems like structural models of trees, biological developmental processes, fungal growth, etc. It has also made contributions to formal language theory or musical composition.

$000 \rightarrow 0$	$100 \rightarrow 1$
$001 \rightarrow 1$	$101 \rightarrow 1$
$010 \rightarrow 1$	$110 \rightarrow 0$
$011 \rightarrow 0$	$111 \rightarrow 0$

Table 2.3: Transition function of the Cellular Automaton

2.1.2.2 Cellular automata

Cellular automata (CA) are formal devices which are discrete and deterministic. They are composed of many simple and identical components (usually finite-state automata) that interact and work in a parallel manner. The interaction is local, but it creates complex behaviour globally. They serve as models for many complex systems that share the mentioned features. In fact, they capture the main mechanisms of many natural systems whose complex behaviour arise from a self-organized set of local agents cooperating.

As in previous cases, there are many variants of the formalism, however any CA presents the following features (based on de Castro [2006]):

- It is composed of a lattice of cells of one or more dimensions.
- All cells works the same way.
- Each cell is formalized as an automaton with a finite number of states. The cell takes only one possible discrete state each time.
- Cells only interact with neighbour cells.
- States change in discrete time steps. Thus, the system has a discrete dynamics.
- Each cell has a neighbourhood or, in other words, a set of local neighbours whose states determine the cell's future state transition.
- A *transition function* defines the automaton that rules each cell. This function maps the neighbourhood states to a new state for the cell in consideration.

The whole lattice/grid state is determined by each cell's state. More formally, the neighbourhood can be defined as a function like $NF : n \times d \rightarrow \mathbb{N}$, where n is the set of neighbours, d is the set of dimensions and the result is the step to do in that dimension to find the neighbour. The transition function has the form $TF : s_1 \times s_2 \times ... \times s_{k+1} \rightarrow s$, where s is the set of possible states and k the number of neighbours. Hence, the next iteration state of a cell is a function of its current state and its neigbours' states.

For a better explanation, we will give the example presented in de Castro [2006]. Consider the simplest CA with a one-dimensional binary grid and a neighbourhood of two elements, the right and left neighbours of the cell. This way, the state s_i of the *i–th* cell is updated each discrete time step following a function like $s_i(t+1) = f(s_{i-1}(t), s_i(t), s_{i+1}(t))$. The exact transition rules are given in table 2.3 and figure 2.4 shows the output of the automata after eight steps.

The scope of cellular automata is big since they are universal computers. Thus, any application can be conceived, although they are specially suitable for simulating inherently parallel systems like insect swarms, bacteria colonies, clouds, etc. Moreover, they are an important tool for dynamical system theory because they are capable of simulating many aspects of temporal dynamics such as chaos, fractals, ordering, etc.

```
0 __________X_________
1 _________XXX________
2 ________X___X_______
3 _______XXX XXX______
4 ______X___X___X_____
5 _____XXX_XXX_XXX____
6 ____X___X___X___X___
7 ___XXX_XXX_XXX_XXX__
```

Figure 2.4: Eight first steps of our example Cellular Automata. The initial state is "1" for the center cell and "0" for the rest.

2.1.2.3 P-Systems

P-Systems and its variants form an area of research called *membrane computing*. They are all inspired in biological cells, their structure, their inner functioning and their chemical interaction. The model abstracts the way chemicals react and change the inner state of a cell and also the way they pass cell's membranes to affect other cells. As other bio-inspired models, P-Systems are intrinsically parallel because they are composed of many cells and, besides, all chemicals react at the same time. The P-System model was introduced by Păun et al. [1998], Păun [2000].

Informally, a P-System consists of a number of membranes which are structured in a hierarchical way, in the sense that each one can contain others. This way, the only membrane that is not contained by any other is called the container-membrane. The outside of the container-membrane is called the environment. At the beginning of computation, membranes contain a finite number of chemicals, catalysts and rules. Chemicals are the basic symbols to compute on, catalysts have the same role as their counterparts in chemistry and trigger reactions or, in our case, the application of rules. A chemical reaction or a rule defines the production of new chemicals from the present ones. Rules can also cause chemicals to pass a membrane or even the dissolution of a membrane. We should remark that each type of symbol is presented in a finite number of copies, thus, a membrane content is defined by multisets. Figure 2.5 shows a scheme of an example P-System.

Computation is carried out through a number of discrete time steps. In each time step all membranes apply their rules in a parallel and non-deterministic way. The parallel feature means that every possible rule application must take place. By non-deterministic, we mean that if the same symbol or group of symbols satisfy more than one rule, the rule to be applied is chosen randomly. This implies a non-deterministic behaviour of the whole system or, in other words, different outputs for the same P-System and initial condition. The next paragraphs give more details.

Rule application As stated, rules define possible chemical reactions or, in our model, the production of new symbols from the existing ones. This is described by the following notation: $A \rightarrow B$, where A stands for the multiset of symbols required to apply the rule and B stands for the multiset of new symbols produced. If the rule is applied the symbols in A are consumed and removed from the membrane. A single membrane can have more than one rule and some of them could be applied at the same time step. In that case, the system randomly chooses the rules to apply, which can lead to different computation paths and it is the origin of the non-deterministic behaviour of P-Systems. It is also possible to assign priorities to rules, in our figured example 2.5 two rules with the same left side are assigned to different priorities through the symbol $>$.

Figure 2.5 presents the P-System initial state of a classical example [Păun et al.,

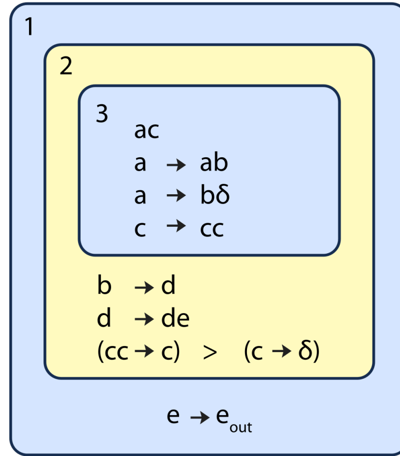


Figure 2.5: An example P-System. The image was taken from *Wikimedia Commons* and is in the public domain.

1998]. This P-System outputs randomly square numbers. We must note that the symbol δ produces the dissolution of the containing membrane. We describe a possible computation of this example step by step below:

- In the initial state, the P-System only has two symbols in membrane 3 ($\{a, c\}$). We can also see the rules contained by each membrane in figure 2.5.

Step 1 The only membrane that can apply rules is number 3 because the others are empty. Given the symbols, all three rules can be apply. Unfortunately, the first two rules need the chemical a and, therefore, we can only apply one of them. We will assume that the first rule is executed $a \rightarrow ab$. The last rule multiplies the number of c symbols by two. At the end of this step, membrane 3 contains $\{a, b, c, c\}$

Step 2 The previous considerations apply, but in this case the second rule is chosen $a \rightarrow b\delta$. Since δ appears, membrane 3 dissolves and its content passes to number 2. At the end of this step, membrane 2 contains $\{b, b, c, c, c, c\}$

Step 3 Membrane 2 is the only one with chemicals inside. Rule $b \rightarrow d$ cause all the b symbols to become d symbols. Furthermore, the second rule can not be applied, since there are no d symbols yet. Finally, rule $cc \rightarrow c$ makes use of its priority and divides the number of c symbols by two. At the end of this step, membrane 2 contains $\{d, d, c, c\}$

Step 4 The first rule can not be apply since there are no more b symbols, instead rule $d \rightarrow de$ produces as many e symbols as d symbols present. Again, rule $cc \rightarrow c$ divides the number of c symbols by two. At the end of this step, membrane 2 contains $\{d, e, d, e, c\}$

Step 5 Again, rule $d \rightarrow de$ produces as many e symbols as d symbols present. However, rule $cc \rightarrow c$ can not be applied any more because there is only one c symbol. Therefore, rule $c \rightarrow \delta$ makes membrane 2 two dissolve and only membrane 1 survives. At the end of this step, membrane 1 contains $\{d, e, e, d, e, e\}$

Step 6 Rule $e \rightarrow e_{output}$ produces as many e_{output} symbols as e symbols present. At the end of this step, membrane 1 contains $\{d, e_{output}, e_{output}, d, e_{output}, e_{output}\}$

Step 7 No more changes are possible. Computation halts.

Considering the number of e_{output} symbols as output, we get the number 4 which is a square number. If we study in detail the example, we can see how it generates a random square number in every computation. The final number depends on the only non-deterministic decision that happens in membrane 3. Note that rule $a \rightarrow ab$ is executed a random number of times until the rule $a \rightarrow b\delta$ is chosen, which causes the dissolution of membrane 3. After that, membrane 2 contains n copies of the symbol b and 2^n copies of the symbol c. Once computation occurs in membrane 2, rules $cc \rightarrow c$ and $c \rightarrow \delta$ divide the number of c symbols by two until the last copy of the symbol c triggers the dissolution of the membrane. Hence, given the number of c symbols and the division factor, $n+1$ steps take place until membrane 2 is dissolved. On the other hand, during the first step in membrane 2 every copy of the symbol b is replaced by copies of the symbol d and, thereafter, n copies of the symbol e are produced each step. Therefore, once membrane 2 dissolves, $n \times n$ copies of the symbol e pass to membrane 1. Obviously, at the end computation there are $n \times n$ copies of the symbol e_{output}, which is a square number.

P-Systems has been proof to be computationally complete. Furthermore, Păun [2001] demonstrate that P-Systems can solve NP problems in polynomial time thanks to their parallel nature. Since they were presented, many applications for biological modeling, computer science or, even, linguistics have been published [].

2.1.3 Nature as hardware

Most models presented above try to take advantage of the intrinsic parallelism of natural phenomena. The main reason for that is the efficiency improvements we can obtain from massively parallel devices. Specifically, in the case of NP problems which could be solved in polynomial time with an intrinsic parallel computer. We must remember that NP stands for *non-deterministic polynomial* time which means that it can be solved in polynomial time by a non-deterministic Turing machine. Note that a non-deterministic automaton can be conceived as a set of deterministic ones working in parallel. This interesting idea is behind many natural computational models and also encourage the search for a real implementation of such a model. Below, we will introduce two new computing paradigms directly inspired by real hardware platforms presented in nature.

2.1.3.1 DNA computing

DNA computing is a subset of *molecular biology* which can be defined as the field that makes use of biological molecules and operations on them to perform computation. In the case of DNA computing, the molecules are those used to store genetic information; DNA molecules. In DNA, the coding alphabet to represent information contains four letters {A,C,T,G}, which stand for the four kind of bases (the main components of DNA): adenine (A), guanine (G), cytosine (C) and thymine (T). Each base is inserted in a nucleotide which is a molecular structure capable of containing the bases. Roughly speaking, the DNA molecule is composed of a strand of these 4 bases and present many interesting features from a computational perspective. Before we detail these features, we will give an insight into the advantages of DNA computing.

Conventional computers are silicon-based and their logical gates are made from transistors. The smaller the transistors the faster the computer. Unfortunately, the size can not be indefinitely reduced, since as we approach the molecular or atomic scale the laws of classical physics can not be applied anymore and, therefore, the transistor-based logical gates are not valid. Furthermore, conventional computers (von Neumann's architecture) are mainly serial. However, DNA computing has the following unique and interesting features [de Castro, 2006]:

- The main data structure is the DNA molecule. Thus, it works with an alphabet of four elements {A,G,C,T} instead of a binary alphabet {0,1}. The structure of DNA needs to be manipulated in a different way from ordinary bits in silicon-based computers.
- DNA molecules can work in a massively parallel way.
- Computation takes place at a molecular level, with the corresponding velocity and space advantages.
- DNA molecules have a great energy efficient functioning, as well as economical information storage.
- Thanks to their parallel nature, DNA computers are specially suited to solve NP-complete problems.

In more detail, DNA consists of two strands bonded together. They remain together because the bases are said to be complementary, which means that A and T bases tend to link in pairs, as well as G and C. This is due to chemical attraction. Hence, DNA is a strand of complementary pairs. For example, in the case of a single strand like $TCGA$, its complementary strand needed to form the whole DNA molecule would be $AGCT$. A very large amount of DNA strands can be presented in a laboratory solution and manipulated through well-know operations. Highly intrinsic parallelism comes from the fact that those operations are performed at the same time for every single DNA strand. The principal operations are (following de Castro [2006]):

Denaturation It separates the complementary DNA strands by heating the molecule. After this, only single strands of DNA are present.

Annealing The opposite of denaturation. It cools the solution to permit the creation of double-stranded DNA molecules. If the single strands in the solution are not completely complementary the chemical attraction does not work. In that case, DNA molecules bind at those partial segments where complementarity exists and sticky ends appear in those segments with no complementary pair.

Polymerase extension It completes DNA molecules with sticky ends by adding the missing nucleotides. This work is carried out by a kind a enzymes called polymerases.

Nuclease degradation It shortens DNA molecules by removing nucleotides. They can be removed from different parts of the molecule depending on the enzyme used.

Endonucleases cutting It cuts DNA molecules. The cutting point can be chosen making use of different kinds of endonucleases enzymes.

Ligation It links two DNA molecules with sticky ends.

Modifying nucleotides It inserts or deletes short subsequences of bases in the DNA molecule.

Amplification It creates multiple copies of the DNA strands in the solution.

Gel electrophoresis It measures the length of DNA molecules and separates them by length. This operation is often used to identify the output of a DNA computing algorithm.

Filtering It extracts specific molecules from the solution. They are identified by a specific pattern in their base sequence.

Synthesis It creates DNA molecules with a pre-determined base sequence.

Sequencing It reads out the sequence of bases present in DNA molecule.

Taking the above operations and others as primitive operators, different DNA-based universal computers have been presented, see de Castro [2006] for a review. Furthermore, it has been demonstrated that NP-problems can be solved efficiently with DNA. The already classic Adleman [1994] experiment solved the Hamiltonian path problem with DNA strands solutions in a laboratory. The number of laboratory steps was linear. Addleman's algorithm takes advantage of DNA computing's massive parallelism to randomly create every possible path in the graph. Different DNA molecules represent each node in the graph and, using some of the aforesaid operations, the algorithm ligates them to build up all possible paths. Note that there is a huge amount of copies for each node and they are all used in parallel. After that, different filtering and detection operations identify the correct path. The key point of the algorithm is the first step, where it tries out all possible paths in the graph. Obviously, the algorithm is a brute force algorithm, but it can afford the huge amount of ligation operations thanks to the parallel nature of DNA computing. Adleman [1994] reported that his DNA algorithm was approximately 1,200,000 times faster than the fastest supercomputer in those days and consumes 10^{10} times less energy per operation.

2.1.3.2 Quantum computing

Quantum computing is quite a young research area that has not still contributed practical or technological advances, but it has offered many stimulating ideas in the search for a massively parallel computer that could overcome the limitations of conventional computers. It is based on quantum physics, the physic branch that explains the natural phenomena at atomic or sub-atomic scales. Unfortunately, classical physics (based mainly in Newton's Mechanics and Maxwell's Electromagnetism theories) is very successful in explaining astronomy or geology, but it is not capable to predict and clarify the phenomena of atomic or sub-atomic systems. This is because the principles of classical physics are radically different to those of quantum physics. In fact, the core ideas of classical physics are easily understood since they are intuitive and follow classical logic we are used to, however, quantum physics is extremely counter-intuitive and seems to violate classical logic. Quantum physics' special principles are the reason for the computer science interest in quantum computing.

We are not explaining in detail quantum theory, since it is out of the scope of this introduction and our knowledge. We are just roughly explaining the main reason for the quantum computing's parallel capabilities. Quantum information is not based on classical bits, but on quantum bits or, in sort, *qubits*. They represent quantum systems with two states, like polarized photons or nuclear spins. However, quantum effects take place in those systems, by which a qubit state can be in '1', '0' or a superposition of both. This means that a computer register of n qubits can be in a superposition of up to 2^n states at the same time. Note that a classical register of n bits can only be in 1 state at one time of the 2^n possible states. Given this principle, qubits are passed through quantum gates, in the same manner bits pass through logical gates in conventional computers. Quantum gates operate on qubits with an input/output function similar to logical gates. These gates are the basic blocks of any quantum algorithm. The ability of qubits to consider any possible state in parallel makes it possible to perform parallel computation. An analogy

for this behaviour could be a deterministic automaton that, thanks to quantum effects, can performs its computation in a new non-deterministic way. Obviously, the temporal complexity of hard problems computed in that new non-deterministic automaton would decrease dramatically with respect to the old deterministic one.

A physical implementation of an universal quantum computer is far from being technically possible. However, different models of quantum universal computers have been proposed, as well as quantum algorithms. Grover [1997] demonstrated that a quantum algorithm could search an element in an unsorted database in $O(n^{1/2})$ time. More examples and detailed explanations can be found in de Castro [2006].

2.2 Evolutionary Automatic Programming and Grammatical Evolution

During this research, we have made use of Grammatical Evolution (from now on GE). GE is an Evolutionary Automatic Programming technique, which is a subclass of the Automatic Programming methods. By Automatic Programming we mean the task of generating a computer program automatically from a high-level description of its requirements. In other words, implementing an algorithm that is capable of creating a program without the intervention of human programmers.

Automatic Programming can be divided into two main branches; the previously mentioned Evolutionary Automatic Programming and Inductive Logic Programming. The latter branch consists of using machine learning tools and logic programming together in order to automatically generate programs from a database of facts. It usually uses PROLOG as the target programming language and, thus, inherits the limitations and advantages of that programming paradigm [Muggleton, 1992].

We will focus our attention on the other Automatic Programming branch, Evolutionary Automatic Programming, to which GE belongs. By *Evolutionary* we mean that these Automatic Programming techniques use methods and concepts of Evolutionary Computation. Evolutionary Computation is a very large and diverse family of algorithms, which have being inspired in natural evolution. All these algorithms perform an stochastic search throughout a landscape of possible solutions to a problem. Firstly, an aleatory population of solutions is generated, after that, evolutionary mechanism such us mutation, crossover and survival of the fittest are applied to the population and its individuals until a suitable solution is found. A detailed survey of evolutionary computation is outside the scope of this work, refer to Eiben and Smith [2003] for a large introduction.

Within Evolutionary Automatic Programming, Genetic Programming (from now on GP) was the first approach to be developed [Koza, 1989]. Soon, it encountered several problems that were attempted to be solved by different variants. These attempts favour the design of the GE system, which finally offered a general and well-structured solution to those problems. Along this chapter, we will present a review of this research process, as well as a thorough explanation of the GE system and its last advances and challenges.

2.2.1 Genetic Programming and its weaknesses

Although the term *Genetic Programming* is mainly used to refer to Koza's pioneer work [Koza, 1992], which is mainly characterized by its tree-based representation of the individuals in the population (in LISP language), it is important to note that the term *Genetic Programming* also comprehends all the variants that have appeared since then.

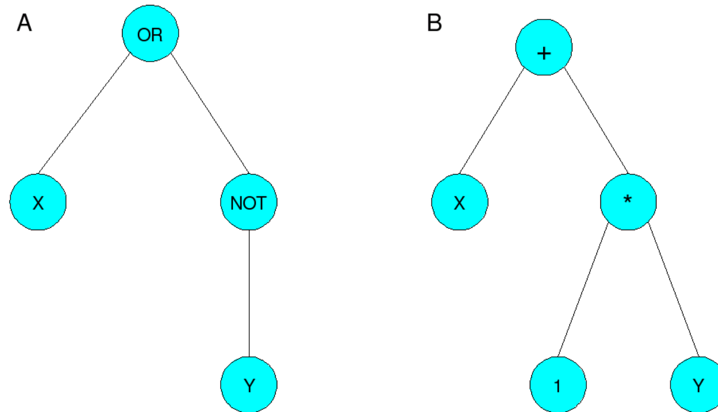


Figure 2.6: Example trees for two simple LISP expressions.

We will mainly focus on classic GP with tree-based representation in LISP, since it represents the largest amount of research in the area and easily shows the weaknesses that most Evolutionary Automatic Programming approaches have and fail to resolve in a general, well-structured way as GE manages to do.

The origin of these weaknesses is that LISP programs are used by the evolutionary engine to make all the evolutionary operations, but they are also the actual final individuals/programs. In other words, using the usual terminology, LISP programs serve as *genotypes*, but also *phenotypes*. By genotype, we mean the individual's representation that suffers the evolutionary variations as mutation or crossover. On the other hand, phenotype is defined as the represenation of the final individual. The transformation from genotype to phenotype is called genotype-phenotype mapping. While using the same representation for genotypes and phenotypes GP did not need any mapping, however, this simplication brought some drawbacks.

The two main weaknesses that the different GP approaches have are:

1. Getting over the closure problem, i. e., generation of valid expressions (or programs) and preservation of its validity as evolutionary variation operators perform (mutation, crossover, etc..).

2. Building a system that is independent of the different languages used. In other words, a system capable of managing arbitrary languages without losing its characteristics, parsimony and advantages.

2.2.1.1 How GP works

Classic GP studies [Koza, 1992, 1994, Koza et al., 1999] make use of the LISP language to represent the programs.

Thus, a typical GP algorithm uses a population of LISP expressions as:

$$(+\ X\ (*\ 1.0\ Y)) \text{ or } (OR\ X\ (NOT\ Y))$$

In fact, these expressions are better understood in the form of trees, as in figure 2.6. Given a population of solutions of that nature, the usual evolutionary operators work as follows;

- **Mutation;** It deletes a subtree of the expression and substitutes it by a new tree, which is created by an aleatory method.
- **Crossover;** Given two expressions or individuals a crossover point is selected in both, resulting in four subtrees, after that, the subtrees of both individuals are exchanged. See figure 2.7 for a crossover example.

However, the first step before implementing the GP algorithm consists of defining the primitives of the system. In other words, the function and terminal sets of the LISP expressions. For example, a terminal set could be a couple of variables and a constant. Together with a functional set of basic algebraic operators we have a simple functional programming language;

$$T = X, Y, 1.0$$
$$F = +, *, -, /$$

Given this specification, some valid expressions could be the previous examples of a LIPS expression or the following;

$$(* \; 1.0 \; (/ \; X \; (* \; Y \; 1.0))) \text{ or } (- \; 1.0 \; (+ \; (+ \; X \; Y) \; X)).$$

With all this in mind, we will show how evolutionary variation operators can generate invalid individuals. The problem has mainly to do with the functions input types. Crossover and mutation are blind to conditions that impose types, therefore can produce expressions where types are not respect. Figure 2.7 gives an example of that situation, a crossover action produces a situation where an integer variable appears inside a logical OR operator and, at the same time, a boolean variable works as an operant inside an arithmetic addition operator.

To avoid this problem, the primitives have to fulfill the *closure* property; each element of the function set has to be able to manage any input the expressions can generate. For some problems, it is not easy to find such a function set, which is one of the most important handicaps of classic GP.

To overcome the so-called closure problem, many other GP systems have been developed. Most of them make use of a linear representation, instead of a tree-based one, and a formal grammar that defines the possible expressions and supports the mechanisms that preserve the validity of the individuals [Banzhaf, 1994, Whigham, 1995, Paterson and Livesey, 1996, Freeman, 1998, Rodrigues and Pozo, 2002]. All these works were clear precursors of the Grammatical Evolution system, as they used a string of integers as the genotype and a formal grammar to make a genotype-phenotype mapping, where the phenotype is the actual expression that we are searching for [2]. However, those systems produce invalid individuals indeed. They apply specific mechanisms to fix them. On the other hand, GE does not create invalid individuals at all, as its functioning make it impossible. Moreover, GE works in a simple and well-structured way that is independent of the language we want to use for our automatically-generated programs.

2.2.2 Grammatical Evolution

Grammatical Evolution, an Evolutionary Automatic Programming system, was presented for the first time in Ryan et al. [1998]. Since then, it has been studied widely in its initial form by its original authors: O'Neill and Ryan [1999a], O'Neill and Ryan [1999b], O'Neill and Ryan [1999c], O'Neill and Ryan [2000], Keijzer et al. [2001] O'Neill et al. [2001d], O'Neill et al. [2001c], O'Neill and Ryan [2003], Ryan et al. [2003]. It represents a unique way of using arbitrary languages in Automatic

[2] These ideas are fully explained for the case of GE in the next section.

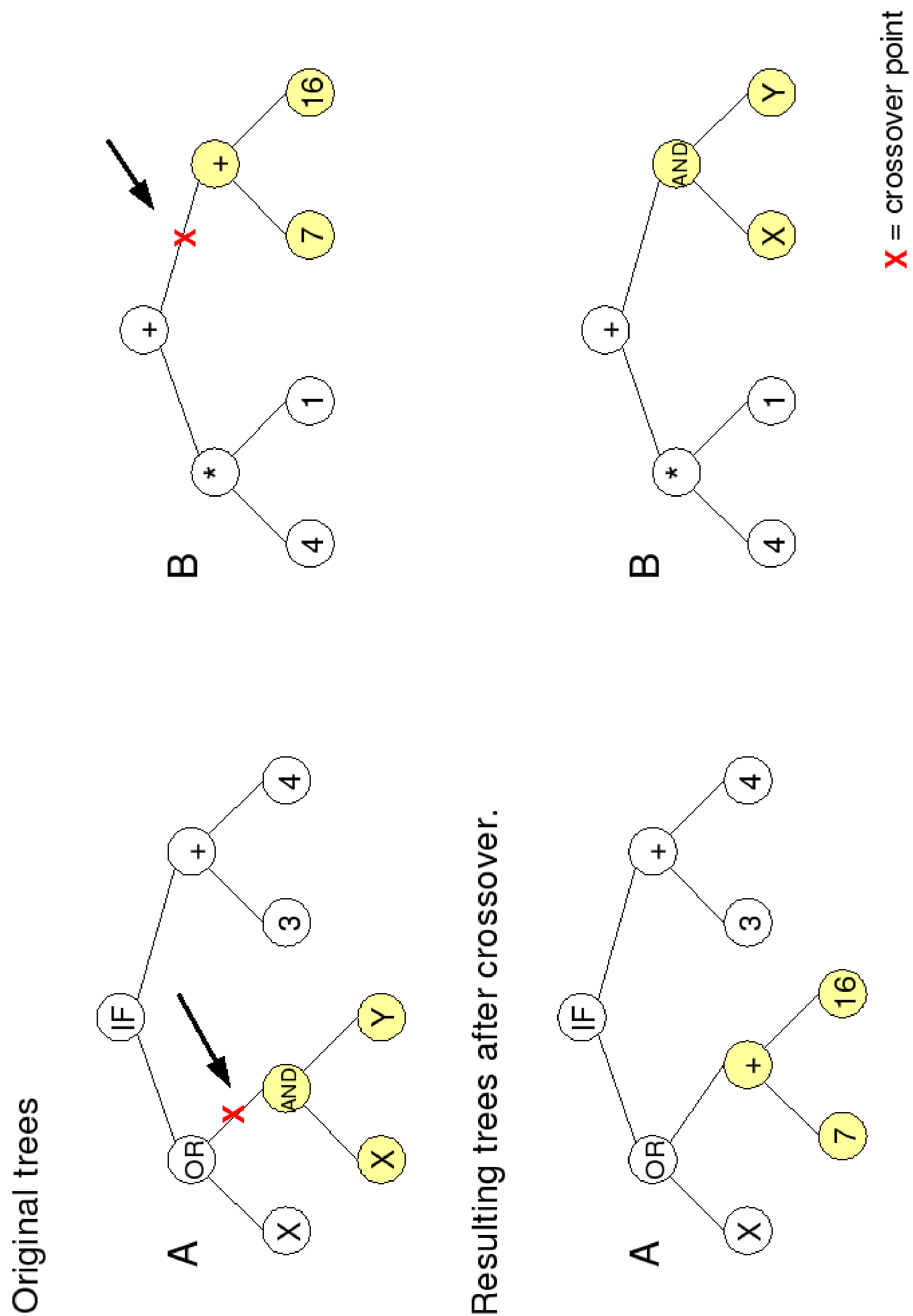


Figure 2.7: Example of a crossover action. The resulting expressions have illegal types. In expression A an integer variable appears inside a logical OR operator, in expression B a boolean variable works as an operand inside an arithmetic addition operator

Programming. Indeed, it can be understood as a tool for searching expressions in any formal language. Its ability to produce only valid expressions within the grammar in a general and parsimonious way is the key virtue of GE.

2.2.2.1 The Grammatical Evolution functioning

GE does not differ much from any other evolutionary algorithm, its representation of individuals is just a string of integers and the evolutionary search is exactly like any other genetic algorithm with that kind of solution representation. Its defining feature is a genotype-phenotype mapping process, by which the integer strings are transformed to valid expressions of the target programming language. A formal grammar specifies the target language and guides the mapping process.

The genotype-phenotype mapping Before performing the genotype-phenotype mapping we need a formal grammar to define the target language for our programs. The typical GE uses context-free grammars. Below is a grammar example in Backus Naur Form, we will employ this grammar for the whole explanation (the grammar is taken from O'Neill and Ryan [2003]).

$$N = expr, op, pre - op, var$$
$$T = Sin, +, -, /, *, X, 1.0, (,)$$
$$S =< expr >$$

```
A) <expr> ::= <expr><op><expr>        (0)
            | (<expr><op><expr>)    (1)
            | <pre-op> (<expr>)     (2)
            | <var>                 (3)

B) <op> ::= +                         (0)
          |-                          (1)
          |*                          (2)
          |/                          (3)

C) <pre-op> ::= Sin

D) <var> ::= X                        (0)
           |1.0                       (1)
```

The grammar defines a very simple language of arithmetic expressions. As we can see the grammar consists of 4 productions rules, each one with a given number of choices (A=B=4, C=1, D=2). The genotype-phenotype mapping is based on the number of choices and the module operator. It generates an expression from the string of integers by expanding a derivation tree repeatedly applying the following rule.

$$Next_rule = Integer_value\ MOD\ Choices_current_nonterminal$$

We always use a left-most derivation of the tree.
Given the integer string;

$[30, 9, 16, 31, 16, 4, 63, 5, 47, 20, 11]$

the steps would be the following ones, starting from the first integer until there is no non-terminals in the expression. The next non-terminal to be considered is always the left-most;

1. Integer = 30
   Current expression = `<expr>`
   Current non-terminal = `<expr>`
   Choices of current non-terminal = 4
   Next rule to apply is 30 $MOD\ 4\ =\ 2$, which is `<expr> ::= <pre-op> ( <expr> )`
   Resulting expression →

   `<pre-op> ( <expr> )`

2. Integer = 9
   Current expression = `<pre-op> ( <expr> )`
   Current non-terminal = `<pre-op>`
   Choices of current non-terminal = 1
   Next rule to apply is 9 $MOD\ 1\ =\ 0$, which is `<pre-op> ::= Sin`
   Resulting expression →

   `Sin ( <expr> )`

3. Integer = 16
   Current expression = `Sin ( <expr> )`
   Current non-terminal = `<expr>`
   Choices of current non-terminal = 4
   Next rule to apply is 16 $MOD\ 4\ =\ 0$, which is `<expr> ::= <expr><op><expr>`
   Resulting expression →

   `Sin ( <expr><op><expr> )`

4. Integer = 31
   Current expression = `Sin ( <expr><op><expr> )`
   Current non-terminal = `<expr>`
   Choices of current non-terminal = 4
   Next rule to apply is 31 $MOD\ 4\ =\ 3$, which is `<var>`
   Resulting expression →

   `Sin ( <var><op><expr> )`

5. Integer = 16
   Current expression = `Sin ( <var><op><expr> )`
   Current non-terminal = `<var>`
   Choices of current non-terminal = 4
   Next rule to apply is 16 $MOD\ 2\ =\ 0$, which is `X`
   Resulting expression →

   `Sin ( X<op><expr> )`

6. Integer = 4
   Current expression = `Sin ( X<op><expr> )`
   Current non-terminal = `<op>`
   Choices of current non-terminal = 4
   Next rule to apply is 4 $MOD\ 4\ =\ 0$, which is `+`
   Resulting expression →

```
Sin ( X+<expr> )
```

7. Integer = 63
   Current expression = `Sin ( X+<expr> )`
   Current non-terminal = `<expr>`
   Choices of current non-terminal = 4
   Next rule to apply is 63 MOD 4 = 3, which is `<var>`
   Resulting expression →

```
Sin ( X+<var> )
```

8. Integer = 5
   Current expression = `Sin ( X+<var> )`
   Current non-terminal = `<var>`
   Choices of current non-terminal = 2
   Next rule to apply is 5 MOD 2 = 1, which is `1.0`
   Resulting expression →

```
Sin ( X+1.0 )
```

Since there is no non-terminal left, the process stops. A syntactically correct expression (phenotype) has been generated from an integer array and a formal grammar. It is easy to show that this method ensures that the expressions are valid. It is possible that the integer array ends before the expression is completely generated, in that case the wrapping operator is executed (see below) or the individual is discard as a syntactically incorrect expression.

After that, the phenotype is confront against the fitness function as in any other evolutionary algorithm. Therefore, GE performs the evolutionary search with the individuals' genotype, but, unlike genetic algorithms where there is no difference between genotype and phenotype, the fitness functions are applied to phenotypes.

System parameters and features Apart from the parameters and features that have to do with the genetic algorithm that works in the background for the GE system, the genotype-phenotype mapping involves other specific elements. They are listed and studied below.

- **Wrapping operator:** As mentioned during the explanation of the phenotype-genotype mapping, it is possible that the integer string ends before the expression is completely expanded. In that case, GE makes use of this operator, which follows the mapping from the beginning of the string again. O'Neill and Ryan [2003] report that using wrapping can improve the algorithm efficiency or, at least, does not worsen it. Wrapping makes it possible to use genetic code more than one time which is a novel characteristic in the evolutionary algorithms family.

- **Degenerate genetic code:** As a direct consequence of the module operator used during the mapping process, more than one integer values can refer to the same production rule. This property by which many different *codons*[3] are *expressed* equivalently in the phenotype is called *Degenerate genetic code* in biology. Again, this characteristic GE feature has been reported to give advantages to the algorithm [O'Neill and Ryan, 2003]. Consequently, a mutation can have no influence in the phenotype of the individual because the integer's change can lead to the same production rule. This way, it permits GE to keep

[3]In biology, a gene is compound by a string of codons.

genetic diversity throughout the run, which can be very useful with problems with a dynamic fitness function or a multimodal nature (very irregular landscapes with a lot of niches or peaks). To show how this work, it is easy to imagine a lot of different genotypes that map into the same phenotype, although they carry a distinct sequence of genes. Under these conditions, as mutation appears on the genotypes, they can result in very different novel phenotypes. This effect is intensified by the fact that during the mapping process one different integer can change the meaning of the following ones.

- **Integers' range:** Usually, the values for the integers in the genotype are between 0 and 128 or 256, but there is not a prior restriction in the range. The larger the range, the stronger the code degeneracy. The optimal degree of code degeneracy depends on the specific grammar and the problem one is trying to solve.

2.2.2.2 Discussion and comments

After the presentation of GE, we can argue why GE is a very useful Automatic Programming system and summarize its characteristic features that make it different to other Automatic Programming techniques. Their main virtues are;

- GE is an Automatic Programming method, however its evolutionary engine working in the background is just a classic genetic algorithm, therefore GE can take advantage of the last advances in that area without noting any difference in its functioning.

- *Wrapping* and *Degenerate genetic code* introduce two useful concepts for the applycation and studying of evolutionary algorithms.

- Its genotype-phenotype mapping procedure creates always valid individuals in a simple way, overcoming one of the key problems in Genetic Programming.

- Its functioning is independent of the language we want to use for our target expressions. This makes GE a very general and versatile automatic programming tool without losing their valuable advantages.

Apart from the theoretical analysis where GE has been faced against different simple benchmark problems [O'Neill and Ryan, 2003], GE has also been applied to real world problems with a significant degree of success. Concerning this, the amount of research is large and diverse, we could mention O'Neill and Ryan [1999c] where GE was employed to generate caching algorithms, Brabazon et al. [2002a,b], Brabazon and O'Neill [2002, 2003, 2004], O'Neill et al. [2001b], Cui et al. [2010], Bradley et al. [2010], **?**], O'Neill et al. [2001a, 2002], as examples of GE applied to financial prediction, Hemberg and O'Reilly [2004] to help surface design in architecture, Moore and Hahn [2003, 2004], Moore et al. [2005] to hierarchical Petri net modeling of complex genetic systems, Ortega et al. [2003] to the design of fractal curves with a given dimension, Tsoulos et al. [2006], Tsoulos and Lagaris [2006] to function estimation and to locate the global minimum of a multidimensional function, Turner et al. [2010], Tsoulos et al. [2008] to neural networks training and construction, Smart et al. [2011] to medical research, Siles et al. [2010] to programming H systems, Sen and Clark [2011] to networks security, Risco-Martin et al. [2011] to the design of memory allocators, Abu Dalhoum et al. [2008], Reddin et al. [2009] to music composition, Perez et al. [2011b,a], Galvan-Lopez et al. [2010] to videogames AI, Peleteiro et al. [2010] to programming multi-agent systems, Nicolau and Costelloe [2011] to 3D image design, Matousek and Bednar [2009b,a] to symbolic regression, Kao et al. [2011], Chen [2011] to water supply management, Cullen

[2008] to digital circuits design, Chen and Wang [2010] to concrete modelling, Burbidge et al. [2009] to robotics, Beaumont and Stepney [2009] to L-systems design, Alexander and Gratton [2009] to compilers optimization and, finally, McKinney et al. [2006] where GE helps to identify nonlinear dynamical systems.

2.2.3 Advances in the Grammatical Evolution framework

The GE framework has many aspects that are improvable and subject to variation. Many of them were pointed out by its original authors in O'Neill and Ryan [2003] (chapters 8 and 9). Some of those ideas have been put in practice; see O'Neill et al. [2001d] for the introduction of introns, O'Neill et al. [2004] for a position-independent translation version of GE and O'Neill and Ryan [2004], O'Neill and Brabazon [2005] for a GE version where the grammar is co-evolved during the evolutionary process.

We will pay special attention, however, to GE extension that have to be with the power of the grammar used. The original GE system employs context-free grammars, which can only generate the subclass of languages called context-free languages. Thus, they can not express any language or, in other words, any recursively enumerable language, which are the languages a Turing machine can accept. This issue is strongly important in our context, where we want to apply GE to modelling problems and, unfortunately, the formalisms needed to express those models are not usually as simple as context-free languages. Below, we will study the possibility of using more powerfull grammars with GE.

2.2.3.1 Attribute Grammar Evolution

Context-free grammars are easy to manipulate, create and study. There are very well known and simple algorithms to handle with the generation of expressions for the grammar as well as with the evaluation of expressions [Aho et al., 1998, Martin, 2003]. They have been widely used to define most aspects of the programming languages' syntax. However, they can only define context-free languages, which are not capable of expressing some other aspects, for example, context-dependent ones. Tipically, a computer program has to declare a variable before using it. Such a restriction is context-dependent and can not be expressed by context-free languages.

Meanwhile, Attribute Grammars [Knuth, 1968] are an extension of context-free grammars that have as much expressive power as possible, i. e. they can define any recursively enumerable language. Moreover, they keep most of the simplicity of context-free grammars as we will see.

The introduction of Attribute Grammars in GE has been already studied in preliminary works [Echeandia et al., 2005, Cleary and O'Neill, 2005] but has not yet been applied to real world problems.

Attribute Grammars Attribute Grammars are not dissimilar to context-free grammars apart from 2 elements;

- Each non-terminal has a set of attributes. They are like variables in a programming language, they have a name, a given value and a domain.
- Every time a rule is executed a set of instructions are computed concerning those attributes. Each rule has its own set of instructions.

We can distinguish two kind of attributes. Having in mind the abstract derivation tree of the grammar:

- **Inherited attributes;** Their values depend on attributes of the parent node or nodes at the same level but calculated before. In other words, the inherited attributes of non-terminals at the right of the rule are dependent on of the attributes of (1) the left side of the rule and (2) their pre-calculated siblings.
- **Synthesised attributes;** Their values depend on attributes calculated in the node's children. In other words, the left side non terminal of the rule calculates its synthesised attributes as a function of the right side attributes of the rule.

For example, given the grammar presented before, we may want to calculate how many operators appear in the whole expression and discard it as a too complex expression if there are more operators than a threshold value. In that case we should add the following attribute calculations to the rules.

If the same non terminal appears two or more times in the same rule, we use subscript indices to differentiate them. We use a notation where inherited attributes have down-arrow symbols like ↓ and the synthesised ones have up-arrows like ↑ (this notation was introduced by Watt and Madsen [1977]). Attributes are in brackets next to their non terminal. We omit those rules that have no attribute calculations.

- <expr>(↑ops) ::= <expr>$_A$(↑ops) <op> <expr>$_B$(↑ops)
  {

  <expr>.↑ops = <expr>$_A$.↑ops + <expr>$_B$.↑ops + 1

  tooComplex(<expr>.↑ops) //It discards the expression if it has too many operators

  }
- <expr>(↑ops) ::= (<expr>$_A$(↑ops) <op> <expr>$_B$(↑ops))
  {

  <expr>.↑ops = <expr>$_A$.↑ops + <expr>$_B$.↑ops + 1

  tooComplex(<expr>.↑ops) //It discards the expression if it has too many operators

  }
- <expr>(↑ops) ::= <pre-op> (<expr>$_A$(↑ops))
  {

  <expr>.↑ops = <expr>$_A$.↑ops + 1

  tooComplex(<expr>.↑ops) //It discards the expression if it has too many operators

  }
- <expr>(↑ops) ::= <var>
  {

  <expr>.↑ops = 0

  }

Thanks to those attributes calculations, as we derive the expression's tree the number of operators is counted from the leaves of the tree to the top.

Integration in the GE system Following Echeandia et al. [2005] we can easily integrate the grammars in GE applying the next algorithm during the phenotype-genotype mapping. Every time we expand a node of the tree, in other words we parse an integer of the genotype, the attributes are computed as follows (remember we assume left-most derivation):

1. Attributes inherited from the parent node are calculated directly as the rule is applied.
2. Attributes inherited from the left siblings are calculated as soon as all the siblings' attributes are available.
3. Synthesized attributes are calculated after the last children of the node is expanded.
4. The axiom has only synthesized attributes.
5. The terminal symbols (leaves of the tree) have no attributes but can be used as input data for other attributes.

2.2.3.2 Christiansen Grammar Evolution

The reasons to integrate Christiansen Grammars [Christiansen, 1985, 1986, 1990] in GE are the same as those for Attribute Grammars. In fact, Christiansen Grammars are just Attribute Grammars with some special features that can be very usefull in some cases. They solve some problems in a more simple and parsimonious way than Attribute Grammars.

Again, Christiansen Grammars in GE has been already study in preliminary works [Ortega et al., 2007], but has not been yet applied to real world problems.

Christiansen Grammar A Christiansen Grammar is just an Attribute Grammar where all the non terminals has as its first attribute a so-called *Christiansen Grammar* attribute. This attribute is an Attribute Grammar itself that defines the production rules that can be applied when expanding that non terminal. This attribute is always inherited, except for the case of the axiom, in this case its first attribute is the initial Christiansen Grammar.

This first attribute can change by adding or deleting rules during the attribute's computations. In practice, all this mean that the actual grammar that is applied to each non terminal suffers modifications as the derivation tree is expanded. Firstly, the axiom has the initial Christiansen Grammar specified by the designer, thereafter this Christiansen Grammar is modified depending on the specific derivation tree created.

For instance, we could have the need to avoid values "1.0" inside a "<pre-op>" operator in our example grammar. Having an operator to delete rules from the Christiansen Grammar called *delete(↓g, ruleToDelete)*, we could accomplish this task with the following Christiansen Grammar. The first attribute or Christiansen Grammar is *↓g*. We omit those rules that have no attribute calculations.

- <expr>(↓g) ::= <expr>$_A$(↓g) <op> <expr>$_B$(↓g)
  {

  <op>.↓g = <expr>.↓g

  <expr>$_A$.↓g = <expr>.↓g

  <expr>$_B$.↓g = <expr>.↓g

  }

- <expr>(↓g) ::= (<expr>$_A$(↓g) <op> <expr>$_B$(↓g))
  {

  <op>.↓g = <expr>.↓g

  <expr>$_A$.↓g = <expr>.↓g

  <expr>$_B$.↓g = <expr>.↓g

  }

- <expr>(↓g) ::= <pre-op>(↓g) (<expr>$_A$(↓g))
  {

  <pre-op>.↓g = <expr>.↓g

  //Rule delete in expressions inside <pre-op> operators

  <expr>$_A$.↓g = *delete(<expr>.↓g, "<var> ::= 1.0")*

  }

- <expr>(↓g) ::= <var>(↓g)
  {

  <var>.↓g = <expr>.↓g

  }

Figure 2.8 shows an example derivation tree for the expression:

$$1 + sen(X)$$

The attribute's computations are not explicitly shown but commented in red.

Integration in the GE system Since Christiansen Grammars are just Attribute Grammars with their special first attribute, the integration in GE follows the same indications as those given for Attribute Grammars 2.2.3.1. The only new element to take into account is the fact that the actual grammar employed in each non terminal is the one in its first attribute.

2.3 NEP

Conventional computers are based on the well known von Neumann architecture, which can be considered an implementation of the Turing machine. One of the current topics of interest in Computer Science is the design of new abstract computing devices that can be considered alternative architectures for the design of new families of computers and algorithms. Some of them are inspired by the way in which Nature efficiently solves difficult tasks; almost all of them are intrinsically parallel (see section 2.1). They are frequently called *natural or unconventional computers*. Networks of Evolutionary Processors (NEPs) are one of these massively parallel new natural computers.

NEPs are a new computing mechanism directly inspired by the behaviour of cell populations. Each cell contains its own genetic information (represented by a set of strings of symbols) that is changed by some *evolutive* transformations (implemented as elemental operations on strings). Cells are interconnected and can exchange information (strings) with other cells.

Thus, from a computational perspective, cells are nodes interconnected forming a net. Each node in the net is a very simple processor which contains words and

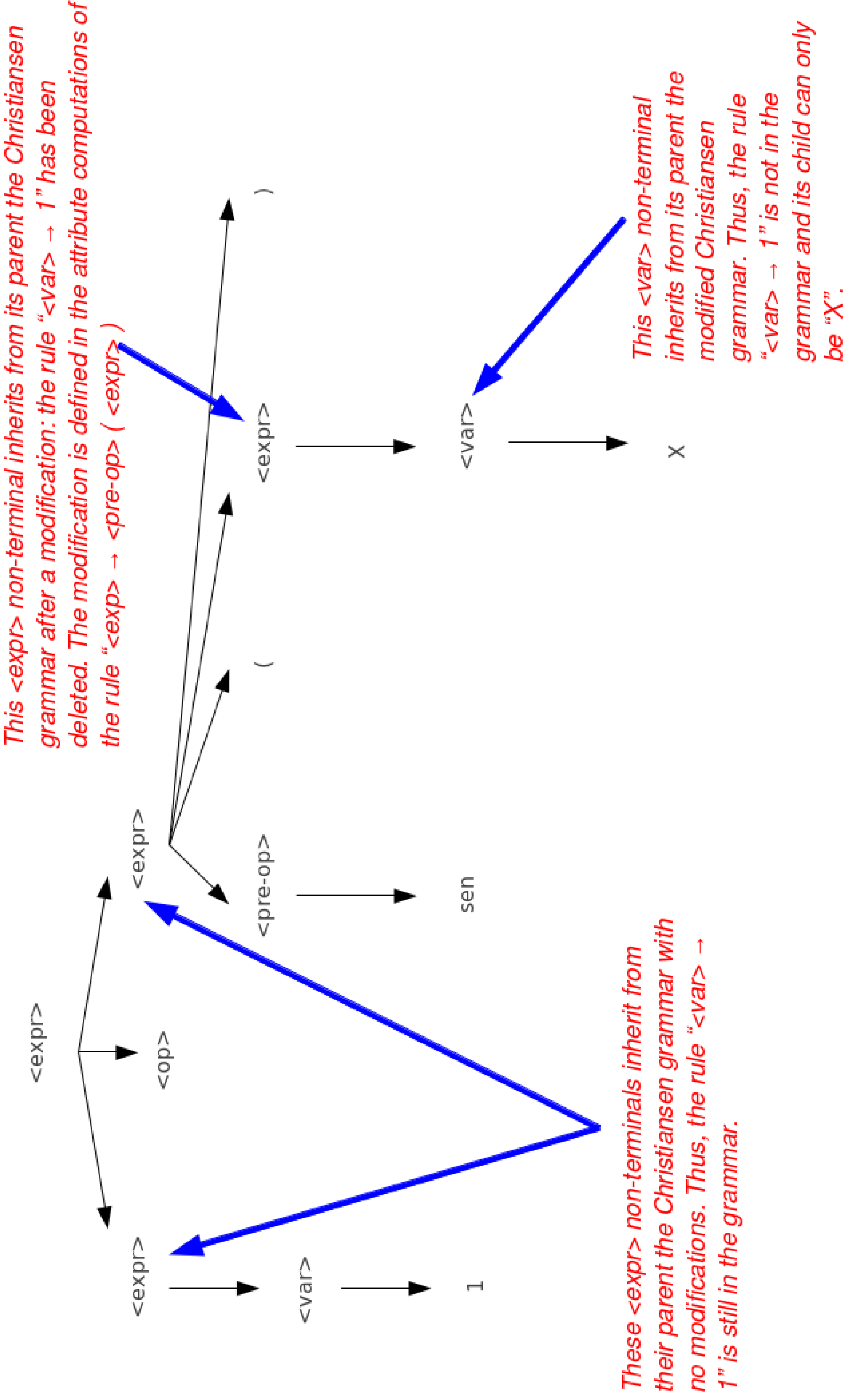


Figure 2.8: Derivation tree for the expression "1 + sen (X)"

performs a few elementary tasks to alter the words or send and receive them to/from other processors.

NEPs were initially used as generating devices in Csuhaj-Varju and Salomaa [1997], Csuhaj-Varjú and Mitrana [2000], and Castellanos et al. [2001]. Broadly speaking, The Connection Machine [Hillis, 1985], the Logic Flow paradigm [Errico and Jesshope, 1994], the biological background of DNA computing [Păun et al., 1998], membrane computing [Păun, 2000], and especially the theory of grammar systems [Csuhaj-Varjú et al., 1993] can be considered precursors of NEPs. The development of the NEP model and its variant follows the chronology presented below.

The aforementioned Connection Machine and Logic Flow paradigm are two computational models designed for parallel processing. They are composed of a set of processors connectted as in a complete graph. Each node contains local data and rules that compute on the data. When two nodes are connected they can share their local data. The connections carry out a filtering process so as to allow or forbid some data to pass. All the local data is sent and received at the same time in a parallel manner. As we will discuss in detail, the general structure and functioning principles are very similar to the NEP model's.

Later on, Csuhaj-Varju and Salomaa [1997] presented a model called *networks of parallel language processors*. The authors were interested in devoloping new ideas from previous works like *grammar systems* [Csuhaj-Varjú et al., 1993]. They were focused on the possibilities of parallel/distributed arquictectures to generate languages and study formal grammars. The underlying idea was very closed to the NEP concept. Each processor manipulates words with simple rewriting operations. In addition, processors are connected following a graph definition. Words can move from one processor to other, if they are able to pass output/input filters in the sending/receiving processor.

Finally, Castellanos et al. [2001] first used the term *Network of Evolutionary Processors*. The main idea was inspired from cell biology. Again, multiple simple processors are placed in a network. Processors manipulate words by very simple operations that remind mutations in DNA sequence, which is the main reason to call them "evolutionary". The whole model was inspired by cell biology and DNA mechanisms: words are DNA sequences that can evolve due to local operations and processors are cells sharing information through their membranes. The original operations or *rules* were insertion, deletion and substitution and each node was specialized in only one of them. The model assumes an arbitrarily large number of copies is availabe for each word and every rule that can be applied to a word is actually applied. This last point is the origin of the intrinsic parallelism of NEPs computation. Moreover, communication between the processors works as in previous models.

After the seminal work of Castellanos et al. [2001] many different studies were published where NEPs were used as generating devices (also called GNEP), as accepting devices (also called ANEPs) and as universal computers, in which case their computational power was studied. Different kinds of filters were presented, mainly two types. Firstly, membership filters which define the output/input filters with a set of words or a regular expression [Castellanos et al., 2003]. The second one is based in random context conditions [Martín-Vide and Mitrana, 2005], which are more biologically plausible and simpler to implement. In addition, the first papers impose many constraints of flexibility on using different kinds of rules or filters in the same NEPs, while later approaches do not apply these restrictions [Martin-Vide et al., 2003, Margenstern et al., 2005]. In that case, some authors consider those NEPs hybrid and, consecuently, call them Hybrid Networks of Evolutionary Processors. Finally, other variants have been presented like those using splicing rules [Csuhaj-Varju et al., 2005, Manea et al., 2007], also called nets of splicing

processors NSP.

As mentioned, there exists many NEP variants in the literature. Nevertheless, all of them share the same general characteristics. A NEP is built from the following elements:

a) A set of symbols which constitutes the alphabet of the words which are manipulated by the processors.

b) A set of processors.

c) An underlying graph where each vertex represents a processor and the edges determine which processors are connected so they can exchange words.

d) An initial configuration defining which words are in each processor at the beginning of the computation.

e) One or more stopping rules to halt the NEP.

An evolutionary processor has three main components:

a) A set of evolutionary rules to modify its words.

b) An input filter that specifies which words can be received from other processors.

c) An output filter that delimits which words can leave the processor to be sent to others.

The variants of NEPs mainly differ in their evolutionary rules and filters. They perform very simple operations, like altering the words by replacing all the occurrences of a symbol by another, or filtering those words whose alphabet is included in a given set of words.

NEP's computation alternates evolutionary and communication steps: an evolutionary step is always followed by a communication step and vice versa. Computation follows the following scheme: when the computation starts, every processor has a set of initial words. At first, an evolutionary step is performed: the rules in each processor modify the words in the processor. Next, a communication step forces some words to leave their processors and also forces the processors to receive words from the net. The communication step depends on the constraints imposed by the connections and the output and input filters. The model assumes that an arbitrary number of copies of each word exists in the processors, therefore all the rules applicable to a word are actually applied, resulting in a new word for each rule. The NEP stops when one of the stopping conditions is met, for example, when the set of words in a specific processor (the output node of the net) is not empty.

Detailed formal descriptions of NEPs can be found in Castellanos et al. [2003], Csuhaj-Varju et al. [2005] or Manea et al. [2007]. There are different variants of NEPs and, therefore, formal definitions can vary slightly. For example, the following definition is literally taken from Castellanos et al. [2003] and will help the reader to understand the model.

A network of evolutionary processors (NEP for short) of size n is a construct $\Gamma = (V, N_1, N_2, ..., N_n, G)$, where V is an alphabet and for each $1 \leq i \leq n, N_i = (M_i, A_i, PI_i, PO_i)$ is the i-th evolutionary node processor of the network. The parameters of every processor are:

- M_i is a finite set of evolution rules of one of the following forms only
    - $a \to b, a, b \in V$ (substitution rules),

- $a \to \lambda, \in V$ (deletion rules),
- $\lambda \to a, a \in V$ (insertion rules),

More clearly, the set of evolution rules of any processor contains either substitution or deletion or insertion rules.

- A_i is a finite set of strings over V . The set A_i is the set of initial strings in the i-th node. Actually, we consider that each string appearing in any node at any step has an arbitrarily large number of copies in that node.
- PI_i and PO_i are subsets of V^*. They are regular languages which represent the input and the output filter, respectively. These filters are defined by the membership condition, namely a string $w \in V^*$ can pass the input filter (the output filter) if $w \in PI_i(w \in PO_i)$.

Finally, $G = (N_1, N_2, ..., N_n, E)$ is an undirected graph called the underlying graph of the network. The edges of G, that is the elements of E, are given in the form of sets of two nodes. The complete graph with n vertices is denoted by K_n.

2.3.1 NEPs in practice

A lot of research effort has been devoted to the definition of different families of NEPs and to the study of their formal properties, such as their computational completeness and their ability to solve NP problems with polynomial performance. However, no relevant effort, apart from Díaz et al. [2007], has tried to develop a NEP simulator or any kind of implementation. Unfortunately, the software described in this reference gives the possibility of using only one kind of rules and filters and, what is more important, violates two of the main principles of the model: 1) NEP's computation should not be deterministic and 2) evolutionary and communication steps should alternate strictly. Indeed, the software is focused in solving decision problems in a parallel way, rather than simulating the NEP model with all its details.

In section 3.1, we will present our NEP simulator called jNEP. It tries to fill the mentioned gap in the literature. Moreover, although simulating NEPs does not imply using a real parallel hardware, it can be a first step to create real parallel implementations of NEPs. For that reason, we are also interested in running NEPs simulators on cluster, as one of the possible ways of exploiting the inherent parallel nature of NEPs. We will see that jNEP has been designed to run in parallel Java platforms. In addition, preliminars attempts has been made to run jNEP variants in high performance clusters [Navarrete Navarrete et al., 2011]. We must note at this point that running in different real processors each node of the NEP does not guarantee a parallel execution of rules in every word contained in the NEP because each word has an arbitrary number of copies. This last parallel feature is the most important one and imposible to implement under a conventional computer architecture. To achive such massively parallel computation, the implementation of NEPs should be based on new hardware paradigms like the ones presented in section 2.1.3. A real implementation of that nature would make possible the use of NEPs to efficiently solved NP-complete problems. We have already simulated that kind of efficient algorithms with jNEP, see section 4.1.

2.4 Two scientific challenges

2.4.1 Modelling associative learning in psychology

Animal associative learning has been studied mainly in the field of behavioural science. Pioneer researches were conducted by Pavlov (see Pavlov [1927]) in the nineteenth century, identifying what is known as classical conditioning. During its highly cited experiments about the digestive system of dogs, Pavlov found that if a stimulus (a little sound or the researcher foodsteps) was presented systematically before the dogs were fed, that stimulus began to elicit the salivary response that, in normal conditions, appears with the presence of food. Explained in a very simplified way, the reason of that learning was the formation of an *association* between the food and the preceding stimulus (a more rigorous explanation will be discuss below). Later, along the twentieth century, intensive research was developed in this area by psychologists, mainly under the methodological perspective of behaviourism, whose principal author and defender was B. F. Skinner (see Skinner [1965]). At the beginning of this period, the second principal type of associative learning was characterized; operant conditioning. In this context, the key issue is the stimuli that *follow* the responses. Depending on the nature of those stimuli, the frequency of the response decreases or increases. In this context, the learning organism builds an association between its responses and whatever follows them (again, a deeper explanation will be discuss below).

As mentioned before, most research in associative learning has been conducted under the methodological perspective of behiorism. It is not our intention to give a profound explanation on this issue or illustrate the intensive debate that behaviorism has produced within psychology and in other disciplines during the twentieth century, but it is useful to have a basic knowledge on it before continuing. From this approach, the main object of the researcher is to find the functional relation between the stimuli of the environment and the responses of the animal. Thus, the atomic elements of this conceptualization are stimuli and responses, in other words; nothing more than what can be directly observed. In fact, one of the strongest assumptions of behaviorism asserts that no other elements are needed to explain, predict and manipulate the behavior of the organism. This assumption excludes from the analysis any other variable, specially mental or inner variables. The exclusion of those variables was one of the defining characteristics of behaviorism in its beginning, since it was intended to fund a *scientific and objective psychology* opposed to other psychological schools of that moment like psychoanalysis or the introspection paradigm. Indeed, a radical behaviorist could go further and defend that the study of any organism's behavior could be resume in the statistical analysis of stimuli and responses acquired by observation (in terms of frequency, intensity etc...), with no need of any other conceptual or theoretical tool.

In summary, most research in associative learning has been developed under this approach. Therefore, the regularities between the responses of the organism and the stimuli of the environment are the main focus of these studies to explain the organism's learning and, thus, its behavior. It is not our aim to illustrate all the flavours of behaviorism and its details; to check a long discussion on this issue refer to O'Donohue and Kitchener [1999].

2.4.1.1 Different types of learning

Associative learning is divided into four groups of phenomena. The first two are actually pre-associative[4] processes, they are sensitization and habituation. The

[4] Usually, the term "non-associative" is used. We prefer "pre-associative" as we understand that those basic types of learning are needed to develop the more complex genuine associative

other two conform the core of associative learning; classical conditioning and operant conditioning. We will describe this four types of learning briefly.

Habituation Habituation functions by decreasing the innate response of the organism to a given stimulus. This decrement occurs as the stimulus is presented repeatedly. For example, the orienting response of many organism after the presentation of a novel sound decreases if the sound is presented many times. Habituation has a different nature than other response decrements like fatigue or adaptation. It has an important function in the organism's learning. Its role consists of filtering stimuli that are irrelevant to the organism. Those filtered stimuli are not processed or taken into account by the organism. See Hall [1991] for a fuller account. Finally, another important feature of habituation is the recovery of the response if the habituated stimulus is not presented for a long time and, then, is presented again. This phenomenon is called *spontaneous recovery*.

Sensitization Sensitization produces a progressive amplification of the response to a stimulus by the repeated presentation of the stimulus. The most important factor is the intensity of the stimulus, which is directly proportional to the response's increment. Most times, sensitization works for a short time, after which habituation occurs.

Classical Conditioning (CC) Classical Conditioning (from now on CC) appears if an unconditioned stimulus (US), a stimulus that elicits a response, is systematically pair with a neutral stimulus (usually the neutral stimulus is presented before the US). After this repeated presentation, the neutral stimulus begins to elicit the response also, now the neutral stimulus is a conditioned stimulus (CS). We say that the US elicits an unconditioned response (UR), since it is innate that the US elicits that response, on the other hand, the CS elicits a conditioned response (CR) as the elicitation of the response by the CS is due to the association created. The classic example involves the Pavlov's dog, which begins to salivate (CR) after the repeated presentation of a sound (CS) before it is fed (the food is an US). Now, not only the food (US) elicits the salivary response but also the sound (CS).

Operant Conditioning In this case, the associations occur between a previous response and the consecutive stimulus. Operant Conditioning works by decreasing or increasing the frequency of a given response of the organism. In the case of a stimulus that works as a *punishment* the frequency will decrease, on the other hand, if it works as a *reinforcement* the frequency will increase. Classic experiments used rats, which were taught to performs simple tasks (for example, a labyrinth) by applying a program of punishments and rewards to their responses.

Last notes and applycations The learning processes explained above have been found in almost any kind of organism, from humans to simple worms. In the case of habituation, it has been found even in single-cell organisms. Therefore associative learning has an all-pervasive nature, which has encouraged psychologists and other scientists to study it as a major issue in the explanation of human learning and behaviour. Moreover, since the middle of the twentieth century, associative learning's general principles are one of the most consistent and established corpus of knowledge in psychology. Unfortunately, the field suffers from a lack of an integral model (i. e. one that includes all types) and formalization of the phenomena that

---

learning. This point of view is explained throughout the folowing sections

would permit a detailed understanding of the processes that produce this learning. For further discussion on this set of phenomena refer to Mazur [2002]

Despite this last week point, associative learning has been applied in many contexts within the general label of "behavioural modification" with a big rate of success. For example, health psychology (treatment of phobias, obsessive-compulsive disorders, etc...), education, industrial psychology or farming. A long explanation of the techniques in Martin and Pear [2002].

Below, we will go over the modelling attempts in this area and make a detailed revision of the current habituation models, which we tried to overcome with this research.

2.4.1.2 Current models and their problems

Although the study of associative learning existed for over a century, formal models [5] did not appear until 1972 with the publication of Rescorla and Wagner [1972]. That lack of formal models is significant in the context of psychology, as most theories in this science are defined by means of general principles, usually written in natural language. This situation provokes a lot of troubles inside the theory, like ambiguity, polysemy and lack of objective measures.

After the pioneer work of Rescorla and Wagner [1972] many other models have been created, many of them centered in Classical Conditioning, for example; Mackintosh [1975], Wagner [1981], Pearce [1980] or Dickinson and Burke [1996]. Most models take into account only one of the 4 types presented above, isolating one of them from the others [6]. From our point of view, this lack of integration is a big weakness of the current state of the art. Especially if we consider that one kind of learning can affect the functionality of the others. In the case of habituation, the last point is easily observed. Given that CC and Operant Conditioning have as their atomic elements the stimuli that are processed or paid attention to by the organism, and that habituation runs automatically any time a stimulus is presented, habituation can not be ignored in the explanation of CC an Operant Conditioning.

The phenomenon of *Latent Inhibition* illustrates this idea very clearly. Latent Inhibition appears in CC when the CS is pre-exposed (and therefore habituated) to the organism. This results in a CR's generation retard. Indeed, as an example of the interest of integrating habituation in CC or Operant Conditioning models, Schmajuk et al. [1996] made use of an habituation module[7] in their CC's model in order to fully reproduce Latent Inhibition.

In addition, each kind of learning comprehends a lot of particular phenomena, current models also fail to reproduce every phenomena within a given kind of learning. In short, the current models can not provide an integrated explanation of associative learning, as each model resolves just a particular set of phenomena.

2.4.1.3 Characteristics of habituation

Before building a model of habituation, we firstly have to identify the main properties of habituation that any model should reproduce. Following Groves and Thompson [1970] and Thompson and Spencer [1966], 6 characteristics define habituation in contrast to other response decrements;

[5] As formal model we mean a description of a real system's behavior expressed in a formal language, usually a mathematical formalism

[6] There exist three exceptions to this. The SOP model [Wagner, 1981, Brandon et al., 2003]), the SLG model [Schmajuk et al., 1996] and the work of Alonso et al. [2005b] and Alonso et al. [2005a]. All of them are treated in page 57

[7] The authors call this module *Novelty System*

1 Exponential decrease of the response's strength when the organism is exposed to the same stimulus repeatedly. The strength curve has the form of a negative exponential curve.

2 Spontaneous recovery of the response after a period without exposure to the stimulus.

3 More rapid habituation with stimuli of lower intensity.

4 More rapid habituation and more pronounced final level of habituation with shorter inter-stimulus interval (ISI).

5 Dishabituation of the stimuli previously habituated upon the introduction of a new stimulus.

6 Two time scales habituation, short and long. The former corresponds to the usual habituation process and vanishes typically within minutes, the latter lasts for hours or days and appears after a long and distributed exposure to the stimulus. In practice, long-term habituation produces that short-term habituation performs faster.

To which we can add another one;

7 The shorter the habituation ISI, the faster the recovery. As the rest of properties, it can be demonstrated in very different kind of animals; Carew et al. [1972], Byrne [1982] and Rankin and Broster [1992].

Given these main properties, we can use them as a quality criteria to asses any model. To finish the state of the art, the next section will give us a detailed review of the current models of habituation.

2.4.1.4 Review of current models

The first formal model of habituation appeared with Stanley [1976]. Table 2.4 compares the models that we can find in the literature since then[8]. The comparison is in terms of the model's capability to reproduce the main properties that we enumerated above.

In general, it is not difficult to model the first 4 basic properties. Most of the models cover the first 3 or 4 simple properties and concentrate their efforts in solving one of the last three. It seems that features 5, 6 and 7 are very hard to model, and even more difficult with the three of them together in a single model.

Finally, it is important to note that we are considering models that work at the psychological level. In other words, models that manipulate stimuli and responses and try to reproduce the phenomena that appears when studying those elements. Other models that are related to habituation, but at a level of neural mechanisms are not taken into account unless the variables and interpretations that concern those models can be extrapolated to a psychological level. An example of a model that is related to habituation but can not be easily interpreted in a behavioural framework is Lara [1983], which proposes a neural mechanism that could explain the way the hippocampus discriminate neural code and can, therefore, perform specific habituation.

[8]We consider models that only try to explain habituation. Other models that also reproduce some habituation properties, but are focused in Classical Conditioning, are described in the following sections

Property/Model	Stanley [1976]	Wang [1994]	Staddon and Higa [1996]	Alonso et al. [2005b]	del Rosal et al. [2006]
Response's exponential decrease	yes	yes	yes	yes	yes
Spontaneous recovery	yes	yes	yes	yes	yes
Lower intensity, faster habituation	yes	no	no	yes*	yes
Shorter ISI, faster habituation	?	yes*	yes	yes	yes
Dishabituation	no	no	no	no	yes
Long-term and short-term habituation	no	yes	no	no	no
Shorter ISI, faster recovery	no	no	yes	no	no

Table 2.4: *Habituation models comparison.* Symbols; a) no = The model can not reproduce it. b) yes = The model can reproduce it. c) yes* = Not explicitly demonstrated by the author but easily deducted from the model's details. d) ? = Claimed by the author but not explicitly demonstrated.

2.4.1.5 Other models related to habituation

Apart from the models we studied previously, there exists models of Classical Conditioning that treat habituation marginally and can, thus, satisfy some properties of those presented above. Despite this marginal treatment, these models can reproduce some characteristics of CC that are not easily cover by other models due to the inclusion of some habituation mechanisms. This supports a perspective under which the integration of low level learning or functionality into models of more complex learning is highly useful.

One of this models is the so-called SOP Wagner [1981], Brandon et al. [2003]. This very complete model of memory and conditioning is capable of performing short-term habituation, providing the ability of reproducing CS priming, a CC phenomenon by which a pre-exposure to a CS prior to a CS-US pairing produces a decrease in the CR's generation. The SOP model manages to reproduce properties 1, 2 and 4 of those listed above and can be considered a model that is very close to the already mentioned idea of integrating the different types of associative learning while modelling.

A second model with these characteristics is the SLG model [Schmajuk et al., 1996] which was mentioned above. As explained before, its habituation skills permit the explanation of the the Latent Inhibition phenomenon. Unfortunately, the SLG model has habituation as a secondary target, making its main efforts in CC. Thes results in a poor performance in habituation. It can only reproduce the first basic property, failing in other basic ones like recovery (2). Thus, concerning habituation, we can say that this model is irrelevant.

Finally, Alonso et al. [2005a] integrate its previous model of habituation [Alonso et al., 2005b] in a model of CC. They demonstrated how habituation could explain Spontaneous Recovery of the CR. This work is a conscious attempt of putting into practice the approach on which we have been commenting.

2.4.1.6 Summing up

During the last section we had the opportunity to confirm that none of the models in the current literature can be proud of reproducing all the main features of habituation. This situation leads to a lack of an integral model of habituation.

Such an integral model would provide a very interesting insight in the explanation of the processes that produce the phenomenon of habituation. In addition, as we discussed previously, if we want to create a general model of associative learning, including CC and Operant Conditioning, we need a fully featured model of habituation to do the filtering or attentional pre-processing tasks over the stimuli. Without this functionality our model of CC and Operant Conditioning would be necessarily incomplete.

2.4.2 Language Processing

Computational Linguistics researches linguistic phenomena that occur in digital data. Natural Language Processing (NLP) is a subfield of Computational Linguistics that focuses on building automatic systems able to interpret or generate information written in natural language [Volk, 2004]. This is a broad area which poses a number of challenges, both for theory and for applications.

Machine Translation was the first NLP application in the fifties [Weaver, 1955]. In general, the main problem found in all cases is the inherent ambiguity of the language [Mitkov, 2003].

A typical NLP system has to cover several linguistic levels:

- **Phonological**: Sound processing to detect expression units in speech.
- **Morphological**: Extracting information about words, such as their part of speech and morphological characteristics [Mikheev, 2002, Alfonseca, 2003]. The best systems have an accuracy of 97% in this level [Brants, 2000].
- **Syntactical**: Using parsers to detect valid structures in the sentences, usually in terms of a certain grammar. One of the most efficient algorithms is the one described by Earley and its derivatives [Earley, 1970, Seifert and Fischer, 2004, Zollmann and Venugopal, 2006]. It provides parsing in polynomial time, with respect to the length of the input (linear in the average case; n^2 and n^3, respectively, for unambiguous and ambiguous grammars in the worst case) Our work is focused on this step.
- **Semantic**: Finding the most suitable knowledge formalism to represent the meaning of the text.
- **Pragmatic**: Interpreting the meaning of the sentence in a context which makes it possibe to react accordingly.

The two last levels are still far from being solved [Gomez et al., 2008]. Typical NLP systems usually cover the linguistic levels previously described in the following way:

$$\Rightarrow \text{Morphologycal analysis} \Rightarrow \text{Syntax analysis} \Rightarrow \text{Semantic interpretation} \Rightarrow$$
$$\text{Discourse text processing} \Rightarrow \text{OCR/Tokenization}$$

A computational model that can be applied to NLP tasks is a network of evolutionary processors (NEPs). NEP as a generating device was first introduced in Csuhaj-Varju and Salomaa [1997] and Csuhaj-Varjú and Mitrana [2000] . The topic is further investigated in Castellanos et al. [2001], while further different variants of the generating machine are introduced and analyzed in Castellanos et al. [2005], Manea [2004b], Manea et al. [2007], Margenstern et al. [2005], Martin-Vide et al. [2003].

In Bel Enguix et al. [2009], a first attempt was made to apply NEPs for syntactic NLP parsing. Our work focuses on the same goal: testing the suitability of NEPs to tackle this task. We have previously mentioned some performance characteristics of one of the most popular families of NLP parsers (those based on Early's algorithm). We will conclude that our approach has a better complexity under the assumptions of the NEP model (see section 4.2). While Bel Enguix et al. [2009] outlines a bottom up approach to natural language parsing with NEPs, in our work we suggest a top-down strategy and show its possible use in a practical application.

We are not giving an introduction on classical syntactic parsing algorithms and their features because they all are knowledge of graduate studies in computer science. The important details concerning our work will be clarified in section 4.2 as we present our work. The interested reader can consult Aho et al. [1998] or Alfonseca Moreno et al. [2006].

Part II

Advances and applications of bio-inspired computing models

Chapter 3

Simulating and programming NEPs

3.1 jNEP

jNEP is a program written in Java which is capable of simulating almost any NEP in the literature. In order to be a valuable tool for the scientific community, it has been developed under the following principles:

a) It rigorously complies with the formal definitions found in the literature.

b) It serves as a general tool, by allowing the use of the different NEP variants and is ready to adapt to future possible variants, as the research in the area advances.

c) It exploits as much as possible the inherent parallel/distributed nature of NEPs.

The jNEP code is freely available in http://jnep.e-delrosal.net.

3.1.1 jNEP design

jNEP offers an implementation of NEPs as general, flexible and rigorous as described in the previous paragraphs. As shown in figure 3.1, the design of the NEP class mimics the NEP model definition. In *jNEP*, a NEP is composed of evolutionary processors and an underlying graph (attribute *edges*) which defines the net topology and guides the inter-processor interactions. The *NEP* class coordinates the main dynamic of the computation and rules the processors (instances of the *EvolutionaryProcessor* class), forcing them to perform alternate evolutionary and communication steps. It also stops the computation when needed. The core of the model includes these two classes, together with the *Word* class, which handles the manipulation of words and their symbols.

jNEP is flexible and adaptable to diferent NEP variants thanks to its design based on Java interfaces.

jNEP offers three interfaces:

a) *StoppingCondition*, which provides the method *stop* to determine whether a *NEP* object should stop according to its state.

b) *Filter*, whose method *applyFilter* determines which objects of class *Word* can pass it.

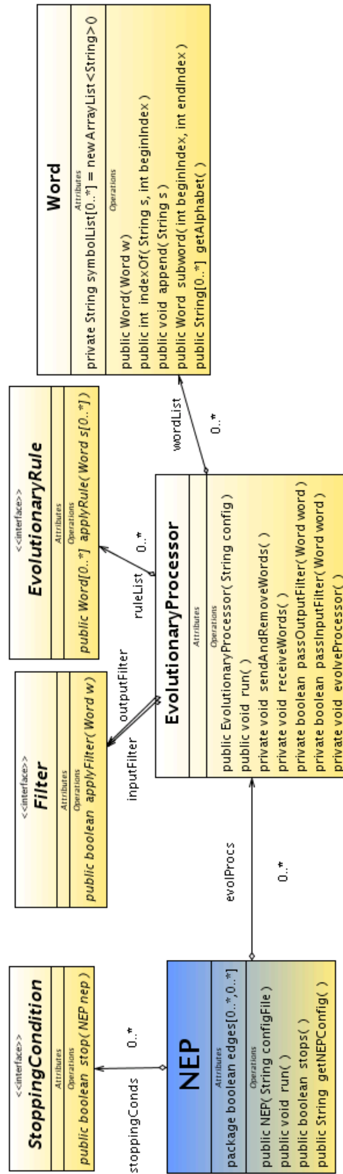


Figure 3.1: Simplified class diagram of jNEP

c) *EvolutionaryRule*, which applies a *Rule* to a set of *Word*s to get a new set.

jNEP tries to implement a wide set of NEP's variants. The *jNEP user guide* (http://jnep.e-delrosal.net) contains the updated list of filters, evolutionary rules and stopping conditions implemented.

Moreover, *jNEP* can widely exploit the parallel/distributed nature of NEPs in different ways. Firstly, the user can choose the parallel/distributed platform on which *jNEP* runs. At the moment, the supported platforms are standard JVM and parallel Java platforms. The Sun Java Virtual Machine, which can be considered the standard Java, cannot be run on massively parallel platforms like clusters. Several attempts have tried to overcome this limitation, for example: Java-Enabled Single-System-Image Computing Architecture 2 (JESSICA2) [JESSICA2, 2008], the cluster virtual machine for Java developed by IBM (IBM cJVM) [IBM, 2000], Proactive PDC [Inria Sophia Antipolis, 2008], DO! [Launay and Pazat, 1997], JavaParty [JavaParty, 2008], and Jcluster [Zhang and Zheng, 2006]. The simulator described in this section has been developed with both JVM and JavaParty. Secondly, concurrency is implemented by means of two different Java approaches: *Thread*s and *Process*es. The first needs more complex synchronization mechanisms. The second uses heavier concurrent threads.

More precisely, in the case of the *Process*es option each processor in the net is actually an independent program in the operating system. The communication between nodes is carried out through the standard input/output streams of the program. The class NEP has access to those streams and coordinates the nodes. The mandatory alternation of communication and evolutionary steps in the computations of NEPs greatly eases their synchronization and communication. The following protocol has been followed for the communication step:

1 NEP class sends a message to every node in the graph asking to start the communication step. Then it waits for their responses.

2 Every node finishes its communication step after sending to the net the words that pass their outputs filters. Then, they indicate to the NEP class that they have finished the communication step.

3 The NEP class moves all the words from the net to the input filters of the corresponding nodes.

The evolutionary step is synchronized by means of an initial message sent by the NEP class to make all the nodes evolve. Afterwards, the NEP class waits until all the nodes finish.

The implementation with *Java Thread*s has other implications. In this option, each processor is an object of the *Java Thread* class. Thus, each processor executes its tasks in parallel as independent execution lines, although all of them belong to the same program. Data exchange between them is performed by direct access to memory. The principles of communication and coordination are the same as in the previous option. The main difference is that instead of waiting for all the streams to finish or to send a certain message, *Thread*s are coordinated by means of basic concurrent programming mechanisms as semaphores, monitors, etc.

In conclusion, jNEP is a very flexible tool that can run in many different environments. Depending on the operating system, the Java Virtual Machine used and the concurrency option chosen, jNEP will work in a slightly different manner. The user should select the best combination for his needs.

3.1.2 jNEP in practice

jNEP is written in Java, therefore to run jNEP one needs a Java virtual machine (version 1.4.2 or above) installed in a computer. Then one has to write a configuration file describing the NEP. The *jNEP user guide* (available at http://jnep.e-delrosal.net) contains the details concerning the commands and requirements needed to launch jNEP. In this section, we want to focus on the configuration file which has to be written before running the program, since it has some complex aspects which are important to be aware of the potentials and possibilities of jNEP.

The configuration file is an XML file specifying all the features of the NEP. Its syntax is described below in BNF format, together with a few explanations. Since BNF grammars are not capable of expressing context-dependent aspects, context-dependent features are not described here. Most of them have been explained informally in the previous sections. Note that the traditional characters <> used to identify non-terminals in BNF have been replaced by [] to avoid confusion with the use of the <> characters in the XML format.

- [configFile] ::= <?xml version="1.0"?> <NEP nodes="[integer]"> [alphabetTag] [graphTag] [processorsTag] [stoppingConditionsTag] </NEP>
- [alphabetTag] ::= <ALPHABET symbols="[symbolList]"/>
- [graphTag] ::= <GRAPH> [edge] </GRAPH>
- [edge] ::= <EDGE vertex1="[integer]" vertex2="[integer]"/> [edge]
- [edge] ::= λ
- [processorsTag] ::= <EVOLUTIONARY_PROCESSORS> [nodeTag] </EVOLUTIONARY_PROCESSORS>

The above rules show the main structure of the NEP: the alphabet, the graph (specified through its vertices) and the processors. It is worth remembering that each processor is identified implicitly by its position in the processors tag (first one is number 0, second is number 1, and so on).

- [stoppingConditionsTag] ::= <STOPPING_CONDITION> [conditionTag] </STOPPING_CONDITION>
- [conditionTag] ::= <CONDITION type="MaximumStepsStoppingCondition" maximum="[integer]"/> [conditionTag]
- [conditionTag] ::= <CONDITION type="WordsDisappearStoppingCondition" words="[wordList]"/> [conditionTag]
- [conditionTag] ::= <CONDITION type="ConsecutiveConfigStoppingCondition"/> [conditionTag]
- [conditionTag] ::= <CONDITION type="NonEmptyNodeStoppingCondition" nodeID="[integer]"/> [conditionTag]
- [conditionTag] ::= λ

The syntax of the stopping conditions shows that a NEP can have several stopping conditions. The first one which is met causes the NEP to stop. The different types try to cover most of the stopping conditions used in the literature. If needed, more of them can be added to the system easily.

At this moment jNEP supports 4 stopping conditions, the *jNEP user guide* explains their semantics in detail:

1. **ConsecutiveConfigStoppingCondition**: It produces the NEP to stop if two consecutive configurations are found as communication and evolutionary steps are performed.

2. **MaximumStepsStoppingCondition**: It produces the NEP to stop after a maximum number of steps.

3. **WordsDisappearStoppingCondition**: It produces the NEP to stop if none of the words specified are in the NEP. It is useful for generative NEPs where the lack of non-terminals means that the computation has reached its goal.

4. **NonEmptyNodeStoppingCondition**: It produces the NEP to stop if one of the nodes is non-empty. Useful for NEPs with an output node.

- [nodeTag] ::= <NODE initCond=”[wordList]” [auxWordList]> [evolutionaryRulesTag] [nodeFiltersTag] </NODE> [nodeTag]
- [nodeTag] ::= λ
- [auxWordList] ::= auxiliaryWords=”[wordList]” | λ
- [evolutionaryRulesTag] ::= <EVOLUTIONARY_RULES> [ruleTag] </EVOLUTIONARY_RULES>
- [ruleTag] ::= <RULE ruleType=“[ruleType]” actionType=“[actionType]” symbol=“[symbol]” newSymbol=“[symbol]”/> [ruleTag]
- [ruleTag] ::= <RULE ruleType=”splicing” wordX=“[symbolList]” wordY=“[symbolList]” wordU=“[symbolList]” wordV=“[symbolList]”/> [ruleTag]
- [ruleTag] ::= <RULE ruleType=”splicingChoudhary” wordX=“[symbolList]” wordY=“[symbolList]” wordU=“[symbolList]” wordV=“[symbolList]”/> [ruleTag]
- [ruleTag] ::= λ
- [ruleType] ::= insertion | deletion | substitution
- [actionType] ::= LEFT | RIGHT | ANY

[nodeFiltersTag] ::= <FILTERS>[inputFilterTag] [outputFilterTag]</FILTERS>

[nodeFiltersTag] ::= <FILTERS>[inputFilterTag]</FILTERS>

[nodeFiltersTag] ::= <FILTERS>[outputFilterTag]</FILTERS>

[nodeFiltersTag] ::= <FILTERS></FILTERS>

- [inputFilterTag] ::= <INPUT [filterSpec]/>
- [outputFilterTag] ::= <OUTPUT [filterSpec]/>
- [filterSpec] ::= type=[filterType] permittingContext=“[symbolList]” forbiddingContext=“[symbolList]”
- [filterSpec] ::= type=“SetMembershipFilter” wordSet=“[wordList]”
- [filterSpec] ::= type=“RegularLangMembershipFilter” regularExpression=“[regExpression]”
- [filterType] ::= 1 | 2 | 3 | 4

Above, we describe the elements of the processors: their initial conditions, rules, and filters. jNEP treats rules with the same philosophy as in the case of stopping conditions, which means that our system supports almost all kinds found in the literature at the moment and, more important, future types can also be added via Java Interfaces easily.

jNEP can work with any of the rules found in the original model [Castellanos et al., 2003, Martin-Vide et al., 2003, Castellanos et al., 2001]. Moreover, it supports splicing rules, which are needed to simulate a derivation of the original model presented in Choudhary and Krithivasan [2005] and Manea et al. [2007]. The two splicing rule types are slightly different. It is important to note that if you use Manea’s splicing rules, you may need to create an auxiliary word set for those processor with splicing rules.

With respect to filters, jNEP is prepared to simulate nodes with filters based on random context conditions. To be more specific, any of the four filter types traditionally used in the literature since Martín-Vide and Mitrana [2005]. In addition, jNEP is capable of creating filters based in membership conditions. A few works use them, for instance Castellanos et al. [2003]. They are in some way non-standard and could be defined as follows:

1. **SetMembershipFilter**: It permits only words that are included in a specific set to pass.

2. **RegularLangMembershipFilter**: This filter contains a regular language to which words need to belong. The language have to be defined as a Java regular expression.

We will finish the explanation of the grammar for our xml files with the rules needed to describe some of the pending non-terminals. They are typical constructs for lists of words, list of symbols, boolean and integer data and regular expressions.

- [wordList] ::= [symbolList] [wordList]
- [wordList] ::= λ
- [symbolList] ::= `a string of symbols separated by the character '_'`
- [boolean] ::= true | false
- [integer] ::= `an integer number`
- [regExpression] ::= `a Java regular expression`

The reader may refer to the *jNEP user guide* for further detailed information.

3.1.2.1 An example

As an example, we present a very simple configuration file below. The NEP described below is very simple, there are two nodes connected where the first one deletes the symbol B and the other one inserts it. The initial word *A_B* travels from one node to the other suffering the deletion and insertion of the symbol *B*. The NEP stops after eight steps.

```
<?xml version="1.0"?>

<!-- NEP Config file-->
<!-- Character '_' is reserved since it is used to separate symbols within words
or within
a set of symbols-->

<NEP nodes="2">

  <ALPHABET symbols="A_B"/>

  <GRAPH>
    <EDGE vertex1="0" vertex2="1"/>
  </GRAPH>

  <EVOLUTIONARY_PROCESSORS>
    <NODE initCond="A_B">
      <EVOLUTIONARY_RULES>
        <RULE ruleType="deletion" actionType="RIGHT" symbol="B" newSymbol=""/>
      </EVOLUTIONARY_RULES>
      <FILTERS>
        <INPUT type="2" permittingContext="A_B" forbiddingContext=""/>
        <OUTPUT type="2" permittingContext="A_B" forbiddingContext=""/>
      </FILTERS>
    </NODE>
    <NODE initCond="">
      <EVOLUTIONARY_RULES>
        <RULE ruleType="insertion" actionType="RIGHT" symbol="B" newSymbol=""/>
      </EVOLUTIONARY_RULES>
      <FILTERS>
        <INPUT type="2" permittingContext="A_B" forbiddingContext=""/>
        <OUTPUT type="2" permittingContext="A_B" forbiddingContext=""/>
      </FILTERS>
    </NODE>
  </EVOLUTIONARY_PROCESSORS>
```

```
  <STOPPING_CONDITION>
    <CONDITION type="MaximumStepsStoppingCondition" maximum="8"/>
  </STOPPING_CONDITION>
</NEP>
```

The corresponding output for this NEP configuration is the following:

```
XML CONFIGURATION FILE LOADED AND PARSED SUCCESSFULLY...

GRAPH INFO PARSED SUCCESSFULLY...

STOPPING CONDITIONS INFO PARSED SUCCESSFULLY...

EVOLUTIONARY PROCESSORS INFO PARSED SUCCESSFULLY...

NEP RUNNING...


***************                NEP INITIAL CONFIGURATION               ***************
        --- Evolutionary Processor 0 ---

A_B

        --- Evolutionary Processor 1 ---



*************** NEP CONFIGURATION - EVOLUTIONARY STEP - TOTAL STEPS: 1 ***************
        --- Evolutionary Processor 0 ---

A

        --- Evolutionary Processor 1 ---



*************** NEP CONFIGURATION - COMMUNICATION STEP - TOTAL STEPS:  2 ***************
        --- Evolutionary Processor 0 ---


        --- Evolutionary Processor 1 ---
A

*************** NEP CONFIGURATION - EVOLUTIONARY STEP - TOTAL STEPS: 3 ***************
        --- Evolutionary Processor 0 ---


        --- Evolutionary Processor 1 ---
A_B

*************** NEP CONFIGURATION - COMMUNICATION STEP - TOTAL STEPS:  4 ***************
        --- Evolutionary Processor 0 ---

A_B

        --- Evolutionary Processor 1 ---
```

```
*************** NEP CONFIGURATION - EVOLUTIONARY STEP - TOTAL STEPS: 5 ***************
        --- Evolutionary Processor 0 ---

A


        --- Evolutionary Processor 1 ---



*************** NEP CONFIGURATION - COMMUNICATION STEP - TOTAL STEPS:  6 ***************
        --- Evolutionary Processor 0 ---


        --- Evolutionary Processor 1 ---
A

*************** NEP CONFIGURATION - EVOLUTIONARY STEP - TOTAL STEPS: 7 ***************
        --- Evolutionary Processor 0 ---


        --- Evolutionary Processor 1 ---
A_B

*************** NEP CONFIGURATION - COMMUNICATION STEP - TOTAL STEPS:  8 ***************
        --- Evolutionary Processor 0 ---

A_B


        --- Evolutionary Processor 1 ---



----------------------- NEP has stopped!!! -----------------------

Stopping condition found:
net.e_delrosal.jnep.stopping.MaximumStepsStoppingCondition

------------------------------------------------------------------

We are glad you used jNEP

Emilio del Rosal
e-mail: emilio.delrosal <at> uam.es
web: www.e-delrosal.net
```

The reader may refer to the *jNEP user guide* for further detailed information.

3.1.3 jNEPView

jNEP, a Network of Evolutionary Processors (NEP) simulator, has been improved with several visualization facilities. jNEPView display the network topology in a friendly manner and shows the complete description of the simulation state in each step. Using this tool, it is easier to program and study NEPs, whose dynamic is quite complex, facilitating theoretical and practical advances on the NEP model.

jNEP has been modified to produce a sequence of log files, one for each simulation step. This sequence of files will be read by jNEPView to show the successive configurations of the NEP. These logs are in a very simple format that contains a line for each processor in the same implicit order in which they appear in the configuration file. Each line contains the strings of the corresponding processor. This little extension of jNEP makes it simple to follow the trace of the simulation and manage it.

3.1.3.1 jNEPView design

To handle and visualize graphs, we have used JGraphT [JGraphT, 2009] and JGraph [JGraph, 2009] which are free Java libraries under the terms of the GNU Lesser General Public License.

JGraphT provides mathematical graph-theory objects and algorithms. It is used by jNEPView to represent formally the NEP underlying graph. Fortunately, JGraphT also allows to display its graphs using the JGraph library, which is a graph visualization library with a lot of utilities.

We use those libraries to show the NEP topology. Once jNEPView is started, a window shows the NEP layout as clear as possible. We have decided to set the nodes in a circle, but the user can freely move each component. In this way, it is easier to interpret the NEP and study its dynamics.

Moreover, several action buttons have been placed to study the NEP state and progress. If the user clicks on a node, a window is open where the words of the node appear. In order to control the simulation development, the user can move throughout the simulation and the contents of the selected nodes are updated in their corresponding windows in a synchronized way.

Before running jNEPView, jNEP should have actually finished the simulation. In this way, jNEPView just reads the jNEP state logs and the user can jump from one simulation step to another quickly, without worrying about the simulation execution times.

3.1.3.2 jNEPView example

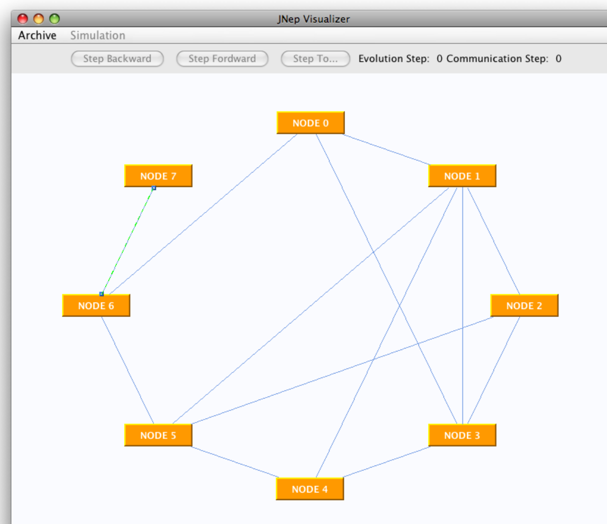


Figure 3.2: Window that shows the layout of the simulated NEP

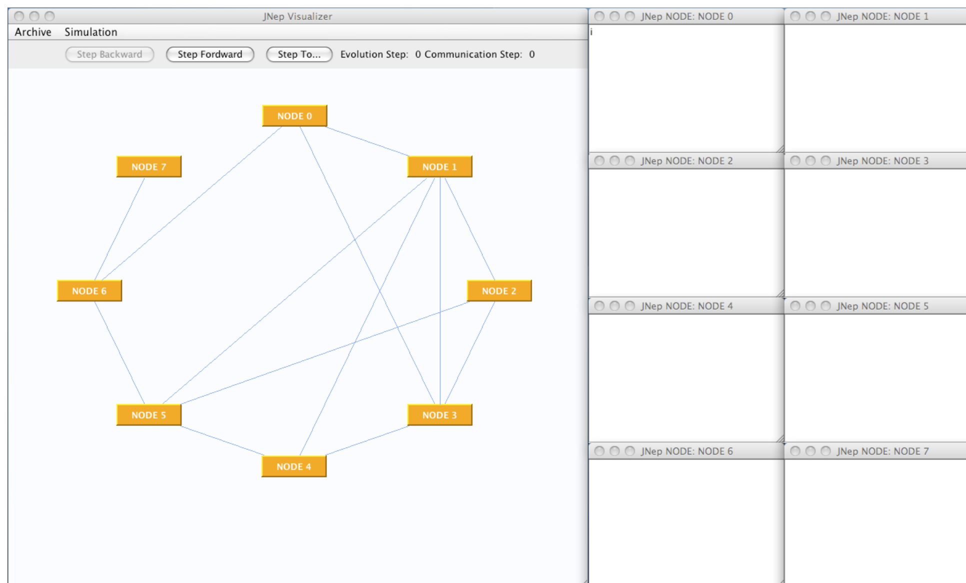


Figure 3.3: Initial simulation step

This section describes how jNEPView shows the execution of a NEP solving a particular case of the Hamiltonian path in an undirected graph. This NEP was presented in del Rosal et al. [2009b] and is detailed in section B. Its configuration file is delivered with the jNEP package. Although this NEP is quite complex and should be studied later, we can illustrate the features of jNEPView following its computation.

Firstly, the user has to select the configuration file for jNEP which defines the NEP to simulate. After that, the layout of the NEP is shown like in figure 3.2.

At this point, the buttons placed in the main window to handle the simulation are activated and the user can select the nodes whose content it wants to inspect during the simulation. In addition, the program allows the user to move throughout the simulation timeline by stepping forward and backward. Figures from 3.3 to 3.6 display the contents of all the nodes in the NEP in four different moments: the first three steps and the final one. The user can also jump to a given simulation step by clicking on the appropriate button.

3.1.4 A Visual Language for Modelling and Simulation of NEPs

The goal of this work is to provide the jNEP user with a visual tool to graphically design the NEPs under consideration. The author and collegues have designed a domain specific visual language for NEPs by means of AToM^3 [Jimenez et al., 2010]. We have also taken advantage of the AToM^3's graph grammar modules to automate some mechanical and time-consuming designing tasks, such as properly placing filters close to their processors, and defining some kinds of standard graph topologies. AToM^3 is a python tool previously developed by some of the authors of [Jimenez et al., 2010].

3.1.4.1 Introduction to AToM^3

Visual Languages play a central role in many computer science activities. For example, in software engineering, diagrams are widely used in most of the phases of software construction. They provide intuitive and powerful domain-specific constructs

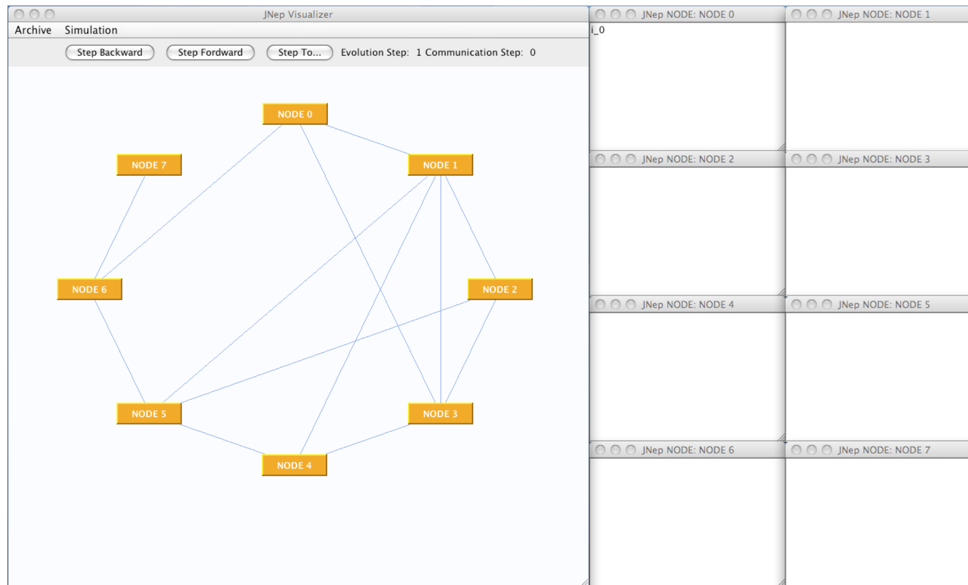


Figure 3.4: Next simulation step

and allow the abstraction from low-level, *accidental* details, enabling reasoning and improving understandability and maintenance. The term Domain Specific Visual Language (DSVL) [Kelly and Tolvanen, 2008] refers to languages that are specially oriented to a certain domain, limited but extremely efficient for the task to be performed. DSVLs are extensively used in Model Driven Development, one of the current approaches to Software Engineering. In this way, engineers no longer have to resort to low-level languages and programming, but are able to synthesize code for the final application from high-level, visual models. This increases productivity, quality, and permits its use by non-programmers.

The design of a DSVL involves defining its concepts and the relations between them. This is called the abstract syntax and is usually defined through a meta-model. Meta-models are normally described through UML class diagrams. Hence, the language spawned by the meta-model is the (possibly infinite) set of models conformant to it. In addition, a DSVL needs to be provided with a concrete syntax. That is, a visualization of the concepts defined in the meta-model. In the most simple case, the concrete syntax just assigns icons to meta-model classes and arrows to associations. The description of the abstract and concrete syntax is enough to generate a graphical modelling environment for the DSVL. Many tools are available that automate such tasks, in this work we use AToM^3 [Zollmann and Venugopal, 2006].

In many scenarios, the description of the DSVL syntax is not enough, but one would like to define manipulations that "*breath life*" into such models. For example, one could like to animate or simulate the models, to define "macros" for complex editing commands, or build code generators for further processing by other tools. As models and meta-models can be described as attributed, typed graphs, they can be visually manipulated by means of graph transformation techniques [Ehrig et al., 2006]. This is a declarative, visual and formal approach to manipulate graphs. Its formal basis, developed in the last 30 years, makes it possible to demonstrate properties of the transformations. A graph grammar is made up of a set of rules and a starting graph. Graph grammar rules are made up of a left and a right hand side (LHS and RHS), each one having graphs. When applying a rule to a graph (called host graph), an occurrence of the LHS should be found in the graph, and

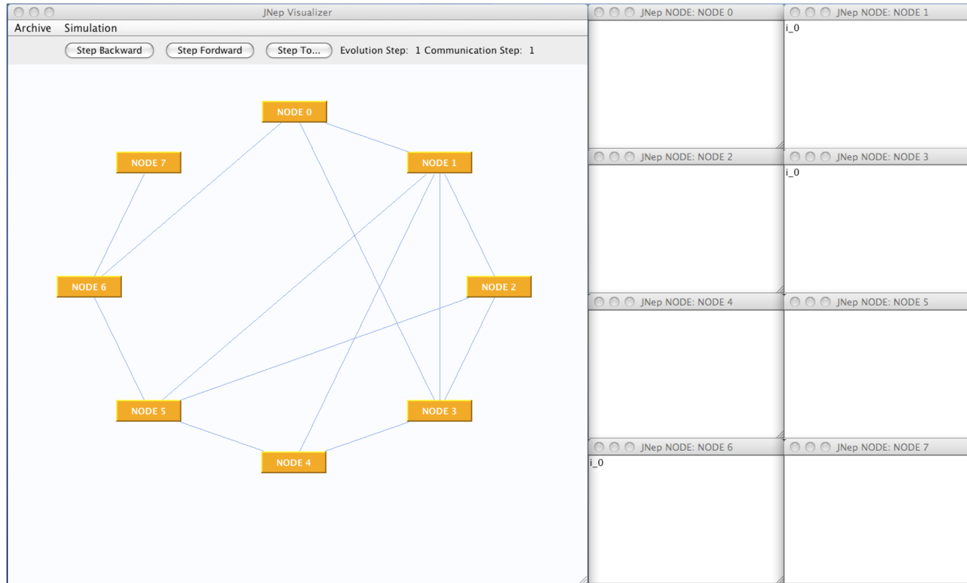


Figure 3.5: Second simulation step

then it can be replaced by the RHS.

In this work, we apply the aforementioned concepts to build a DSVL to design Networks of Evolving Provessors (NEPs). For this purpose, we built a meta-model in the AToM^3 tool and a graphical modelling environment was automatically generated. Then, such environment was enriched by providing rules to automate complex editing commands, and by a code generator to synthesize code for jNEPs, in order to perform simulations. The approach has the advantage that the final user does not need to be proficient in the jNEPs textual input language, but he can model and simulate NEPs visually.

3.1.4.2 NEPs visual language

The system consists of four parts: two are core and essential parts of it (meta-model, which gives the elements that will be used to build models; and the code generator, which is a program that creates the code to be used by the simulator), and the other two are just helpful in giving usability (constraints, which are defined inside the meta-model and control some aspects of the models to ensure that they are syntactically correct; graph grammars are used to generate parts of the models crated that might be dull to be done manually).

This way the user perceives that only a few buttons, GUI elements, and common actions are needed to build a model that can be executed by the simulator, not being aware of the complexity of the code used to feed it.

From this point the parts which compose the system are described.

Fig. 3.7 presents the UML class diagram of the meta-model that represents the NEPs domain for the simulator. We can see several classes for the usual elements of a NEPs system in it: alphabet, processors, filters, rules, and stopping conditions. We need the following subclasses:

- rule → inserting, deleting, deriving, substituting, and regular_expression
- stopping_condition → consecutive_config, maximum_steps, words_disappear, and non_empty_node

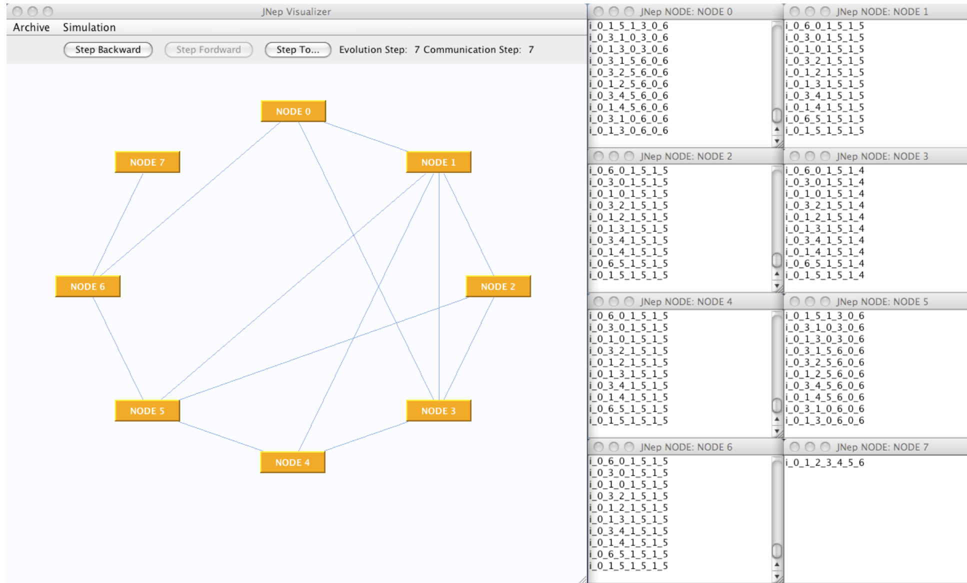


Figure 3.6: End of simulation

Code generator is the other core part of the system. It is responsible for creating the XML file that will feed the simulator (using Python code). In order to do this, it performs a couple of tasks: first of all it checks some properties that the model must meet, and then it goes through the instances that compose the model created by the user in order to generate parts of the XML.

In the first task the properties that are checked are the following: an alphabet must exist, there must be one stopping condition, there cannot be symbols in the model which are not in the alphabet, and there is not more than one connection from one processor to another one. Some of these properties are checked in the time of model creation by using constraints, but we check them here to increase reliability.

After checking that the model is valid, the second task is executed, where a loop over the instances is performed and, using their connections and attributes values, the XML file is written.

Graph grammars formal tool is used here to maximize the usability and make model design less dull and repetitive to the user.

The problems solved by means of this tool are the automatic assignment of input and output filters to processors, and automatic creation of connections among processors in order to get a fully connected graph.

Even being related to the final user's point of view, the correspondence among classes (non abstract) and graphical elements is an important part of the design phase. These direct relations are: alphabet → big rectangle, processor → small rectangle, filter → triangles, rule's subclasses → ovals, stopping_condition's subclasses → A text containing the name of the type of stopping condition

All of them show its attributes' values as well.

3.1.4.3 User point of view - How domain experts use it

The way of making models is very simple: using GUI elements (like buttons) and dropping entities on a canvas. Also other well-known elements like combo-boxes or text input areas might be used.

Then, we can imagine that AToM^3 interface is quite intuitive. In Fig. 3.8,

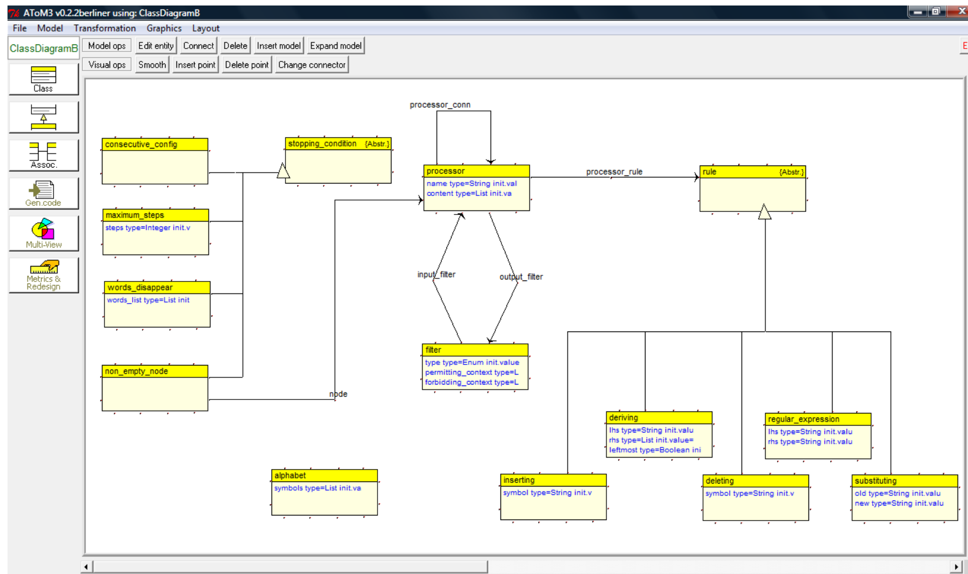


Figure 3.7: The meta-model UML class diagram

a sample of this interface for defining NEPs is shown. We can observe in it the graphical representations of the classes that compose the system.

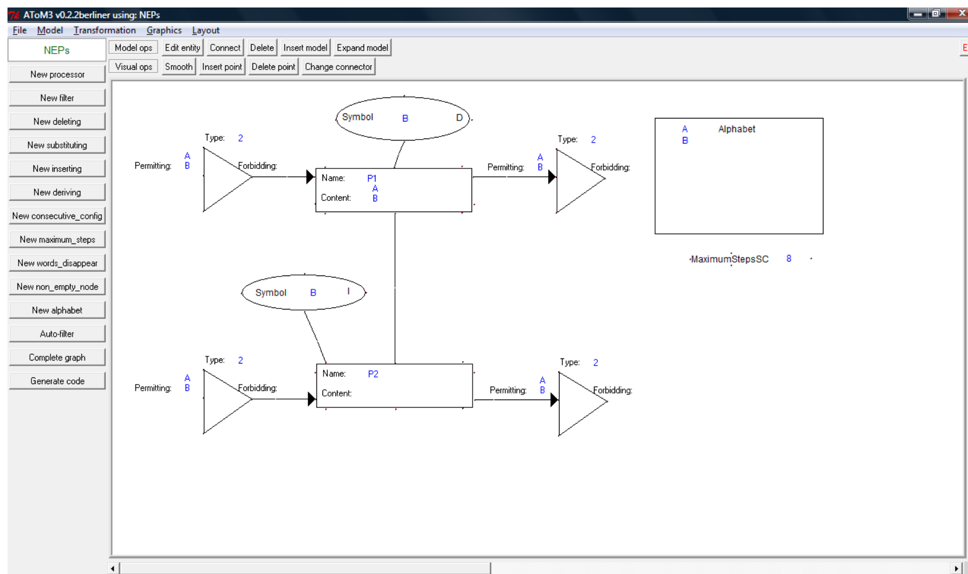


Figure 3.8: The visual language in action.

It seems obvious in Fig. 3.8 that the complexity of writing the XML is quite higher than building the model just using the mouse with AToM^3. This difference is even bigger when the end user is not used to write programs, and this is one of the system's biggest advantages together with the ability the user gets on seeing the model just at one sight (there is no need to inspect the code).

3.1.5 NEPs-Lingua

The previous tools all together forms a development environment for programming NEPs that includes a Java NEP simulator (jNEP), a Java graphical viewer of its simulations (jNEPview), and a domain specific visual language for NEPs designed with AToM^3 (NEPVL). In addition, the author and colleagues have developed a high level textual programming language for NEPs [de la Cruz et al., 2011]. We will briefly introduced this language called NEPs-Lingua along this section. NEPs-Lingua is the first textual programming language for NEPs. The main reference of this word is P-Lingua (García-Quismondo et al. [2009] and http://www.p-lingua.org); a textual programming language for P-Systems (see section 2.1.2.3). It is a first step to extending the P-Lingua approach to other bio-inspired models of computation. The final objective is to provide the researchers with a homogeneous family of languages for programming natural computers. This is the reason why NEPs-Lingua is designed to be similar to P-Lingua. NEPs-Lingua has two main goals: 1) Like P-Lingua, it aims to provide the researchers with a syntax as close as possible to the one used to describe NEPs in the literature. 2) It tries to ease some usually boring, mechanical and time-consuming tasks needed to describe NEPs with the input formalisms of the available tools.

3.1.5.1 The NEPs-Lingua syntax

In the following paragraphs we describe, mainly by examples, the syntax of NEPs-Lingua. A full ANTLR [1] description of the complete grammar may be asked from the authors [de la Cruz et al., 2011]. The main components of a NEPs-Lingua program are atomic data, comments, nodes, the alphabet, the initial contents of the nodes, evolutionary rules, filters, the connections of the NEP graph and stopping conditions.

Atoms There are two classes of atomic data: alfanumeric strings of symbols (they have to start with an alphabetic character); and integer arithmetic expressions, with the usual mathematical notation, that include the operators in the set $\{\wedge(power), +, -, *, /\}$

Comments The typical C++ comments are also available in NEPs-Lingua.

Line comments For example `// Comment`.
The comment includes every symbol until the end of the line.

Multi line comments For example

```
/*   ... Comment
 ...          */
```

Where the comment includes everything (even the *end of line* markers) between the symbols "/*" and "*/".

Alphabet It is the alphabet of the NEP, a set of strings of symbols. The expression `@A={X,S,a,b,o,O}` defines an alphabet that contains the elements "X", "S", "a", "b", "O", and "o".

[1] ANTLR is a Java tool to design top-down parsers and language processors, developed by Terence Par. Further information can be found at http://www.antlr.org/

Nodes This is the most complex type of NEPs-Lingua data. There are two classes of nodes: with and without indexes. There are two kinds of indexes: numeric (defined by a range) and symbolic (defined by a set of strings of symbols). The syntax of indexes with numeric ranges is borrowed from P-Lingua.

Non indexed nodes The expression `{initial, final}` defines two nodes without indexes with names *initial* and *final*.

Indexed nodes The example defines a family of nodes with two indexes. One of them (i) takes its values from the interval $[0, 10]$. The values of the other (j) are taken from the set $\{o, a, b\}$.

```
{m{i,j}: 0<=i<=10, j->{o,a,b}}
```

The explicit set of the 33 defined nodes is $\{m_{0,a}, m_{0,b}, m_{0,c}, \dots m_{10,a}, m_{10,b}, m_{10,c}\}$.

Different kinds of nodes can be mixed by means of the union operator. The next example shows the definition of a set of nodes that contains the two previous examples.

```
@N={initial, final}+{m{i,j}: 0<=i<=10, j->{o,a,b}}
```

Initial content It describes the set of strings that a given node initially contains. Notice that the node is written as a parameter of the *content directive* `@c`. The expression `@c{n{X}}` = `{X, S}` sets the initial content of the node n_X to $\{X, S\}$

Rules Each type of rule has a different notation. Notice that, as in P-Lingua, the symbol `#` stands for the empty string and the string `-->` separates the left and right sides of the rule. The sentences `#-->a`, `a-->#` and `S-->aSb` are examples of respectively insertion, deletion, and substitution (or deriving) rules.

All the rules for a given node are given together in the same sentence. The sentence `@r{n{S}}` = `{S-->aSb, S-->ab}` assigns two deriving rules to the node n_S.

Filters Each processor needs an input and an output filter. Different papers previously mentioned define three components in the filters: their type and the permitting and forbidding contexts. We have grouped the different filters of the literature in six types (depending on the way in which they are applied): types from 1 to 4 and filters defined by means of regular expressions or by means of sets of strings. Both contexts are just sets of symbols described by means of regular patterns or explicit sets of strings. The following examples define several filters:

```
@pif{n{S}}= {1, {abc, oo}}
@fof{initial} = {@regular_pattern, ( ((a[]b)+) ][ (c*) )][ # }
@pif{n{2,a}}= {@set, {a,ab,aabb}}
```

where `@pif` and `@fof` stand respectively for permitting input and forbidding ouput filter (the same for forbidding input and permitting output filters). In regular expressions `[], ][, +, *, #` represent intersection, union, + and *, and the empty string.

Connections This element makes it possible to get a compact representation of NEPs. There are two ways of defining connections: the directive `@complete`, that stands for a complete graph; and an explicit set of connections defined by means of pairs of nodes. The next examples show both options:

```
@C=@complete
@C={ (final,n{X}), (n{X},m{9,a}) }
```

Stopping conditions The stopping conditions are written in a set after the directive `@S`. Each kind of condition is represented by its name and its required parameters. Both names and parameters are easy to identify in the following example:

```
@S={@no_change, @max_steps = 3+4, @non_emtpy_node={n{O}, n{X}} }
```

where `@no_change` stands for two consecutive equal configurations; `@max_steps` requires an expression to define the number of steps (the NEP stops after taking the given number of steps); and `@non_empty_node` includes a set of nodes whose contents are initially empty (the NEP stops when one of these nodes receives some string).

3.1.5.2 Examples

In this section we will show some complete NEPs-Lingua programs. Our main goal is to highlight the two main characteristics of NEPs-Lingua: reducing the size and keeping close to the formal notation.

Reducing the size of the representations First we show the NEPs-Lingua program for the example with two processors described in section 3.1.2.1. We can appreciate with this simple example that the NEPs-Lingua program is more compact than the other two representations.

```
@A={A,B}
@N={ n{i}: 0 <= i <= 1}
@c{n{0}}={A,B}
@r{n{0}}={B-->#}
@r{n{1}}={#-->B}
@S={@max_steps = 8 }
@C={@complete}
```

The reduction in size is greater as the complexity of the NEP increases. Especially when we need a complete graph because the XML configuration file in that case is forced to explicitly contain all the nodes and connections while the NEPs-Lingua source has to contain just two sentences.

Keeping NEPs-Lingua as close as possible to the formal notation used in the literature Section 4.2 contains another example: a NEP associated with the context free grammar for axiom X with the derivation rules $\{X \to SO, S \to aSb, S \to ab, O \to o, O \to oO, O \to Oo\}$. It is easy to see that the following NEPs-Lingua program for this NEP is quite similar to its formal definition.

```
@A={X,S,a,b,o,O} // Alphabet
@N=  {final} + {n{symbol}:symbol->{X,S,O}} /* Nodes associated with
     non terminal symbols
*/
@c{n{X}}={X}  // Initial content of the axiom node
@r{n{X}}= {X-->SO} // Deriving rules for the axiom
@r{n{S}}= {S-->aSb, S-->ab}
@r{n{O}}= {O-->o, O-->oO, O-->Oo}
@C=@complete  // The graph is complete
@S={ @non_emtpy_node={final} } // Stopping conditions
```

3.1.5.3 NEPs Lingua semantics

The semantic constraints that every NEPs-Lingua program has to satisfy are outlined below:

- It has to contain exactly one alphabet and one set of node declarations.
- It needs, at most, one of the following elements:
  - Connection declaration set. By default, the graph is considered complete.
  - Set of stopping conditions. `@no_change` is assumed by default.
- Filters, rules and initial contents are optional.
- Nodes have to be defined before their use.
- Each symbol representing rules, filters and initial contents has to be included in the alphabet.

NEPs-Lingua compilers should ensure these conditions. The usual way of controlling the last one is by means of a symbol table that is filled while processing the declaration sentences and is consulted by the sentences that use nodes and symbols. We have used different `Hashtable` Java objects to check these constrains. The following example shows some semantic mistakes:

```
@A={A}
@N={ n{i}: 0 <= j <= 1}
@c{n{0}}={A,B}
@r{n{0}}={B-->#}
@r{n{2}}={#-->B}
@S={@max_steps = 8 }
@C={@complete}
```

- The third, fourth and fifth lines contain the symbol `B`, which is not in the alphabet.
- The second line defines the index `j`, while the declared one is `i`
- The fifth line defines the rules for the node n_2, but the value for index (2) is invalid

3.1.6 Final comments

jNEP is one of the first and more complete implementations of the family of abstract computing devices called NEPs. *jNEP* simulates not only the basic model, but also most of its variants, and is able to run on parallel Java platforms, specially JavaParty [2008]. It is worth noticing that a version of jNEP adapted to *ANSI C++* has been developed to be run on clusters based on the well-known *Message Passing Interface*[2] libraries [Navarrete Navarrete et al., 2011].

With this work we have completed a framework to simulate NEPs with jNEP. The complete software platform includes the following modules:

1. jNEP, a Java NEPs simulator.
2. jNEPView, a Java graphical viewer of the simulations run by jNEP
3. An AToM^3 domain specific visual language for designing NEPs.

[2] http://www.dmoz.org/Computers/Parallel_Computing/Programming/Libraries/MPI/

4. An under-development high level programming language; NEPs-Lingua.

We would like to remark that NEP performance is improved thanks to the following features. They can not be implemented in a conventional von Neumann computer or even a cluster of them. At the moment, they are just simulated:

- Each processor contains as many copies as it needs of its strings without any additional restriction.
- All these words are simultaneously modified by the rules of the processors in the same step.
- All the processors in the net perform their steps simultaneously, that is, all the communication steps are done at the same time as well as all the evolutionary steps.

The above features convert NEPs in a promising model to solve NP problems efficiently. In other words, the model can solve NP problems in polynomial time, while a procedure has not been yet found for deterministic Turing machines so efficient and it is plausible that it does not exist at all. Remember that NP problems are the set of decision problems whose yes-instances can be identified and verified in polynomial time by a non-deterministic Turing machine (note that NP stands for non-deterministic polynomial). The process can be described as follows: in the first stage every possible solution is generated in a non-deterministic way, while during the second and final stages the solutions are verified in a deterministic way. In NP problems, the number of possible solutions increases at a rate greater than polynomial, however non-deterministic functioning allows to generate them in a polynomial time. The above features permit to NEPs emulate a similar procedure. Thanks to the simultaneous modification of arbitrarily big amount of copies, NEPs can produce a non-polynomial amount of solutions in polynomial time. We will see different examples of this behaviour in section 4.1.

This kind of formal design (inherent parallelism and an unrestricted amount of available memory) is frequent in the natural computing devices and are usually needed to get polynomial performance for NP problems.

Chapter 4

NEP's applications

4.1 Solving NP-complete problems with jNEP

In this section we informally introduce this topic. A formal description could be found in any manual [Garey and Johnson, 1979] on complexity and is out of the scope of this section.

NP may be informally defined as the set of decision problems that can be solved in polynomial time on a non-deterministic Turing machine. An NP problem is also complete if and only if every other problem in NP can be easily (in polynomial time) transformed into it. Polynomial performance on a non-deterministic Turing machine frequently corresponds to exponential performance (or worse) on a deterministic Turing machine. Classical von Neumann computers can be considered the closest implementation of deterministic Turing machines. Even more informally, the reader can consider a non-deterministic Turing machine as a set of as many Turing machines as needed, searching in parallel for a solution to the problem. Such a device will stop as soon as the first solution is found. Each Turing machine is expected to check its solution in polynomial time. In the previous statement, as many Turing machines as needed usually means an exponential number of machines. The reader can easily understand that if the same work has to be done by a single Turing machine, it has to check each of the possible solutions (an exponential amount of them) in a polynomial time, which results in a final exponential performance.

4.1.0.1 Solving the SAT problem with linear resources

Reference Manea et al. [2007] describes a NEP with splicing rules (ANSP) which solves the boolean satisfiability problem (SAT), well-known NP-complete problem, with linear resources, in terms of the complexity classes also present in Manea et al. [2007].

ANSP stands for Accepting Networks of Splicing Processors. In short, a ANSP is a NEP where the transformation rules of its nodes are *splicing rules*. The transformation performed by those rules is very similar to the genetic crossover. To be more precise, a *splicing rule* σ is a quadruple of words written as $\sigma = [(x, y); (u, v)]$. Given this *splicing rule* σ and two words (w,z), the action of σ on (w,z) is defined as follows:

$$\sigma(w, z) = \{t \mid w = \alpha xy\beta, z = \gamma uv\delta \text{ for any words } \alpha, \beta, \gamma, \delta \text{ and } t = \alpha xv\delta \text{ or } t = \gamma uy\beta\}$$

We can use jNEP to actually build and run the ANSP that solves the boolean satisfiability problem (SAT). We will see how the features of NEPs and the *splicing rules* can be used to tackle this problem. The following is a broad summary of

the configuration file for such a ANSP, applied to the solution of the SAT problem for three variables. The entire file can be downloaded from jnep.e-delrosal.net or studied in appendix A.

```
<NEP nodes="9">
  <ALPHABET symbols="A_B_C_!A_!B_!C_AND_OR_(_)_[A=1]_[B=1]_[C=1]_[A=0]_[B=0]_[C=0]_#_UP_{_}_1"/>
  <!-- WE IGNORE THE GRAPH TAG TO SAVE SPACE. THIS NEP HAVE A COMPLETE GRAPH -->
  <STOPPING_CONDITION>
    <CONDITION type="NonEmptyNodeStoppingCondition" nodeID="1"/>
  </STOPPING_CONDITION>
  <EVOLUTIONARY_PROCESSORS>
    <!-- INPUT NODE -->
    <NODE initCond="{_(_A_)_AND_(_B_OR_C_)_}" auxiliaryWords="{_[A=1]_# {_[A=0]_# {_[B=1]_# {_[B=0]_#
                                                              {_[C=1]_# {_[C=0]_#">
      <EVOLUTIONARY_RULES>
        <RULE ruleType="splicing" wordX="{" wordY="(" wordU="{_[A=1]" wordV="#"/>
        <RULE ruleType="splicing" wordX="{" wordY="(" wordU="{_[A=0]" wordV="#"/>
        <RULE ruleType="splicing" wordX="{" wordY="[A=0]" wordU="{_[B=0]" wordV="#"/>
        <RULE ruleType="splicing" wordX="{" wordY="[A=0]" wordU="{_[B=1]" wordV="#"/>
        <RULE ruleType="splicing" wordX="{" wordY="[A=1]" wordU="{_[B=0]" wordV="#"/>
        <RULE ruleType="splicing" wordX="{" wordY="[A=1]" wordU="{_[B=1]" wordV="#"/>
        <RULE ruleType="splicing" wordX="{" wordY="[B=0]" wordU="{_[C=0]" wordV="#"/>
        <RULE ruleType="splicing" wordX="{" wordY="[B=0]" wordU="{_[C=1]" wordV="#"/>
        <RULE ruleType="splicing" wordX="{" wordY="[B=1]" wordU="{_[C=0]" wordV="#"/>
        <RULE ruleType="splicing" wordX="{" wordY="[B=1]" wordU="{_[C=1]" wordV="#"/>
      </EVOLUTIONARY_RULES>
      <FILTERS>
        <INPUT type="4" permittingContext="" forbiddingContext="[A=1]_[B=1]_[C=1]_[A=0]_[B=0]_[C=0]_#_UP_{_}_1"/>
        <OUTPUT type="4" permittingContext="[C=1]_[C=0]" forbiddingContext=""/>
      </FILTERS>
    </NODE>
    <!-- OUTPUT NODE -->
    <NODE initCond="">
      <EVOLUTIONARY_RULES>
      </EVOLUTIONARY_RULES>
      <FILTERS>
        <INPUT type="1" permittingContext="" forbiddingContext="A_B_C_!A_!B_!C_AND_OR_(_)"/>
        <OUTPUT type="1" permittingContext="" forbiddingContext="[A=1]_[B=1]_[C=1]_[A=0]_[B=0]_[C=0]_#_UP_{_}_1"/>
      </FILTERS>
    </NODE>
    <!-- COMP NODE -->
    <NODE initCond="" auxiliaryWords="#_[A=0]_} #_[A=1]_} #_} #_1_)_}">
      <EVOLUTIONARY_RULES>
        <RULE ruleType="splicing" wordX="" wordY="A_OR_1_)_}" wordU="#" wordV="1_)_}"/>
        <RULE ruleType="splicing" wordX="" wordY="!A_OR_1_)_}" wordU="#" wordV="1_)_}"/>
        <RULE ruleType="splicing" wordX="" wordY="B_OR_1_)_}" wordU="#" wordV="1_)_}"/>
        <RULE ruleType="splicing" wordX="" wordY="!B_OR_1_)_}" wordU="#" wordV="1_)_}"/>
        <RULE ruleType="splicing" wordX="" wordY="C_OR_1_)_}" wordU="#" wordV="1_)_}"/>
        <RULE ruleType="splicing" wordX="" wordY="!C_OR_1_)_}" wordU="#" wordV="1_)_}"/>
        <RULE ruleType="splicing" wordX="" wordY="AND_(_1_)_}" wordU="#" wordV="}"/>
        <RULE ruleType="splicing" wordX="" wordY="[A=1]_(_1_)_}" wordU="#" wordV="[A=1]_}"/>
        <RULE ruleType="splicing" wordX="" wordY="[A=0]_(_1_)_}" wordU="#" wordV="[A=0]_}"/>
      </EVOLUTIONARY_RULES>
      <FILTERS>
        <INPUT type="1" permittingContext="1" forbiddingContext=""/>
        <OUTPUT type="1" permittingContext="" forbiddingContext="#_1"/>
      </FILTERS>
    </NODE>
    <!-- A=1 NODE -->
    <NODE initCond=""  auxiliaryWords="#_1_)_} #_)_}">
      <EVOLUTIONARY_RULES>
        <RULE ruleType="splicing" wordX="" wordY="A_)_}" wordU="#" wordV="1_)_}"/>
        <RULE ruleType="splicing" wordX="" wordY="(_!A_)_}" wordU="#" wordV="UP"/>
        <RULE ruleType="splicing" wordX="" wordY="OR_!A_)_}" wordU="#" wordV=")_}"/>
        <RULE ruleType="splicing" wordX="" wordY="B_)_}" wordU="#" wordV="UP"/>
        <RULE ruleType="splicing" wordX="" wordY="C_)_}" wordU="#" wordV="UP"/>
      </EVOLUTIONARY_RULES>
```

```
      <FILTERS>
        <INPUT type="1" permittingContext="[A=1]" forbiddingContext="[A=0]_1"/>
        <OUTPUT type="1" permittingContext="" forbiddingContext="#_UP"/>
      </FILTERS>
    </NODE>
    <!-- A=0 NODE -->
    <NODE initCond=""  auxiliaryWords="#_1_)_} #_)_}">
      <EVOLUTIONARY_RULES>
        <RULE ruleType="splicing" wordX="" wordY="OR_A_)_}" wordU="#" wordV=")_}"/>
        <RULE ruleType="splicing" wordX="" wordY="(_A_)_}" wordU="#" wordV="UP"/>
        <RULE ruleType="splicing" wordX="" wordY="!A_)_}" wordU="#" wordV="1"/>
        <RULE ruleType="splicing" wordX="" wordY="B_)_}" wordU="#" wordV="UP"/>
        <RULE ruleType="splicing" wordX="" wordY="C_)_}" wordU="#" wordV="UP"/>
      </EVOLUTIONARY_RULES>
      <FILTERS>
        <INPUT type="1" permittingContext="[A=0]" forbiddingContext="[A=1]_1"/>
        <OUTPUT type="1" permittingContext="" forbiddingContext="#_UP"/>
      </FILTERS>
    </NODE>
    <!-- NODES FOR 'B' AND 'C' ARE ANALOGOUS TO THOSE FOR 'A'. WE DO NOT PRESENT
     THEM TO SAVE SPACE-->
  </EVOLUTIONARY_PROCESSORS>
</NEP>
```

With this configuration file, at the end of its computation, jNEP outputs the interpretation which satisfies the logical formula contained in the file, namely:

```
(_A_)_AND_(_B_OR_C_): {_[C=0]_[B=1]_[A=1]_} {_[C=1]_[B=1]_[A=1]_} {_[C=1]_[B=0]_[A=1]_}
```

This ANSP is able to solve any formula with three variables. The formula to be solved must be specified as the value of the *initCond* attribute of the input node.

```
***************                NEP INITIAL CONFIGURATION              ***************
--- Evolutionary Processor 0 ---
{_(_A_)_AND_(_B_OR_C_)_}
```

Our ANSP works as follows. Firstly, the first node creates all the possible combinations for the 3 variables values. We show below the jNEP output for the first step:

```
*************** NEP CONFIGURATION - EVOLUTIONARY STEP - TOTAL STEPS: 1 ***************
--- Evolutionary Processor 0 ---
{_# {_[A=1]_(_A_)_AND_(_B_OR_C_)_}
{_[A=0]_(_A_)_AND_(_B_OR_C_)_}
```

As shown, the splicing rules of the initial node has appended the two possible values of A to two copies of the logical formula. The concerning rules are:

```
<RULE ruleType="splicing" wordX="{" wordY="(" wordU="{_[A=1]" wordV="#"/>
<RULE ruleType="splicing" wordX="{" wordY="(" wordU="{_[A=0]" wordV="#"/>
```

These kind of rules (Manea's splicing rules) use some auxiliary words that are never removed from the nodes. In our ANSP we use the following auxiliary words:

```
auxiliaryWords="{_[A=1]_# {_[A=0]_# {_[B=1]_# {_[B=0]_# {_[C=1]_# {_[C=0]_#"
```

The end of this first stage arises after $2n - 1$ steps, where n is the number of variables:

```
--- Evolutionary Processor 0 ---
{_# {_[C=0]_[B=0]_[A=0]_(_A_)_AND_(_B_OR_C_)_}
{_[C=0]_[B=0]_[A=1]_(_A_)_AND_(_B_OR_C_)_}
{_[C=1]_[B=0]_[A=0]_(_A_)_AND_(_B_OR_C_)_}
{_[C=1]_[B=0]_[A=1]_(_A_)_AND_(_B_OR_C_)_}
{_[C=0]_[B=1]_[A=0]_(_A_)_AND_(_B_OR_C_)_}
{_[C=0]_[B=1]_[A=1]_(_A_)_AND_(_B_OR_C_)_}
{_[C=1]_[B=1]_[A=0]_(_A_)_AND_(_B_OR_C_)_}
{_[C=1]_[B=1]_[A=1]_(_A_)_AND_(_B_OR_C_)_}
```

We would like to remark that NEPs take advantage of the possibility of applying all the rules to one word in the same step. This is because the model states that each word has an arbitrary number of copies in its processor. Therefore, the above task (which is $\Theta(2^n)$) can be completed in n steps, since each step doubles the number of words by appending to each word a new variable with the value 1 or 0.

After this first stage, the words can leave the initial node and travel to the rest of the nodes. In the net, there is one node per variable and value, in other words, there is one node for $A = 1$, another for $C = 0$ and so on. Each of this node reduces, from right to left, the word formula according to the variable values. For example, the sixth node is responsible for $C = 1$ and, thus, makes the following modification to the word `{_[C=1]_[B=1]_[A=1]_(_A_)_AND_(_B_OR_C_)_}`:

`{_[C=1]_[B=1]_[A=1]_(_A_)_AND_(_B_OR_C_)_}` $\Longrightarrow$

`{_[C=1]_[B=1]_[A=1]_(_A_)_AND_(_B_OR_1_)_}`

However, the ninth node is responsible for $C = 0$ and, therefore, produces the following change:

`{_[C=0]_[B=1]_[A=1]_(_A_)_AND_(_B_OR_C_)_}` $\Longrightarrow$

`{_[C=0]_[B=1]_[A=1]_(_A_)_AND_(_B_)_}`

In this way, the nodes share the results of their modifications until one of them produces a word where the formula is empty and, thus, it only contains the left side with the variable values. This kind of words is allowed to pass the input filter of the output node, therefore, they will enter it. At this point the NEP halts, since the stopping condition of the NEP says that a non-empty output node is the signal to stop the computation.

For more details we refer to Manea et al. [2007], the implementation in jnep.e-delrosal.net and appendix A.

4.1.0.2 Hamiltonian path problem

This well-known NP-complete problem searches an undirected graph for a Hamiltonian path, that is, one that visits each vertex exactly once.

In his work Adleman [1994], Adleman proposed a way to solve this problem with polynomial resources by means of DNA manipulations in the laboratory. Figure 4.1 shows the graph used by Adleman. In this case, the solution is obvious (path 0-1-2-3-4-5-6) Despite its simplicity, Adleman described a general algorithm applicable to almost any graph with the same performance.

Adlemans algorithm can be summarized as follows: 1. Generating randomly all the possible paths. 2. Selecting those paths that begin and end in the proper nodes. 3. Selecting only the paths that contain exactly the total number of nodes. 4. Removing those paths that contain some node more than once. 5. The remaining paths are solutions for the problem.

The present work follows a similar approach. The NEP graph is very similar to the one studied above: an extra node is added to ease the definition of the stopping condition. The set i,0,1,2,3,4,5,6 is used as the alphabet. Symbol i is the initial content of the initial node (v0) Each node (except the final one) adds its number to the string received from the network. Input and output filters are defined to allow the communication of all the possible words without any special constraint. The input filter of the final node excludes any string which is not a solution. It is easy to imaging a regular expression for the set of solutions (those words with the proper length, the proper initial and final node and where each node appears only

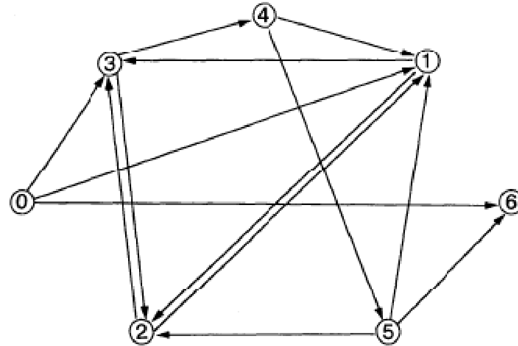


Figure 4.1: Graph studied by Adleman

once). The NEP basic model allows defining filters by means of regular expressions. It is also easy to devise a set of additional nodes that performs the previous filter following Adlemans checks (proper beginning and end, proper length, and number of occurrences of each node). For the sake of simplicity we have used explicitly the solution word (i_0_1_2_3_4_5_6) instead of a more complex regular expression or a greater NEP.

The reader will find at http://jnep.e-delrosal.net the complete XML file for this problem (Adleman.xml), it is also available in appendix B. Previously, in section 3.1.3 we have executed it with jNEPView. Some of their sections are explained below:

The XML file for this example defines the alphabet with this tag

```
<ALPHABET symbols="i_0_1_2_3_4_5_6" />
```

the initial content of node 0 in the following way

```
<NODE initCond="i">
```

The rules for adding the number of the node to its string are defined as follows (here for node 2)

```
<RULE ruleType = "insertion" actionType = "RIGHT" symbol = "2"/>
```

There are several ways of defining filters for the desired behavior (to allow the communication of all the possible words without any special constraint). We have used only the permitted input and output filters. A string can enter a node if it contains any of the symbols of the alphabet and no string is forbidden.

```
<FILTERS>
  <INPUT type="2"
    permittingContext="i_0_1_2_3_4_5_6"
    forbiddingContext="" />
  <OUTPUT type="2"
    permittingContext="i_0_1_2_3_4_5_6"
    forbiddingContext="" />
</FILTERS>
```

The behavior of the NEP is sketched as follows:

1. In the initial step the only non-empty node is 0 and contains the string i

2. After the first step, 0 is added to this string and thus, node 0 contains i_0

3. This string is moved to the nodes connected with node 0. In the next steps only nodes 1, 3 and 6 contain i_0

4. These nodes add their number to the received string. In the next step their contents are respectively i_0_1, i_0_3 and i_0_6

5. This process is repeated as many times as needed to produce a string that meets the conditions of the solution. In this final step the solution string i_0_1_2_3_4_5_6 is sent to node 7 and the NEP stops.

The definition of filters in NEP model poses some difficulties to the design of NEPs and, thus, to the development of a simulator. These filters are defined [Castellanos et al., 2001, 2003] by means of two couples of filters (forbidden and allowed) to each operation (input and output). There exists, in addition, different ways of combining and apply the filters to translate them into a set of strings. This mechanism contains obvious redundancies that make it difficult to design NEPs. A more general agreement of the researchers is advisable to ease and simplify the development of NEPs simulators.

4.1.0.3 Coloring problems

This problem introduces a map whose regions have to be colored with only three colors, and with a different one for each pair of adjacent regions. We have used the NEP defined in Castellanos et al. [2003]. The map is translated into an undirected graph whose nodes stand for the regions and whose edges represent the adjacency relationship between regions. Figure 4.2 shows one of the examples studied in this section. It is ease to prove that there is no solution to this map.

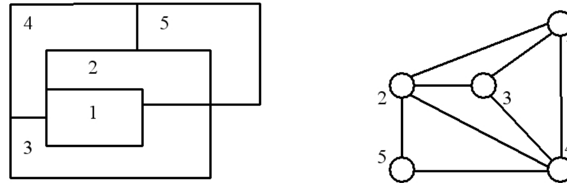


Figure 4.2: Example of a map and its adjacency graph. In this case, there is no solution for the 3-colorability problem

The NEP has a complete graph with two special nodes (for the initial and final steps) and a set of seven nodes associated to each edge of the adjacency graph. These nodes perform the tasks outlined below. The next paragraphs describe them with more detail.

The initial (final) node is responsible for starting (stopping) the computation. The seven nodes associated with an edge of the map are grouped in three couples (one for each color). There is, in addition, a special node to communicate with the set of nodes of the next edge. Each couple is responsible for the main operation in the NEP: to check that a coloring constraint is not violated for the current edge. It performs this task in the following way:

Let us suppose that the color red is the one associated with the pair of nodes. The first node in the NEP associates the color red to the first node of the edge in the map. The second node in the NEP simultaneously keeps all the allowed coloring (two, in this case) for the second node of the edge: (blue and green) It is clear that the only acceptable colorings for this edge are red-blue and red-green.

The behavior of the complete NEP could be sketched as follows:

1) The initial node generates all the possible assignment of colors to all the regions in the map and adds a symbol to identify the first edge to be checked. These strings are communicated to all the nodes of the graph. 2) The set of nodes

associated with each edge accepts only the strings marked with the symbol of the edge. These nodes remove all the strings that violate the coloring constraint for the regions of the edge. One special node in the set replaces the edge mark with that which corresponds to the next edge. In this way, the process continues. 3) The final node of the NEP collects the strings that satisfy the constraints of all the edges. It is ease to see that these strings are the solutions.

Some fragments of the XML file for this example are shown below to describe the above behavior with more detail:

The alphabet of the NEP is defined as follows:

```
<ALPHABET
symbols="b1_r1_g1_b2_r2_g2_b3_r3_g3_b4_r4_g4_b5_r5_g5_B1_R1_G1_B2_R2_G2_B3_R3_G3
_B4_R4_G4_B5_R5_G5_a1_a2_a3_a4_a5_X1_X2_X3_X4_X5_X6_X8_X9"/>
```

This alphabet contains the following subsets of symbols: a1,...,a5 represents the initial situation of the regions (uncolored). b1, r1, g1,..., b5, r5, g5 represents the assignment of the colors to the regions. B1, R1, G1,..., B5, R5, G5 is a copy of the previous set to be used while checking the constraint associated with a couple of adjacent regions.

The string contained in the initial node at the beginning represents the complete map uncolored and the number of the first edge to be tackled (X1)

```
<NODE initCond="a1_a2_a3_a4_a5_X1">
```

The rules of the initial node assign all the possible colors to all the regions. The following rules refer to the second region:

```
<RULE ruleType = "substitution"
      actionType = "ANY"
      symbol="a2" newSymbol="b2"/>
<RULE ruleType="substitution"
      actionType="ANY"
      symbol="a2" newSymbol="r2"/>
<RULE ruleType="substitution"
      actionType="ANY"
      symbol="a2" newSymbol="g2"/>
```

The node in the NEP that assigns a color (Red, in this case) to the first region ("1" in the example) of an edge in the map uses the following rule:

```
<RULE ruleType="substitution"
      actionType="ANY" symbol="r1"
      newSymbol="R1"/>
```

The other node ensures that the adjacent region (2 in this case) has a different color by means of these rules:

```
<RULE ruleType="substitution"
      actionType="ANY"
      symbol="b2"
      newSymbol="B2"/>

<RULE ruleType="substitution"
      actionType="ANY"
      symbol="g2"
      newSymbol="G2"/>
```

The node used for starting the process in the next edge removes any special (capitalized) color symbol and sets the edge marking to the next one. The following rules correspond to the first edge.

```
<RULE ruleType="substitution"
      actionType="ANY" symbol="R1"
      newSymbol="r1"/>
<RULE ruleType="substitution"
      actionType="ANY" symbol="B1"
```

```
      newSymbol="b1"/>
<RULE ruleType="substitution"
      actionType="ANY" symbol="G1"
      newSymbol="g1"/>
<RULE ruleType="substitution"
      actionType="ANY" symbol="R2"
      newSymbol="r2"/>
<RULE ruleType="substitution"
      actionType="ANY" symbol="B2"
      newSymbol="b2"/>
<RULE ruleType="substitution"
      actionType="ANY" symbol="G2"
      newSymbol="g2"/>
<RULE ruleType="substitution"
      actionType="ANY" symbol="X1"
      newSymbol="X2"/>
```

Notice that the set of nodes associated with the last edge (in this case with the number 8) mark its strings with the following number that does not correspond with any edge in the graph (9 in our example). This is important for the design of the final node. A special node of the NEP checks the stopping condition (Non Empty Node Stopping Condition). This final node only accepts strings with the corresponding mark (one that does not correspond to any edge in the adjacency graph).

Figure 4.3 shows another map that could be colored with 3 colors. Splitting region 3 and 4 in figure 4.2 generates this map. Figure 4.3 also summarizes the sequence of steps for one of the possible solutions. It is worth noticing that all the solutions are simultaneously kept in the configurations of the NEP.

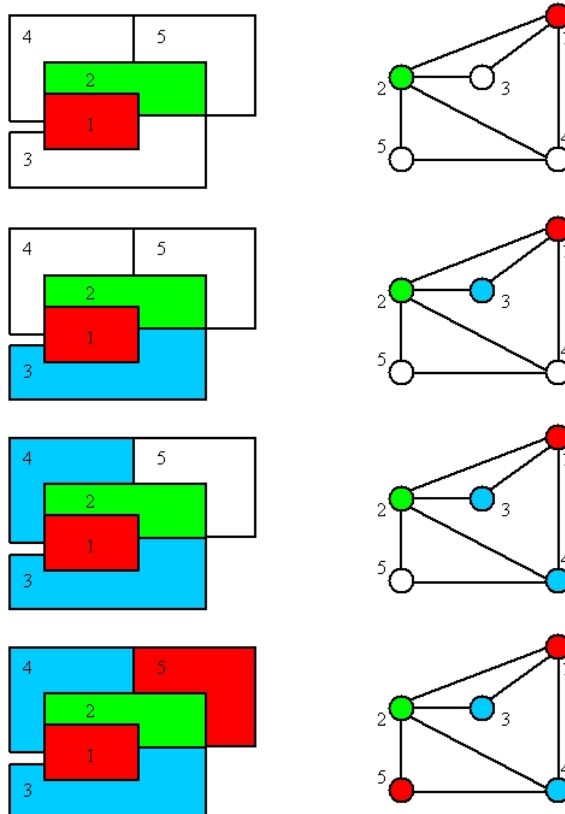


Figure 4.3: Sequence of steps in the solution of a 3-coloring problem by jNEP

The behavior of the NEP for this map could be summarized as follows: the initial content of the initial node is a1_a2_a3_a4_a5_X1. This node produces all the possible coloring combinations. In the second step of the computation, for example, it contains the following strings:

b1_a2_a3_a4_a5_X1
r1_a2_a3_a4_a5_X1
g1_a2_a3_a4_a5_X1
a1_b2_a3_a4_a5_X1
a1_r2_a3_a4_a5_X1
a1_g2_a3_a4_a5_X1
a1_a2_b3_a4_a5_X1
a1_a2_r3_a4_a5_X1
a1_a2_g3_a4_a5_X1
a1_a2_a3_b4_a5_X1
a1_a2_a3_r4_a5_X1
a1_a2_a3_g4_a5_X1
a1_a2_a3_a4_b5_X1
a1_a2_a3_a4_r5_X1
a1_a2_a3_a4_g5_X1

The NEP still needs a few more steps to get all the combinations. After that, the coloring constraints are applied simultaneously to all the possible solutions and those assignments that violate some constraint are removed. We describe below a sequence of strings generated by the NEP that corresponds to the solution graphically shown in figure 4.2: r1_g2_b3_b4_r5_X1 is generated in the initial steps. After checking the 1st edge (regions 1 and 2) the NEP contains these two strings R1_g2_b3_b4_r5_X1 and R1_G2_b3_b4_r5_X1 After checking the 2nd edge (regions 1 and 3) R1_g2_B3_b4_r5_X2 And after checking the edges 3, 4, 5, 6 and 8 (remember that edge 7 was removed to make the map colorable) associated respectively with the pairs of regions 1-4, 2-3, 2-4, 2-5 and 4-5, the following strings are in the NEP: R1_g2_b3_B4_r5_X3 r1_G2_B3_b4_r5_X4 r1_G2_b3_B4_r5_X5 r1_G2_b3_b4_R5_X6 r1_g2_b3_B4_R5_X8 Finally, the complete solution is found r1_g2_b3_B4_R5_X9 and r1_g2_b3_b4_r5_X9

This NEP processes all the solutions at the same time. It removes all the coloring combinations that violate any constraint. The final node contains in the last step all the solutions found. Castellanos et al. [2003] describes one of the kinds of NEPs (simple NEPs) that is simulated by jNEPs. As we have briefly mentioned before, we have observed that the authors have used slightly different filters for the 3-coloring problem. We could not use these filters and we had to change some of them (most of the output filters) in order to get a proper behavior of the NEP. The complete XML file is available at http://jnep.e-delrosal.net or in appendix D.

4.1.0.4 Final comments

We have tackled the solution of several NP-complete problems found in the NEP's literature by means of jNEP. We have observed that there exists different ways of implementing the same formal device, mainly with respect to input and output filters. These open aspects have to be defined when the model is implemented to solve given problems. We conclude that simulation needs both: a formal definition and also some standardization in the way in which different authors particularize these open aspects in the implementation of their own NEPs. These differences make it very difficult to fully understand the behavior of the proposed NEPs as well as their simulation. Although we have not found any significant mistake in

the simulation of the formal model, we had to modify and improve jNEP in several subtle details in order to ease the handling of the NEPs described in the literature.

We have also identified some common techniques to these different NP problems. They suggested us some tools that could be added to jNEP to increase the comfort of the NEPs designer. A more abstract input format would. For example, most of the NEPs defined to solve NP problems uses complete graphs. The current XML configuration file explicitly defines each edge, which implies a big amount of tedious and mechanical work. It would be very useful for some automatic mechanism to do this task. We have already presented jNEP's modules which try to facilitate this kind of task in sections 3.1.4 and 3.1.5. It could be also very useful adding some diagnose tool to check the correctness of the NEPs. It is worth noticing that jNEP is just a block that will be used to build more complex applications.

4.2 NEPs for parsing

In this section we propose PNEP (*Parsing NEP*), a simple extension to NEP and a procedure to translate a grammar into a PNEP that recognizes the same language. These parsers based on NEPs do not impose any additional constraint to the structure of the grammar, which can contain all kinds of recursive, lambda or ambiguous rules. This flexibility makes this procedure specially suited for Natural Languge Processing (NLP). In a first proof with a simplified English grammar, we got a performance (a linear time complexity) similar to that of the most popular syntactic parsers (Early and its derivatives).

Other authors have previously studied the relationships between NEPs, regular, context-free, and recursively enumerable languages [Castellanos et al., 2005, Manea, 2004a, Manea et al., 2006, Martin-Vide et al., 2003, Margenstern et al., 2005]. For our porpouses, it is especially important Csuhaj-Varju et al. [2005], where it is shown how NEPs simulate the application of context free rules ($A \to \alpha, A \in V, \alpha \in V^*$ for alphabet V): a set of additional nodes is needed to implement a rather complex technique to rotate the string and locate A in one of the string ends, then delete it and add all the symbols in α. Given this result, we can assume that NEPs can apply context free rules and, for the sake of simplicity, add this function to the PNEP primitive functions. Thus, PNEPs use context free rules rather than classic substitution, insertion and deletion NEP rules. In this way, the expressive power of NEP processors is bounded, while providing a more natural and comfortable way to describe the parsed language for practical purposes.

PNEPs implement a top down parser for context free grammars. We will show how PNEPs explore all the possible derivation trees, taking advantage of the inherent parallelism of NEPs. In this way, the parser is able to generate all the possible derivations of each string in the language generated by the grammar. As the PNEP explore all the derivation trees in a top-down manner, its temporal complexity is bounded by the length of the analyzed string. In other words, the PNEP does not need to continue the search once the strings derived get longer than the target string. This bound can be used to stop the computation when processing incorrect strings, thus avoiding running the PNEP for a possible infinite number of steps.

The PNEP is built from the grammar in the following way: (1) We assume that each derivation rule in the grammar has a unique index that can be used to reconstruct the derivation tree. (2) There is a node for each non terminal. Each node applies to the strings all the derivation rules for its non terminal. The filters, as well as the graph layout, allow all the nodes to share all the intermediate steps in the derivation process. (3) There is an additional output node, in which the *parsed string* can be found: this is a version of the input, enriched with information that will make it possible to reconstruct the derivation tree (the rules indices). (4) The

graph is complete.

Obviously the same task can be performed using a trivial PNEP with only a node for all the derivation rules. However, the proposed PNEP is easier to analyze and more useful to distribute the work among several nodes.

We shall consider as an example the grammar $G_{a^n b^n c^m}$ induced by the following derivation rules (notice that indexes have been added in front of the corresponding right hand side):

$$X \Rightarrow (1)SO \ , \ S \Rightarrow (2)aSb|(3)ab \ , \ O \Rightarrow (4)Oo|(5)oO|(6)o$$

It is easy to prove that the language corresponding to this grammar is $\{a^n b^n o^m \mid n, m > 0\}$. Furthermore, the grammar is ambiguous, since every sequence of o symbols can be generated in two different ways: by producing the new terminal o with rule 4 or with rule 6.

The input filters of the output node describe parsed copies of the initial string. In other words, strings whose symbols are preceded by strings of any length (including 0) of the possible rules indexes. As an example, a parsed version of the string $aabboo$ would be $12a3abb5o6o$.

In this work, we assume that a PNEP Γ is formally defined as follows:

$\Gamma = (V, N_1, N_2, ..., N_n, G)$, where V is an alphabet and for each $1 \leq i \leq n, N_i = (M_i, A_i, PI_i, PO_i)$ is the i-th evolutionary node processor of the network. The parameters of every processor are:

- M_i is a finite set of context-free evolution rules
- A_i is the set of initial strings in the i-th node.
- PI_i and PO_i are subsets of V^* representing respectively the input and the output filters. These filters are defined by the membership condition, namely a string $w \in V^*$ can pass the input filter (the output filter) if $w \in PI_i (w \in PO_i)$. In this case, we will use two kinds of filters:
  - Those defined as two components (P, F) of *P*ermitting and *F*orbidding contexts (a word w passes the filter if $(\alpha(w) \subseteq P) \wedge (F \cap \alpha(w) = \Phi)$).
  - Those defined as regular expressions r (a word w passes the filter if $w \in L(r)$, where $L(r)$ stands for the language defined by the regular expression r).

Finally, $G = (N_1, N_2, ..., N_n, E)$ is an undirected graph (whose edges are E), called the underlying graph of the network.

We will now describe the way in which our PNEP is defined, starting from a certain grammar. Given the context free grammar $G = \{\Sigma_T = \{t_1, ..., t_n\}, \Sigma_N = \{N_1, ..., N_m\}, A, P\}$ with $A \in \Sigma_N$ its axiom and $P = \{l_j \to \gamma_j \mid j \in \{1, .., k\}, l_j \in \Sigma_N \wedge \gamma_j \in (\Sigma_T \cup \Sigma_N)^*\}$ its set of k production rules the PNEP is defined as

$$\Gamma_G = (V = \Sigma_T \cup \Sigma_N \cup \{1, ..., k\}, node_{output}, N_1, N_2, ..., N_m, G)$$

where 1) $node_{output}$ is the output node; 2) G is a complete graph and 3) the N_i node corresponding to the axiom is called the *input node* A, which is the only one with a non empty initial content (A). Each non terminal node N_i in the PNEP has a context free rule for each derivation rule in the grammar applicable to it. This rule changes the nonterminal by a string made by appending the right hand side of the derivation rule with the index of the rule in P. For example, the PNEP for grammar $G_{a^n b^n o^m}$ described above has a node for nonterminal S with the following substitution rules: $\{S \to 2aSb, S \to 3ab\}$ The input filters of these nodes allow all strings containing some copy of their non terminal to input the node. Strings that

do not contain a copy of the non terminal pass the output filter. We can get this behavior by using the set with the non terminal symbol of the node ($\{N_i\}$) both as the permitted input filter and as the forbidden output filter.

The input filter for the output node $node_{output}$ has to describe what we have called *parsed strings*. Parsed strings will contain numbers, corresponding to the derivation rules which have been applied, among the symbols of the analyzed string. We can easily create a regular expression and define the input filter by means of membership. For example, in order to parse the string *aabbo* with the grammar we are using above, the regular expression can be $\{1,2,3,4,5,6\}*a\{1,2,3,4,5,6\}*a\{1,2,3,4,5,6\}*b\{1,2,3,4,5,6\}*b\{1,2,3,4,5,6\}*o$. For the sake of simplicity, our PNEP will stop computing whenever a string enters the output node, however it could continue parsing to find derivations of any depth. As commented before, the temporal bound could be the length of the analyzed string. This bound could be implemented by stopping conditions that check the length of the current derivations in the NEP or stop after a predetermined number of steps.

The complete PNEP for our example ($\Gamma_{a^nb^no^m}$) is defined as follows:

- Alphabet $V = \{X, O, S, a, b, o, 1, 2, 3, 4, 5, 6\}$
- Nodes
  - $node_{output}$: $A_{output} = \Phi$ is the initial content; $M_{output} = \Phi$ is the set of rules; $PI_{output} = \{$ (regular expression membership filter); $\{\{1,2,3,4,5,6\}*a\{1,2,3,4,5,6\}*a\{1,2,3,4,5,6\}*b\{1,2,3,4,5,6\}*b\{1,2,3,4,5,6\}*o\}\}$; $PO_{output} = \Phi$ is the output filter
  - N_X: $A_X = \{X\}$; $M_X = \{X \to 1SO\}$; $PI_X = \{P = \{X\}, F = \Phi\}$; $PO_X = \{F = \{X\}, P = \Phi\}$
  - N_S: $A_S = \Phi$; $M_S = \{S \to 2aSb, S \to 3ab\}$; $PI_S = \{P = \{S\}, F = \Phi\}$; $PO_S = \{F = \{S\}, P = \Phi\}$
  - N_O: $A_O = \Phi$; $M_O = \{O \to 4oO, O \to 5Oo, O \to 5o\}$; $PI_O = \{P = \{O\}, F = \Phi\}$; $PO_O = \{F = \{O\}, P = \Phi\}$
  - It has a complete graph
  - It stops the computation when some string enters $node_{output}$

Some of the strings generated by all the nodes of the PNEP in successive communication steps when parsing the string *aboo* are shown below (each set corresponds to a different step): $\{X\} \Rightarrow \{1SO\} \Rightarrow \{..., 13abO, ...\} \Rightarrow$
$\{..., 13ab4Oo, 13ab5oO, ...\} \Rightarrow \{..., 13ab46oo, ..., 13ab5o6o, ...\}$

The last set contains two different derivations for *aboo* by ($G_{a^nb^nno^m}$), that can enter the output node and stop the computation of the PNEP.

It is easy to reconstruct the derivation tree from the parsed strings in the output node, by following their sequence of numbers. For example, consider the parsed string $13ab6o$ and its sequence of indexes 136; *abo* is generated in the following steps: $X \Rightarrow$ (rule 1 $X \Rightarrow SO$) SO , $SO \Rightarrow$ (rule 3 $S \Rightarrow ab$) abO , $abO \Rightarrow$ (rule 6 $O \Rightarrow o$) abo

In section 3.1.2 we have described the structure of the xml input files for *jNEP*. In order to keep jNEP as general as possible, we have added new xml descriptions for the extension needed in PNEP.

Context free rules are represented in the xml file as follows:

```
<RULE ruleType="substitution" symbol="[symbol]" newString="[symbolList]"/>
```

The complete xml representation of the NEP $\Gamma_{a^nb^no^m}$ is shown below. The first node is the output node where the final syntactic trees will be placed. The other nodes correspond to each non-terminal and, consequently, contain the rules to derive each of them.

```
<NEP nodes="4">

  <ALPHABET symbols="X_S_a_b_O_o_1_2_3_4_5_6"/>

  <GRAPH>
    <EDGE vertex1="0" vertex2="1"/>
    <EDGE vertex1="0" vertex2="2"/>
    <EDGE vertex1="0" vertex2="3"/>
    <EDGE vertex1="1" vertex2="2"/>
    <EDGE vertex1="1" vertex2="3"/>
    <EDGE vertex1="2" vertex2="3"/>
  </GRAPH>

  <EVOLUTIONARY_PROCESSORS>

   <NODE initCond="">
      <EVOLUTIONARY_RULES>
      </EVOLUTIONARY_RULES>
      <FILTERS>
        <INPUT type="RegularLangMembershipFilter" regularExpression="[1-6]*a[1-6]*a[1-6]*a[1-6]*b[1-6]*b[1-6]*b[1-6]*
                                                        o[1-6]*o[1-6]*o[1-6]*o"/>
        <OUTPUT type="1" permittingContext="" forbiddingContext=""/>
      </FILTERS>
    </NODE>

    <NODE initCond="X">
      <EVOLUTIONARY_RULES>
        <RULE ruleType="contextFreeParsing" actionType="ANY" symbol="X" newSymbol="1_S_O"/>
      </EVOLUTIONARY_RULES>
      <FILTERS>
        <INPUT type="1" permittingContext="X" forbiddingContext=""/>
        <OUTPUT type="1" permittingContext="" forbiddingContext="X"/>
      </FILTERS>
    </NODE>

    <NODE initCond="">
      <EVOLUTIONARY_RULES>
        <RULE ruleType="contextFreeParsing" actionType="ANY" symbol="S" newSymbol="2_a_S_b"/>
        <RULE ruleType="contextFreeParsing" actionType="ANY" symbol="S" newSymbol="3_a_b"/>
      </EVOLUTIONARY_RULES>
      <FILTERS>
        <INPUT type="1" permittingContext="S" forbiddingContext=""/>
        <OUTPUT type="1" permittingContext="" forbiddingContext="S"/>
      </FILTERS>
    </NODE>

    <NODE initCond="">
      <EVOLUTIONARY_RULES>
        <RULE ruleType="contextFreeParsing" actionType="ANY" symbol="O" newSymbol="4_O_o"/>
        <RULE ruleType="contextFreeParsing" actionType="ANY" symbol="O" newSymbol="5_o_O"/>
        <RULE ruleType="contextFreeParsing" actionType="ANY" symbol="O" newSymbol="6_o"/>
      </EVOLUTIONARY_RULES>
      <FILTERS>
        <INPUT type="1" permittingContext="O" forbiddingContext=""/>
        <OUTPUT type="1" permittingContext="" forbiddingContext="O"/>
      </FILTERS>
    </NODE>

  </EVOLUTIONARY_PROCESSORS>

  <STOPPING_CONDITION>
    <CONDITION type="NonEmptyNodeStoppingCondition" nodeID="0"/>
  </STOPPING_CONDITION>
</NEP>
```

The actual output of jNEP for this configuration file is presented below. The output shows in detail how the derivation trees are built, while only those fitting the target word arrive at the output node.

```
***************                NEP INITIAL CONFIGURATION                ***************
--- Evolutionary Processor 0 ---

--- Evolutionary Processor 1 ---
X
--- Evolutionary Processor 2 ---

--- Evolutionary Processor 3 ---
```

```
*************** NEP CONFIGURATION - EVOLUTIONARY STEP - TOTAL STEPS: 1 ***************
--- Evolutionary Processor 0 ---

--- Evolutionary Processor 1 ---
1_S_0
--- Evolutionary Processor 2 ---

--- Evolutionary Processor 3 ---


*************** NEP CONFIGURATION - COMMUNICATION STEP - TOTAL STEPS:  2 ***************
--- Evolutionary Processor 0 ---

--- Evolutionary Processor 1 ---

--- Evolutionary Processor 2 ---
1_S_0
--- Evolutionary Processor 3 ---
1_S_0


*************** NEP CONFIGURATION - EVOLUTIONARY STEP - TOTAL STEPS: 3 ***************
--- Evolutionary Processor 0 ---

--- Evolutionary Processor 1 ---

--- Evolutionary Processor 2 ---
1_2_a_S_b_0 1_3_a_b_0
--- Evolutionary Processor 3 ---
1_S_5_o_0 1_S_4_0_o 1_S_6_o


*************** NEP CONFIGURATION - COMMUNICATION STEP - TOTAL STEPS:  4 ***************
--- Evolutionary Processor 0 ---

--- Evolutionary Processor 1 ---

--- Evolutionary Processor 2 ---
1_2_a_S_b_0 1_S_4_0_o 1_S_6_o
--- Evolutionary Processor 3 ---
1_S_5_o_0 1_3_a_b_0


*************** NEP CONFIGURATION - EVOLUTIONARY STEP - TOTAL STEPS: 5 ***************
--- Evolutionary Processor 0 ---

--- Evolutionary Processor 1 ---

--- Evolutionary Processor 2 ---
1_2_a_S_b_4_0_o 1_3_a_b_4_0_o 1_3_a_b_6_o 1_2_a_2_a_S_b_b_0 1_2_a_S_b_6_o 1_2_a_3_a_b_b_0
--- Evolutionary Processor 3 ---
1_3_a_b_4_0_o 1_S_5_o_4_0_o 1_3_a_b_6_o 1_S_5_o_5_o_0 1_3_a_b_5_o_0 1_S_5_o_6_o


*************** NEP CONFIGURATION - COMMUNICATION STEP - TOTAL STEPS:  6 ***************
--- Evolutionary Processor 0 ---

--- Evolutionary Processor 1 ---

--- Evolutionary Processor 2 ---
1_S_5_o_4_0_o 1_S_5_o_5_o_0 1_S_5_o_6_o
--- Evolutionary Processor 3 ---
1_2_a_S_b_4_0_o 1_3_a_b_4_0_o 1_2_a_2_a_S_b_b_0 1_2_a_3_a_b_b_0


*************** NEP CONFIGURATION - EVOLUTIONARY STEP - TOTAL STEPS: 7 ***************
--- Evolutionary Processor 0 ---

--- Evolutionary Processor 1 ---

--- Evolutionary Processor 2 ---
1_3_a_b_5_o_5_o_0 1_2_a_S_b_5_o_4_0_o 1_2_a_S_b_5_o_5_o_0 1_3_a_b_5_o_6_o 1_3_a_b_5_o_4_0_o 1_2_a_S_b_5_o_6_o
--- Evolutionary Processor 3 ---
1_2_a_S_b_4_6_o_o 1_3_a_b_4_6_o_o 1_3_a_b_4_5_o_0_o 1_2_a_3_a_b_b_4_0_o 1_2_a_3_a_b_b_6_o 1_2_a_2_a_S_b_b_4_0_o
1_2_a_S_b_4_4_0_o_o 1_2_a_2_a_S_b_b_5_o_0 1_2_a_S_b_4_5_o_0_o 1_3_a_b_4_4_0_o_o 1_2_a_3_a_b_b_5_o_0 1_2_a_2_a_S_b_b_6_o


*************** NEP CONFIGURATION - COMMUNICATION STEP - TOTAL STEPS:  8 ***************
--- Evolutionary Processor 0 ---
1_3_a_b_4_6_o_o 1_3_a_b_5_o_6_o
```

```
--- Evolutionary Processor 1 ---

--- Evolutionary Processor 2 ---
1_2_a_S_b_4_6_o_o 1_2_a_2_a_S_b_b_5_o_0 1_2_a_S_b_4_5_o_0_o 1_2_a_2_a_S_b_b_4_0_o 1_2_a_S_b_4_4_0_o_o 1_2_a_2_a_S_b_b_6_o
--- Evolutionary Processor 3 ---
1_3_a_b_5_o_5_o_0 1_2_a_S_b_5_o_4_0_o 1_2_a_S_b_5_o_5_o_0 1_3_a_b_5_o_4_0_o


---------------------- NEP has stopped!!! ----------------------

Stopping condition found: net.e_delrosal.jnep.stopping.NonEmptyNodeStoppingCondition

-----------------------------------------------------------------
```

4.2.1 Efficiency improvements

The previous approach seems naive and spatially inefficient, because of the big number of strings and derivations simultaneously considered which the processors have to store. However, the theoretical model assumes that the number of strings (and copies of each string) is virtually boundless, besides the evolution of all the string in one step is a primitive operation of the device. We must remember at this point that the NEP model is inspired by cell and molecular biology. Therefore, from a theoretical point of view, the derivation of a huge amount of strings does not imply spatial or time inefficiencies. On the other hand, from a simulation point of view, this issue is quite important. In practice, our NEP shall be simulated in a conventional von Neumann computer and, thus, reducing the number of strings and operations on them is a key efficiency point. For that reason, we have added two additional mechanisms to overcome this practical inefficiency.

Discarding non promising sentential forms

The first check we have implemented is the lightest and it is present in almost all the parsers: discarding any sentential form that contains a terminal symbol that the analyzed string does not contain. Parsers usually check sequentially for this condition, starting at the right-end of the string. NEPs filters offer the possibility of checking the condition regardless of the positions in the sentential form where the incorrect symbols are. Thus, we have implemented this feature by means of a new node with a specific filter that only allows strings composed of non-terminals and terminals included in the analyzed string. This pruning node does not play any other role in the computation, since it just contains a deletion rule that deletes nothing (no symbol). To be more specific, all the derivation nodes are connected to the pruning node and only to it. The pruning actually happens during the communication step because it is done by means of the input filter. A string can pass the filter if it only contains non terminals or terminals that belong to the input string being parsed. Next communication step sends the strings back to the derivation nodes. This way, PNEPs duplicate the number of steps needed to parse a string, but reduce the number of strings stored by the processors.

Forcing a left-most derivation order

Applying in parallel all the possible rules to a sentential form produces a lot of different derivations that actually are the same derivation tree. They only differ in the order in which the non terminal symbols of the same sentential form are derived. We extend the NEP model with a new specialized kind of context free evolutive rule that applies only to the left-most non terminal. The symbol $\rightarrow_l$ will be used to represent this kind of rule. The result of applying the rule $r : A \rightarrow_l s$, where $s \in V^*$ (V stands for the NEP's alphabet) on a given string w, can be formally defined as follows:

$$r(w) = t, w = w_1Aw_2 \wedge t = w_1sw_2 \wedge not_contains(w_1, A) \wedge s, w_1, w_2 \in V^*$$

For example, the rule $r : A \rightarrow_l s$ will change the following words as shown below:

$Aw_1 \Rightarrow sw_1$: substitutes the left-most occurrence of non-terminal A, which is the left-most non-terminal.

$BAw_1 \Rightarrow Bsw_1$: substitutes the left-most occurrence of non-terminal A, although non-terminal B is on its left.

$cdAw_1 \Rightarrow cdsw_1$: there are terminals to the left of A.

$cdA_1w_1A_2 \Rightarrow cdsw_1A_2$: only the first instance of A is substituted.

From context free grammars to PNEPs

The improved PNEP is built from the grammar in the following way:

1. We assume that each derivation rule in the grammar has a unique index that can be used to reconstruct the derivation tree.
2. There is a node for each non terminal (*deriving* nodes) that applies to its strings all the derivation rules for its left-most non terminal.
3. There is an additional node (*discarding* node) which discards non promising sentential forms. It receives all the sentential forms generated and sends to the net those that just contain non terminal symbols or terminals which are also contained in the input string.
4. The *deriving* nodes are connected to the *discarding* and output nodes only.
5. There is an output node, in which the *parsed string* can be found: this is a version of the input, enriched with information that will make it possible to reconstruct the derivation tree (the rules indexes).

4.2.1.1 Formal description

We will now describe the way in which our improved PNEP is defined, starting from a certain grammar. Given the context free grammar $G = \{\Sigma_T = \{t_1, ..., t_n\}, \Sigma_N = \{N_1, ..., N_m\}, A, P\}$ with $A \in \Sigma_N$ its axiom and $P = \{N_i \rightarrow \gamma_j \mid j \in \{1, .., k\}, i \in \{1, ..., n\} \wedge \gamma_j \in (\Sigma_T \cup \Sigma_N)^*\}$ its set of k production rules; the PNEP is defined as

$$\Gamma_G = (V = \Sigma_T \cup \Sigma_N \cup \{1, ..., k\}, node_{output}, N_1, N_2, ..., N_m, N_1^t, N_2^t, ..., N_m^t, G)$$

where

1. N_i is the family of *deriving* nodes. Each node contains the following set of rules: $\{N_i \rightarrow_l \gamma_j\}$ ($\{N_i \rightarrow \gamma_j\}$ are the derivation rules for N_i in G)
2. N^t is the *discarding* node. As it was previously described it only contains the deletion rule $\rightarrow$
3. $node_{output}$ is the output node
4. G is a graph that contains an edge for
   - Each couple (N_i, $node_{output}$)
   - Each couple (N_i, N_i^t)
5. The *input node* A is the only one with a non empty initial content (A)

6. The filters for each node are designed to produce the behavior informally described above. In general, the *deriving* nodes have empty output filters

The principles for deriving non-terminals and stopping the NEP are not different to those explained in the previous section. For the discarding node, PI_{N^t} is a random context filter of type 2, where $P = \{a, b, o, X, S, O\}$ and $F = \emptyset$. The derivation nodes have a random context PI_{N_i} of type 1, where $P = \{N_i\}$ and $F = \emptyset$. Finally, any other filters are designed to accept any word without additional constraints.

The complete PNEP for our example ($\Gamma_{a^n b^n o^m}$) is defined as follows:

- Alphabet $V = \{X, O, S, a, b, o, 1, 2, 3, 4, 5, 6\}$
- Nodes
  - $node_{output}$:
    - $A_{output} = \emptyset$ is the initial content;
    - $M_{output} = \emptyset$ is the set of rules;
    - $PI_{output} = \{$ (regular expression membership filter);
    - $\{\{1,2,3,4,5,6\} * a\{1,2,3,4,5,6\} * a\{1,2,3,4,5,6\} * b\{1,2,3,4,5,6\} * b\{1,2,3,4,5,6\} * o\}\}$;
    - $PO_{output} = \emptyset$ is the output filter
  - N_X:
    - $A_X = \{X\}$;
    - $M_X = \{X \rightarrow_l 1SO\}$;
    - $PI_X = \{P = \{X\}, F = \emptyset\}$;
    - $PO_X = \emptyset$
  - N_S:
    - $A_S = \emptyset$;
    - $M_S = \{S \rightarrow_l 2aSb, S \rightarrow_l 3ab\}$;
    - $PI_S = \{P = \{S\}, F = \emptyset\}$;
    - $PO_S = \emptyset$
  - N_O:
    - $A_O = \emptyset$;
    - $M_O = \{O \rightarrow_l 4oO, O \rightarrow_l 5Oo, O \rightarrow_l 5o\}$;
    - $PI_O = \{P = \{O\}, F = \emptyset\}$;
    - $PO_O = \emptyset$
  - N^t:
    - $A = \{\}$;
    - $M = \{\rightarrow\}$;
    - $PI = \{P = \{X, O, S, a, b, o\}, F = \emptyset\}$;
    - $PO = \{F = \emptyset, P = \emptyset\}$
  - Its graph contains and an edge for each couple $\{(N_X, N^t)$, (N_S, N^t), (N_O, N^t), $(N_X, node_{output})$, $(N_S, node_{output})$, $(N_O, node_{output})\}$
  - It stops the computation when some string enters $node_{output}$

The previous section's examples for the input string *aboo* also applies to this improved NEP. The only new feature is a decrease of the strings amount during simulation.

4.2.1.2 jNEP description of PNEPs

We have added new *jNEP* xml description for the new sort of rule applied to the left-most non terminal. The syntax is the following:

```
<RULE ruleType="leftMostParsing" symbol="NON-TERMINAL" string="SUBSTITUTION_STRING"
nonTerminals="GRAMMAR_NON-TERMINALS"/>
```

Three of the sections of the xml representation of the PNEP $\Gamma_{a^n b^n o^m}$ previously defined (the output node, the *deriving* node for axiom X and the *discarding* node) are shown below.

```
<NODE initCond="">
   <EVOLUTIONARY_RULES>
   <RULE ruleType="deletion" actionType="RIGHT" symbol=""/>
   </EVOLUTIONARY_RULES>
   <FILTERS>
     <INPUT type="RegularLangMembershipFilter"
      regularExpression="[1-6]*a[1-6]*b[1-6]*o[1-6]*o"/>
     <OUTPUT type="1" permittingContext="" forbiddingContext="a_b_o_o"/>
   </FILTERS>
 </NODE>

 <NODE initCond="X">
   <EVOLUTIONARY_RULES>
     <RULE ruleType="leftMostParsing" symbol="X" string="1_S_O" nonTerminals="S_O_X"/>
   </EVOLUTIONARY_RULES>
   <FILTERS>
     <INPUT type="1" permittingContext="X" forbiddingContext=""/>
   </FILTERS>
 </NODE>


 <NODE initCond="">
   <EVOLUTIONARY_RULES>
     <RULE ruleType="deletion" actionType="RIGHT" symbol=""/>
   </EVOLUTIONARY_RULES>
   <FILTERS>
     <INPUT type="2" permittingContext="a_b_o_o_0_1_2_3_4_5_S_O_X" forbiddingContext=""/>
   </FILTERS>
 </NODE>
```

The nodes for other non terminal symbols are similar, but with an empty ("") *initial condition* and their corresponding derivation rules.

4.2.2 On PNEP temporal complexity

We have previously stated that PNEP temporal complexity is linear. We can get to this conclusion if we consider the parsing trees at the end of computation. Assuming a well-formed grammar (without lambda rules, re-writing rules or other superfluous elements), the worst case in terms of temporal complexity is a parsing tree which is a perfect binary tree (a full binary tree in which all leaves are at the same level). Any other parsing tree has less nodes and, therefore, it would require less NEP steps to be formed. We should remember that PNEP expands one node each communication-evolutionary step. In other words, increasing the number of nodes by a factor of two at each level is the slowest way to reach to a number of n leaves, where n is the length of the analyzed string. Hence, given such a tree, the number of nodes is $2n-1$, which can be easily concluded by an intuitive inductive reasoning. More formally, the size of the tree follows the geometric series $1+2+4+8+16...$ or $ar^0+ar^1+ar^2+ar^3+...+ar^d...$ where $a=1, r=2$ and d stands for the tree depth minus one. In that case, the well known formula for a geometric series tells us that the sum is equal to $\frac{a(1-r^{d+1})}{1-r}$, in our case $2^{d+1}-1$. At this point, we must remember that the depth of the tree is, in terms of number of leaves, $\log_2(n)+1$. Thus, the sum is equal to $2^{\log_2(n)+1}-1=2n-1$.

Therefore, under the NEP computational model, our PNEP takes time $O(n)$ for parsing a string of length n in the worst case. It is worth mentioning that the Early algorithm under the conventional Turing machine computational model takes time $O(n^3)$ for the worst case. In addition, PNEP parses in parallel any possible parsing tree for the target string and, thus, gives as output all possible parsing trees.

4.3 Natural language parsing with PNEPs

Syntactic analysis is one of the classical problems related to language processing, and applies to both *artificial* and *natural* languages. There is a wide range of parsing tools that computer scientists and linguists can use [Jurafsky and Martin, 2000].

The characteristics of the particular language determine the suitability of the parsing technique. Two of the main differences between natural and formal languages are ambiguity and the size of the required representation. Ambiguity introduces many difficulties to parsing, therefore programming languages are usually designed to be unambiguous. On the other hand, ambiguity is an almost implicit characteristic of natural languages. To compare the size of different representations, the same formalism should be used. Context-free grammars are widely used to describe the syntax of languages. It is possible to informally compare the sizes of context free grammars for some programming languages and for some natural languages. We conjecture that the representations needed for parsing natural languages are frequently greater than those we can use for high level imperative programming languages.

Furthermore, parsing techniques for programming languages usually restrict the representation (grammar) used in different ways: it must be unambiguous, recursion is restricted, erasing rules must be removed, they must be written in a normal form, etc. These conditions mean extra work for the grammar designer, difficult to understand for non-experts in the field of formal languages. This may be one of the reasons why formal representations such as grammars are seldomly used or even unpopular. Moreover, natural languages usually do not fulfill these constraints.

This work is focused on new computational models to increase the efficiency of parsing for languages with non-restricted context free grammars. In this way, our approach will be applicable at the same time for natural and formal languages. The most important point in this context is the way we parse those grammars' strings or, in terms of natural language, the way we syntactically analyzed our sentences.

Formal parsing techniques for natural languages mainly face inefficiency, grammar size and ambiguity problems. The complexity of the grammars used for syntactic parsing depends on the desired target. Thus, these kinds of grammars are usually very complex, which makes them one of the bottlenecks in NLP tasks. Because of that, the length of the sentences that these techniques are able to parse is usually small (usually less than a typical computer program) or they provide a partial solution called shallow parsing. Shallow parsing will be described later. It is a parsing technique frequently used in natural language processing to overcome the inefficiency of other approaches to syntactic analysis.

The author has previously proposed PNEPs: an extension to NEPs that makes them suitable for efficient parsing of any kind of context free grammars, especially applicable to those languages that share characteristics with natural languages (inherent ambiguity, for example). In the following sections, we will give an example of PNEPs working with a natural language oriented grammar. Later on, we will modify and use PNEPs for shallow parsing so as to compare our approach to a current technique in natural language parsing.

Finally, the author has contributed to the developement of IBERIA [Porta et al., 2011], a corpus of scientific Spanish which is able to process the sentences at the morphological level. We are very interested in adding syntactic analysis tools to IBERIA. The current contribution has this as its long-term goal .

4.3.1 An example

We will use the grammar deduced from the following derivation rules (whose axiom is the non terminal Sentence). This grammar is similar to those devised by other authors in previous attempts to use NEPs for parsing (natural) languages [Bel Enguix et al., 2009]. We have added the index of the derivation rules, that will be used later.

Sentence	$\rightarrow$ (0-0)	NounPhraseStandard PredicateStandard
	| (0-1)	NounPhrase3Singular Predicate3Singular
NounPhrase3Singular	$\rightarrow$ (1-0)	DeterminantAn VowelNounSingular
	| (1-1)	DeterminantSingular NounSingular
	| (1-2)	Pronoun3Singular
NounPhraseStandard	$\rightarrow$ (2-0)	DeterminantPlural NounPlural
	| (2-1)	PronounNo3Singular
NounPhrase	$\rightarrow$ (3-0)	NounPhraseStandard
	| (3-1)	NounPhrase3Singular
PredicateStandard	$\rightarrow$ (4-0)	VerbStandard NounPhrase
Predicate3Singular	$\rightarrow$ (5-0)	Verb3Singular NounPhrase
DeterminantSingular	$\rightarrow$ (6-0)	a
	| (6-1)	the
	| (6-2)	this
DeterminantAn	$\rightarrow$ (7-0)	an
VowelNounSingular	$\rightarrow$ (8-0)	apple
NounSingular	$\rightarrow$ (9-0)	boy
Pronoun3Singular	$\rightarrow$ (10-0)	it
	| (10-1)	she
	| (10-2)	he
DeterminantPlural	$\rightarrow$ (11-0)	these
	| (11-1)	several
	| (11-2)	the
NounPlural	$\rightarrow$ (12-0)	boys
	| (12-1)	apples
PronounNo3Singular	$\rightarrow$ (13-0)	I
	| (13-1)	you
	| (13-2)	we
	| (13-3)	they
VerbStandard	$\rightarrow$ (14-0)	eat
Verb3Singular	$\rightarrow$ (15-0)	eats

It is worth noticing that this grammar is very simple. Nevertheless, NLP syntax parsing usually takes as input the results of the morphological analysis. In this way, the previous grammar can be simplified by removing the derivation rules for the last 9 non terminals (from DeterminantSingular to Verb3Singular): those symbols become terminals for the new grammar.

Notice, also, that this grammar implements grammatical agreement by means of context free rules. For each non terminal, we had to use several different *specialized versions*. For instance, NounPhraseStandard and NounPhrase3Singular are specialized versions of non terminal NounPhrase.

We can build the PNEP associated with this context free grammar by following the steps described in section 4.2. Below, the jNEP configuration file is presented:

```
<?xml version="1.0"?>
<NEP nodes="18">
  <GRAPH>
    <EDGE vertex1="0" vertex2="17"/>
    <EDGE vertex1="0" vertex2="16"/>
    <EDGE vertex1="1" vertex2="17"/>
    <EDGE vertex1="1" vertex2="16"/>
    <EDGE vertex1="2" vertex2="17"/>
    <EDGE vertex1="2" vertex2="16"/>
    <EDGE vertex1="3" vertex2="17"/>
    <EDGE vertex1="3" vertex2="16"/>
    <EDGE vertex1="4" vertex2="17"/>
    <EDGE vertex1="4" vertex2="16"/>
    <EDGE vertex1="5" vertex2="17"/>
    <EDGE vertex1="5" vertex2="16"/>
    <EDGE vertex1="6" vertex2="17"/>
    <EDGE vertex1="6" vertex2="16"/>
    <EDGE vertex1="7" vertex2="17"/>
    <EDGE vertex1="7" vertex2="16"/>
    <EDGE vertex1="8" vertex2="17"/>
    <EDGE vertex1="8" vertex2="16"/>
    <EDGE vertex1="9" vertex2="17"/>
    <EDGE vertex1="9" vertex2="16"/>
```

```
  <EDGE vertex1="10" vertex2="17"/>
  <EDGE vertex1="10" vertex2="16"/>
  <EDGE vertex1="11" vertex2="17"/>
  <EDGE vertex1="11" vertex2="16"/>
  <EDGE vertex1="12" vertex2="17"/>
  <EDGE vertex1="12" vertex2="16"/>
  <EDGE vertex1="13" vertex2="17"/>
  <EDGE vertex1="13" vertex2="16"/>
  <EDGE vertex1="14" vertex2="17"/>
  <EDGE vertex1="14" vertex2="16"/>
  <EDGE vertex1="15" vertex2="17"/>
  <EDGE vertex1="15" vertex2="16"/>
</GRAPH>
<EVOLUTIONARY_PROCESSORS>
  <NODE initCond="Sentence" id="0">
    <EVOLUTIONARY_RULES>
      <RULE ruleType="substitution" actionType="ANY" symbol="Sentence" newSymbol="0-0_NounPhraseStandard_PredicateStandard"/>
      <RULE ruleType="substitution" actionType="ANY" symbol="Sentence" newSymbol="0-1_NounPhrase3Singular_Predicate3Singular"/>
    </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="1" permittingContext="Sentence" forbiddingContext=""/>
    </FILTERS>
  </NODE>
  <NODE initCond="" id="1">
    <EVOLUTIONARY_RULES>
      <RULE ruleType="substitution" actionType="ANY" symbol="NounPhrase3Singular"
                                                     newSymbol="1-0_DeterminantAn_VowelNounSingular"/>
      <RULE ruleType="substitution" actionType="ANY" symbol="NounPhrase3Singular"
                                                     newSymbol="1-1_DeterminantSingular_NounSingular"/>
      <RULE ruleType="substitution" actionType="ANY" symbol="NounPhrase3Singular" newSymbol="1-2_Pronoun3Singular"/>
    </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="1" permittingContext="NounPhrase3Singular" forbiddingContext=""/>
    </FILTERS>
  </NODE>
  <NODE initCond="" id="2">
    <EVOLUTIONARY_RULES>
      <RULE ruleType="substitution" actionType="ANY" symbol="NounPhraseStandard" newSymbol="2-0_DeterminantPlural_NounPlural"/>
      <RULE ruleType="substitution" actionType="ANY" symbol="NounPhraseStandard" newSymbol="2-1_PronounNo3Singular"/>
    </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="1" permittingContext="NounPhraseStandard" forbiddingContext=""/>
    </FILTERS>
  </NODE>
  <NODE initCond="" id="3">
    <EVOLUTIONARY_RULES>
      <RULE ruleType="substitution" actionType="ANY" symbol="NounPhrase" newSymbol="3-0_NounPhraseStandard"/>
      <RULE ruleType="substitution" actionType="ANY" symbol="NounPhrase" newSymbol="3-1_NounPhrase3Singular"/>
    </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="1" permittingContext="NounPhrase" forbiddingContext=""/>
    </FILTERS>
  </NODE>
  <NODE initCond="" id="4">
    <EVOLUTIONARY_RULES>
      <RULE ruleType="substitution" actionType="ANY" symbol="PredicateStandard" newSymbol="4-0_VerbStandard_NounPhrase"/>
    </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="1" permittingContext="PredicateStandard" forbiddingContext=""/>
    </FILTERS>
  </NODE>
  <NODE initCond="" id="5">
    <EVOLUTIONARY_RULES>
      <RULE ruleType="substitution" actionType="ANY" symbol="Predicate3Singular" newSymbol="5-0_Verb3Singular_NounPhrase"/>
    </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="1" permittingContext="Predicate3Singular" forbiddingContext=""/>
    </FILTERS>
  </NODE>
  <NODE initCond="" id="6">
    <EVOLUTIONARY_RULES>
      <RULE ruleType="substitution" actionType="ANY" symbol="DeterminantSingular" newSymbol="6-0_a"/>
      <RULE ruleType="substitution" actionType="ANY" symbol="DeterminantSingular" newSymbol="6-1_the"/>
      <RULE ruleType="substitution" actionType="ANY" symbol="DeterminantSingular" newSymbol="6-2_this"/>
    </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="1" permittingContext="DeterminantSingular" forbiddingContext=""/>
    </FILTERS>
  </NODE>
  <NODE initCond="" id="7">
```

```
    <EVOLUTIONARY_RULES>
      <RULE ruleType="substitution" actionType="ANY" symbol="DeterminantAn" newSymbol="7-0_an"/>
    </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="1" permittingContext="DeterminantAn" forbiddingContext=""/>
    </FILTERS>
  </NODE>
  <NODE initCond="" id="8">
    <EVOLUTIONARY_RULES>
      <RULE ruleType="substitution" actionType="ANY" symbol="VowelNounSingular" newSymbol="8-0_apple"/>
    </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="1" permittingContext="VowelNounSingular" forbiddingContext=""/>
    </FILTERS>
  </NODE>
  <NODE initCond="" id="9">
    <EVOLUTIONARY_RULES>
      <RULE ruleType="substitution" actionType="ANY" symbol="NounSingular" newSymbol="9-0_boy"/>
    </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="1" permittingContext="NounSingular" forbiddingContext=""/>
    </FILTERS>
  </NODE>
  <NODE initCond="" id="10">
    <EVOLUTIONARY_RULES>
      <RULE ruleType="substitution" actionType="ANY" symbol="Pronoun3Singular" newSymbol="10-0_it"/>
      <RULE ruleType="substitution" actionType="ANY" symbol="Pronoun3Singular" newSymbol="10-1_she"/>
      <RULE ruleType="substitution" actionType="ANY" symbol="Pronoun3Singular" newSymbol="10-2_he"/>
    </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="1" permittingContext="Pronoun3Singular" forbiddingContext=""/>
    </FILTERS>
  </NODE>
  <NODE initCond="" id="11">
    <EVOLUTIONARY_RULES>
      <RULE ruleType="substitution" actionType="ANY" symbol="DeterminantPlural" newSymbol="11-0_these"/>
      <RULE ruleType="substitution" actionType="ANY" symbol="DeterminantPlural" newSymbol="11-1_several"/>
      <RULE ruleType="substitution" actionType="ANY" symbol="DeterminantPlural" newSymbol="11-2_the"/>
    </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="1" permittingContext="DeterminantPlural" forbiddingContext=""/>
    </FILTERS>
  </NODE>
  <NODE initCond="" id="12">
    <EVOLUTIONARY_RULES>
      <RULE ruleType="substitution" actionType="ANY" symbol="NounPlural" newSymbol="12-0_boys"/>
      <RULE ruleType="substitution" actionType="ANY" symbol="NounPlural" newSymbol="12-1_apples"/>
    </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="1" permittingContext="NounPlural" forbiddingContext=""/>
    </FILTERS>
  </NODE>
  <NODE initCond="" id="13">
    <EVOLUTIONARY_RULES>
      <RULE ruleType="substitution" actionType="ANY" symbol="PronounNo3Singular" newSymbol="13-0_I"/>
      <RULE ruleType="substitution" actionType="ANY" symbol="PronounNo3Singular" newSymbol="13-1_you"/>
      <RULE ruleType="substitution" actionType="ANY" symbol="PronounNo3Singular" newSymbol="13-2_we"/>
      <RULE ruleType="substitution" actionType="ANY" symbol="PronounNo3Singular" newSymbol="13-3_they"/>
    </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="1" permittingContext="PronounNo3Singular" forbiddingContext=""/>
    </FILTERS>
  </NODE>
  <NODE initCond="" id="14">
    <EVOLUTIONARY_RULES>
      <RULE ruleType="substitution" actionType="ANY" symbol="VerbStandard" newSymbol="14-0_eat"/>
    </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="1" permittingContext="VerbStandard" forbiddingContext=""/>
    </FILTERS>
  </NODE>
  <NODE initCond="" id="15">
    <EVOLUTIONARY_RULES>
      <RULE ruleType="substitution" actionType="ANY" symbol="Verb3Singular" newSymbol="15-0_eats"/>
    </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="1" permittingContext="Verb3Singular" forbiddingContext=""/>
    </FILTERS>
  </NODE>
  <NODE initCond="">
```

```
      <EVOLUTIONARY_RULES>
        <RULE ruleType="deletion" actionType="RIGHT" symbol=""/>
      </EVOLUTIONARY_RULES>
      <FILTERS>
        <INPUT type="2" permittingContext="the_boy_eats_an_apple_0-0_0-1_1-0_1-1_1-2_2-0_2-1_3-0_3-1_4-0_5-0_6-0
                                          _6-1_6-2_7-0_8-0_9-0_10-0_10-1_10-2_11-0_11-1_11-2_12-0_12-1_13-0_13-1
                                          _13-2_13-3_14-0_15-0_NounPhrase_VerbStandard_Pronoun3Singular
                                          _NounPhrase3Singular_DeterminantPlural_DeterminantAn_Verb3Singular
                                          _NounPhraseStandard_NounSingular_PronounNo3Singular_Sentence
                                          _DeterminantSingular_VowelNounSingular_Predicate3Singular_NounPlural
                                          _PredicateStandard" forbiddingContext=""/>
      </FILTERS>
    </NODE>
    <NODE initCond="">
      <EVOLUTIONARY_RULES>
        <RULE ruleType="deletion" actionType="RIGHT" symbol=""/>
      </EVOLUTIONARY_RULES>
      <FILTERS>
        <INPUT type="RegularLangMembershipFilter"
               regularExpression="[0-9\-]*(the)[0-9\-]*(boy)[0-9\-]*(eats)[0-9\-]*(an)[0-9\-]*(apple)"/>
        <OUTPUT type="1" permittingContext="" forbiddingContext="the_boy_eats_an_apple"/>
      </FILTERS>
    </NODE>
  </EVOLUTIONARY_PROCESSORS>
  <STOPPING_CONDITION>
    <CONDITION type="NonEmptyNodeStoppingCondition" nodeID="17"/>
  </STOPPING_CONDITION>
</NEP>
```

Let us consider the English sentence *the boy eats an apple* Some of the strings generated by the nodes of the PNEP in succesive communication steps while parsing this string are shown below (we show the initials, rather than the full name of the symbols). A left derivation of the string is highlighted:

- { S } ⇒
- { ..., 0-1 NPh3S P3S, ... } ⇒
- { ..., 0-1 1-1 DS NS P3S, ... } ⇒
- { ..., 0-1 1-1 6-1 the NS P3S, ... } ⇒
- { ..., 0-1 1-1 6-1 the 9-0 boy P3S, ... } ⇒
- { ..., 0-1 1-1 6-1 the 9-0 boy 5-0 V3S NPh, ... } ⇒
- { ..., 0-1 1-1 6-1 the 9-0 boy 5-0 15-0 eats NPh, ... } ⇒
- { ..., 0-1 1-1 6-1 the 9-0 boy 5-0 15-0 eats 3-1 NPh3S, ... } ⇒
- { ..., 0-1 1-1 6-1 the 9-0 boy 5-0 15-0 eats 3-1 1-0 DA VNS, ... } ⇒
- { ..., 0-1 1-1 6-1 the 9-0 boy 5-0 15-0 eats 3-1 1-0 7-0 an VNS, ... } ⇒
- { ..., 0-1 1-1 6-1 the 9-0 boy 5-0 15-0 eats 3-1 1-0 7-0 an 8-0 apple, ... }

The following fragments of the jNEP output for this case show with more detail the contents of some nodes of the PNEP during its execution.

Notice that:

- Node 16 is the *discarding* node, node 17 is the output node and the rest are the *deriving* nodes.
- The indexes of the rules added to the string in order to build the derivation tree include two numbers:
  1. The first one identifies their non terminal
  2. The second identifies the right hand side

For example, index 1-8 refers to the eighth right hand side of the first non terminal.

- The string [...] means that a piece of output is not shown to save space. Comments are also written between square brackets.
- Partial parsing trees that will become the final ouput are pointed by the symbol `[!]`.

```
***************              NEP INITIAL CONFIGURATION              ***************
--- Evolutionary Processor 0 ---
Sentence
--- Evolutionary Processor 1 ---

--- Evolutionary Processor 2 ---
[...]
--- Evolutionary Processor 16 ---

--- Evolutionary Processor 17 ---


*************** NEP CONFIGURATION - EVOLUTIONARY STEP - TOTAL STEPS: 1 ***************
--- Evolutionary Processor 0 ---
0-1_NounPhrase3Singular_Predicate3Singular [!]
0-0_NounPhraseStandard_PredicateStandard
--- Evolutionary Processor 1 ---

--- Evolutionary Processor 2 ---
[...]
--- Evolutionary Processor 16 ---

--- Evolutionary Processor 17 ---


*************** NEP CONFIGURATION - COMMUNICATION STEP - TOTAL STEPS:  2 ***************
--- Evolutionary Processor 0 ---
[...]
--- Evolutionary Processor 15 ---

--- Evolutionary Processor 16 ---
0-1_NounPhrase3Singular_Predicate3Singular [!]
0-0_NounPhraseStandard_PredicateStandard
--- Evolutionary Processor 17 ---


*************** NEP CONFIGURATION - EVOLUTIONARY STEP - TOTAL STEPS: 3 ***************
--- Evolutionary Processor 0 ---
[...]
--- Evolutionary Processor 15 ---

--- Evolutionary Processor 16 ---
0-1_NounPhrase3Singular_Predicate3Singular [!]
0-0_NounPhraseStandard_PredicateStandard
--- Evolutionary Processor 17 ---


*************** NEP CONFIGURATION - COMMUNICATION STEP - TOTAL STEPS:  4 ***************
--- Evolutionary Processor 0 ---

--- Evolutionary Processor 1 ---
0-1_NounPhrase3Singular_Predicate3Singular [!]
--- Evolutionary Processor 2 ---
0-0_NounPhraseStandard_PredicateStandard
--- Evolutionary Processor 3 ---

--- Evolutionary Processor 4 ---
0-0_NounPhraseStandard_PredicateStandard
--- Evolutionary Processor 5 ---
0-1_NounPhrase3Singular_Predicate3Singular [!]
--- Evolutionary Processor 6 ---

--- Evolutionary Processor 7 ---

--- Evolutionary Processor 8 ---

--- Evolutionary Processor 9 ---

--- Evolutionary Processor 10 ---
```

```
--- Evolutionary Processor 11 ---

--- Evolutionary Processor 12 ---

--- Evolutionary Processor 13 ---

--- Evolutionary Processor 14 ---

--- Evolutionary Processor 15 ---

--- Evolutionary Processor 16 ---

--- Evolutionary Processor 17 ---


*************** NEP CONFIGURATION - EVOLUTIONARY STEP - TOTAL STEPS: 5 ***************
[...]
*************** NEP CONFIGURATION - COMMUNICATION STEP - TOTAL STEPS:  6 ***************
[...]
*************** NEP CONFIGURATION - EVOLUTIONARY STEP - TOTAL STEPS: 7 ***************
[...]
*************** NEP CONFIGURATION - COMMUNICATION STEP - TOTAL STEPS:  8 ***************
[...]
*************** NEP CONFIGURATION - EVOLUTIONARY STEP - TOTAL STEPS: 9 ***************
--- Evolutionary Processor 0 ---

--- Evolutionary Processor 1 ---
0-1_1-2_Pronoun3Singular_5-0_Verb3Singular_NounPhrase
0-1_1-0_DeterminantAn_VowelNounSingular_5-0_Verb3Singular_NounPhrase
0-1_1-1_DeterminantSingular_NounSingular_5-0_Verb3Singular_NounPhrase [!]
--- Evolutionary Processor 2 ---
0-0_2-0_DeterminantPlural_NounPlural_4-0_VerbStandard_NounPhrase
0-0_2-1_PronounNo3Singular_4-0_VerbStandard_NounPhrase
--- Evolutionary Processor 3 ---
0-0_NounPhraseStandard_4-0_VerbStandard_3-1_NounPhrase3Singular
0-1_NounPhrase3Singular_5-0_Verb3Singular_3-1_NounPhrase3Singular [!]
0-0_NounPhraseStandard_4-0_VerbStandard_3-0_NounPhraseStandard
0-1_NounPhrase3Singular_5-0_Verb3Singular_3-0_NounPhraseStandard
--- Evolutionary Processor 4 ---
0-0_2-0_DeterminantPlural_NounPlural_4-0_VerbStandard_NounPhrase
0-0_2-1_PronounNo3Singular_4-0_VerbStandard_NounPhrase
--- Evolutionary Processor 5 ---
0-1_1-2_Pronoun3Singular_5-0_Verb3Singular_NounPhrase
0-1_1-0_DeterminantAn_VowelNounSingular_5-0_Verb3Singular_NounPhrase
0-1_1-1_DeterminantSingular_NounSingular_5-0_Verb3Singular_NounPhrase [!]
--- Evolutionary Processor 6 ---
0-1_1-1_6-2_this_NounSingular_Predicate3Singular
0-1_1-1_6-1_the_NounSingular_Predicate3Singular [!]
0-1_1-1_6-0_a_NounSingular_Predicate3Singular
--- Evolutionary Processor 7 ---
0-1_1-0_7-0_an_VowelNounSingular_Predicate3Singular
--- Evolutionary Processor 8 ---
0-1_1-0_DeterminantAn_8-0_apple_Predicate3Singular
--- Evolutionary Processor 9 ---
0-1_1-1_DeterminantSingular_9-0_boy_Predicate3Singular [!]
--- Evolutionary Processor 10 ---
0-1_1-2_10-2_he_Predicate3Singular
0-1_1-2_10-1_she_Predicate3Singular
0-1_1-2_10-0_it_Predicate3Singular
--- Evolutionary Processor 11 ---
0-0_2-0_11-0_these_NounPlural_PredicateStandard
0-0_2-0_11-1_several_NounPlural_PredicateStandard
0-0_2-0_11-2_the_NounPlural_PredicateStandard
--- Evolutionary Processor 12 ---
0-0_2-0_DeterminantPlural_12-0_boys_PredicateStandard
0-0_2-0_DeterminantPlural_12-1_apples_PredicateStandard
--- Evolutionary Processor 13 ---
0-0_2-1_13-3_they_PredicateStandard
0-0_2-1_13-1_you_PredicateStandard
0-0_2-1_13-0_I_PredicateStandard 0-0_2-1_13-2_we_PredicateStandard
--- Evolutionary Processor 14 ---
0-0_NounPhraseStandard_4-0_14-0_eat_NounPhrase
--- Evolutionary Processor 15 ---
0-1_NounPhrase3Singular_5-0_15-0_eats_NounPhrase [!]
--- Evolutionary Processor 16 ---

--- Evolutionary Processor 17 ---
```

```
*************** NEP CONFIGURATION - COMMUNICATION STEP - TOTAL STEPS:  10 ***************
--- Evolutionary Processor 0 ---
[...]
[AT THIS POINT, PARSING TREES WITH INCORRECT TERMINALS HAVE BEEN PRUNED]
--- Evolutionary Processor 16 ---
0-1_1-0_DeterminantAn_VowelNounSingular_5-0_Verb3Singular_NounPhrase
0-0_2-0_DeterminantPlural_NounPlural_4-0_VerbStandard_NounPhrase
0-1_NounPhrase3Singular_5-0_Verb3Singular_3-1_NounPhrase3Singular [!]
0-0_NounPhraseStandard_4-0_VerbStandard_3-0_NounPhraseStandard
0-0_2-1_PronounNo3Singular_4-0_VerbStandard_NounPhrase
0-1_NounPhrase3Singular_5-0_15-0_eats_NounPhrase [!]
0-0_2-0_11-2_the_NounPlural_PredicateStandard
0-1_1-0_7-0_an_VowelNounSingular_Predicate3Singular
0-0_NounPhraseStandard_4-0_VerbStandard_3-1_NounPhrase3Singular
0-1_1-2_Pronoun3Singular_5-0_Verb3Singular_NounPhrase
0-1_1-1_DeterminantSingular_9-0_boy_Predicate3Singular [!]
0-1_1-0_DeterminantAn_8-0_apple_Predicate3Singular
0-1_1-1_DeterminantSingular_NounSingular_5-0_Verb3Singular_NounPhrase [!]
0-1_1-1_6-1_the_NounSingular_Predicate3Singular [!]
0-1_NounPhrase3Singular_5-0_Verb3Singular_3-0_NounPhraseStandard
--- Evolutionary Processor 17 ---


*************** NEP CONFIGURATION - EVOLUTIONARY STEP - TOTAL STEPS: 11 ***************
--- Evolutionary Processor 0 ---
[...]
--- Evolutionary Processor 16 ---
0-1_1-0_DeterminantAn_VowelNounSingular_5-0_Verb3Singular_NounPhrase
0-0_2-0_DeterminantPlural_NounPlural_4-0_VerbStandard_NounPhrase
0-1_NounPhrase3Singular_5-0_Verb3Singular_3-1_NounPhrase3Singular [!]
0-0_NounPhraseStandard_4-0_VerbStandard_3-0_NounPhraseStandard
0-0_2-1_PronounNo3Singular_4-0_VerbStandard_NounPhrase
0-1_NounPhrase3Singular_5-0_15-0_eats_NounPhrase [!]
0-0_2-0_11-2_the_NounPlural_PredicateStandard
0-1_1-0_7-0_an_VowelNounSingular_Predicate3Singular
0-0_NounPhraseStandard_4-0_VerbStandard_3-1_NounPhrase3Singular
0-1_1-2_Pronoun3Singular_5-0_Verb3Singular_NounPhrase
0-1_1-1_DeterminantSingular_9-0_boy_Predicate3Singular [!]
0-1_1-0_DeterminantAn_8-0_apple_Predicate3Singular
0-1_1-1_DeterminantSingular_NounSingular_5-0_Verb3Singular_NounPhrase [!]
0-1_1-1_6-1_the_NounSingular_Predicate3Singular [!]
0-1_NounPhrase3Singular_5-0_Verb3Singular_3-0_NounPhraseStandard
--- Evolutionary Processor 17 ---


*************** NEP CONFIGURATION - COMMUNICATION STEP - TOTAL STEPS:  12 ***************
--- Evolutionary Processor 0 ---

--- Evolutionary Processor 1 ---
0-0_NounPhraseStandard_4-0_VerbStandard_3-1_NounPhrase3Singular
0-1_NounPhrase3Singular_5-0_Verb3Singular_3-1_NounPhrase3Singular [!]
0-1_NounPhrase3Singular_5-0_15-0_eats_NounPhrase [!]
0-1_NounPhrase3Singular_5-0_Verb3Singular_3-0_NounPhraseStandard
--- Evolutionary Processor 2 ---
0-0_NounPhraseStandard_4-0_VerbStandard_3-1_NounPhrase3Singular
0-0_NounPhraseStandard_4-0_VerbStandard_3-0_NounPhraseStandard
0-1_NounPhrase3Singular_5-0_Verb3Singular_3-0_NounPhraseStandard
--- Evolutionary Processor 3 ---
0-1_1-2_Pronoun3Singular_5-0_Verb3Singular_NounPhrase
0-1_1-0_DeterminantAn_VowelNounSingular_5-0_Verb3Singular_NounPhrase
0-0_2-0_DeterminantPlural_NounPlural_4-0_VerbStandard_NounPhrase
0-1_1-1_DeterminantSingular_NounSingular_5-0_Verb3Singular_NounPhrase [!]
0-0_2-1_PronounNo3Singular_4-0_VerbStandard_NounPhrase
0-1_NounPhrase3Singular_5-0_15-0_eats_NounPhrase [!]
--- Evolutionary Processor 4 ---
0-0_2-0_11-2_the_NounPlural_PredicateStandard
--- Evolutionary Processor 5 ---
0-1_1-1_DeterminantSingular_9-0_boy_Predicate3Singular [!]
0-1_1-0_DeterminantAn_8-0_apple_Predicate3Singular
0-1_1-1_6-1_the_NounSingular_Predicate3Singular [!]
0-1_1-0_7-0_an_VowelNounSingular_Predicate3Singular
--- Evolutionary Processor 6 ---
0-1_1-1_DeterminantSingular_9-0_boy_Predicate3Singular [!]
0-1_1-1_DeterminantSingular_NounSingular_5-0_Verb3Singular_NounPhrase [!]
--- Evolutionary Processor 7 ---
0-1_1-0_DeterminantAn_VowelNounSingular_5-0_Verb3Singular_NounPhrase
0-1_1-0_DeterminantAn_8-0_apple_Predicate3Singular
--- Evolutionary Processor 8 ---
0-1_1-0_DeterminantAn_VowelNounSingular_5-0_Verb3Singular_NounPhrase
0-1_1-0_7-0_an_VowelNounSingular_Predicate3Singular
```

```
--- Evolutionary Processor 9 ---
0-1_1-1_DeterminantSingular_NounSingular_5-0_Verb3Singular_NounPhrase [!]
0-1_1-1_6-1_the_NounSingular_Predicate3Singular [!]
--- Evolutionary Processor 10 ---
0-1_1-2_Pronoun3Singular_5-0_Verb3Singular_NounPhrase
--- Evolutionary Processor 11 ---
0-0_2-0_DeterminantPlural_NounPlural_4-0_VerbStandard_NounPhrase
--- Evolutionary Processor 12 ---
0-0_2-0_DeterminantPlural_NounPlural_4-0_VerbStandard_NounPhrase
0-0_2-0_11-2_the_NounPlural_PredicateStandard
--- Evolutionary Processor 13 ---
0-0_2-1_PronounNo3Singular_4-0_VerbStandard_NounPhrase
--- Evolutionary Processor 14 ---
0-0_NounPhraseStandard_4-0_VerbStandard_3-1_NounPhrase3Singular
0-0_2-0_DeterminantPlural_NounPlural_4-0_VerbStandard_NounPhrase
0-0_NounPhraseStandard_4-0_VerbStandard_3-0_NounPhraseStandard
0-0_2-1_PronounNo3Singular_4-0_VerbStandard_NounPhrase
--- Evolutionary Processor 15 ---
0-1_1-2_Pronoun3Singular_5-0_Verb3Singular_NounPhrase
0-1_1-0_DeterminantAn_VowelNounSingular_5-0_Verb3Singular_NounPhrase
0-1_NounPhrase3Singular_5-0_Verb3Singular_3-1_NounPhrase3Singular
0-1_1-1_DeterminantSingular_NounSingular_5-0_Verb3Singular_NounPhrase [!]
0-1_NounPhrase3Singular_5-0_Verb3Singular_3-0_NounPhraseStandard
--- Evolutionary Processor 16 ---

--- Evolutionary Processor 17 ---


*************** NEP CONFIGURATION - EVOLUTIONARY STEP - TOTAL STEPS: 13 ***************
[...]
*************** NEP CONFIGURATION - COMMUNICATION STEP - TOTAL STEPS:  14 ***************
[...]
*************** NEP CONFIGURATION - EVOLUTIONARY STEP - TOTAL STEPS: 15 ***************
[...]
[...]
[...]
*************** NEP CONFIGURATION - COMMUNICATION STEP - TOTAL STEPS:  38 ***************
--- Evolutionary Processor 0 ---

--- Evolutionary Processor 1 ---

--- Evolutionary Processor 2 ---

--- Evolutionary Processor 3 ---

--- Evolutionary Processor 4 ---

--- Evolutionary Processor 5 ---

--- Evolutionary Processor 6 ---

--- Evolutionary Processor 7 ---

--- Evolutionary Processor 8 ---

--- Evolutionary Processor 9 ---

--- Evolutionary Processor 10 ---

--- Evolutionary Processor 11 ---

--- Evolutionary Processor 12 ---

--- Evolutionary Processor 13 ---

--- Evolutionary Processor 14 ---

--- Evolutionary Processor 15 ---

--- Evolutionary Processor 16 ---
0-1_1-1_6-1_the_9-0_boy_5-0_15-0_eats_3-1_1-0_7-0_an_8-0_apple [!]
0-1_1-0_7-0_an_8-0_apple_5-0_15-0_eats_3-1_1-1_6-1_the_9-0_boy
0-1_1-0_7-0_an_8-0_apple_5-0_15-0_eats_3-1_1-0_7-0_an_8-0_apple
0-1_1-1_6-1_the_9-0_boy_5-0_15-0_eats_3-1_1-1_6-1_the_9-0_boy
--- Evolutionary Processor 17 ---
0-1_1-1_6-1_the_9-0_boy_5-0_15-0_eats_3-1_1-0_7-0_an_8-0_apple [!]


----------------------- NEP has stopped!!! -----------------------
```

```
Stopping condition found: net.e_delrosal.jnep.stopping.NonEmptyNodeStoppingCondition
----------------------------------------------------------------
```

If we analyze now an incorrect sentence, such as *the boy eat an apple*, the PNEP will continue the computation after the steps summarized above, because in this case it is impossible to find a parsed string. As explained in the previous section, it would be easy to modify our PNEP to stop when this circumstance happens. To modify our PNEP to stop when this happens, it is enough to take into account that the length of the input string is a bound for the number of steps needed (it is always possible to get equivalent context free grammars without chaining and lambda rules; in addition, the length of a given string is usually less than the depth of its derivation trees).

4.4 PNEP and shallow parsing: PNEP in a real natural language context

In the following sections, we will introduce FreeLing [Padró et al., 2010], a well-known free platform that offers parsing tools such as a Spanish grammar and shallow parsers for this grammar. Then, we will describe how PNEPs can be used for shallow parsing and describe a jNEP implementation. Finally some examples and conclusions will be given.

4.4.1 Introduction to FreeLing and shallow parsing

We can summarize some of the main difficulties encountered by parsing techniques when building complete parsing trees for natural languages:

- Spatial and temporal performance of the analysis. The Early algorithm and its derivatives [Earley, 1970, Seifert and Fischer, 2004, Zollmann and Venugopal, 2006] are one of the most efficient approaches. They, for example, provide parsing in polynomial time, with respect to the length of the input. Its time complexity for parsing context-free languages is linear in the average case, while in the worst case it is $O(n^2)$ and $O(n^3)$, respectively, for unambiguous and ambiguous grammars. To be more precise, we must say that those time measures correspond to the Early algorithm mechanism to recognize the input. If there is an exponential number of parsing trees, it obviously requires exponential time to return them, although this scenario is not possible in most practical cases.
- The size and complexity of the corresponding grammar, which is, in addition, difficult to design. Natural languages, for instance, are usually ambiguous.

The parsing methods mentioned above raise limitations and efficiency problems when facing a real natural language task. As explained before, natural language grammars are big and very complex due to ambiguity, recursion, lambda rules etc. Therefore, using traditional parsing schemes for context-free grammars turn out to be slow. Furthermore, the design of the grammar itself is a very difficult challenge, because of these difficulties.

Most of the algorithms for syntactic analysis of natural language are actually focused on partial analysis, that is, a parsing alternative called shallow parsing. The goal of shallow parsing is to analyze the main components of the sentences (for example, noun groups, verb groups, etc.) rather than complete sentences. It ignores the actual syntactic structure of the sentences, which are considered as just sets of

these basic blocks. Shallow parsing tries, in this way, to overcome the performance difficulties that arise when building complete derivation trees.

In our context, the main consequence of shallow parsing is that the final result of the process is a sequence of subtrees which is, obviously, an incomplete analysis of the sentence. Most times, the subtrees sequence is presented with a common parent node, which plays the role of being a virtual root node. This addition could make the subtrees sequence seems a complete derivation tree, however it is just a way of replacing the high level structure of the sentence that could not be analyzed.

This way of presenting the results of the analysis can confuse the inexperienced reader, because the final tree is not a real derivation tree: neither its root is the axiom of the grammar nor its branches corresponding to actual derivation rules.

Shallow parsing includes different particular algorithms and tools. Most of them use cascades of finite-state automata [Harris, 1962]. They use finite-state automata to parse basic natural language phrases. Since recursion is a key feature of natural language but, unfortunately, it can not be implemented in a finite-state automata, they include a limited number of *cascades* to permit the same limited number of recursions. They consist of different levels of finite-state automata, where the higher level states can be expanded in lower level finite-state automata. Obviously, the temporal complexity of these algorithms is much better than those based in context-free grammars. FreeLing software [Padró et al., 2010] is a real example of those particular tools.

FreeLing is a suite of language analyzers that provides the scientist with several different tools and techniques. FreeLing includes a context-free grammar of Spanish, adapted for shallow parsing, that does not contain a real axiom. This grammar has almost two hundred non-terminals and approximately one thousand rules. The actual number of rules is even greater, because they use regular expressions rather than terminal symbols. Each rule, in this way, represents a set of rules, depending on the terminal symbols that match the regular expressions.

The terminals of the grammar are *part-of-speech* tags produced by the morphological analysis, not words. Thus, the English word "I" would be firstly morphologically analyzed, resulting in the "PRP" tag (personal pronoun) which is a terminal of the syntactic grammar. So they include labels like "plural adjective", "third person noun", etc. Furthermore, some terminals are actually regular expression, which were included to write the grammar in a more compact way. For example, any adjective, with no person or number specification would be represented as a terminal similar to *A***, where the asterisk stands for all possible persons and number categories. This way, Freeling's grammar avoids listing many different rules which only differ in little morphological aspects from the terminals.

Figure 4.6 shows the output of FreeLing for a very simple sentence like "Él es ingeniero"[1]. FreeLing built three subtrees: two noun phrases and a verb. After that, FreeLing just joins them under the fictitious axiom. Figure 4.4 shows a more complex example.

4.4.2 PNEP extension for shallow parsing

The main difficulty to adapt PNEPs to shallow parsing is the fictitious axiom. PNEPs is designed to handle context free grammars that must have an axiom.

We have also found additional difficulties in the way in which FreeLing reduces the number of needed derivation rules of its grammar. As we have previously introduced, FreeLing uses regular expressions rather than terminal symbols. These kinds of rules actually represents a set of rules: those whose terminals match the regular expressions. We have also added this mechanism to PNEPs in the corresponding

[1] *He is an engineer*

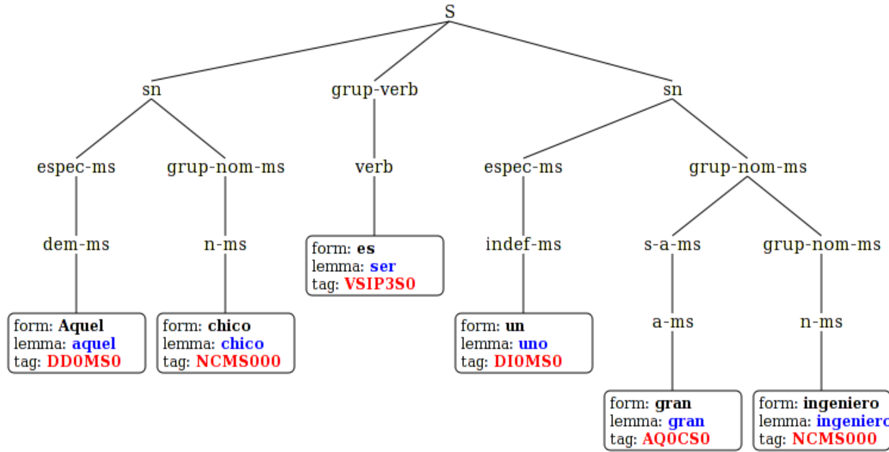


Figure 4.4: FreeLing output for "Aquel chico es un gran ingeniero" (*That guy is a great engineer*)

filters that implement the matching. In the following paragraphs we will explain both problems with more detail.

The virtual root node and the partial derivation trees (for the different components of the sentence) force some changes in the behavior of PNEPs. Firstly, we have to derive many trees at once, one per each constituent, instead of only one tree for the complete sentence. This implies that we need to add many initial strings in the NEP; one per each non-terminal capable of being the root of a constituent. The new initial strings are placed in their corresponding nodes, in other words, in the nodes responsable for deriving the corresponding non-terminal/constituent. Therefore, all the nodes that will apply derivation rules for the nonterminals associated with the components will contain their symbol in the initial step. In section 4.2 the node of the axiom was the only non empty node. In a more formal way:

- Initially, in the original PNEP [Ortega et al., 2009], the only non empty node is associated with the axiom and contains a copy of the axiom. Formally, N_A and Σ_N stand respectively for the node associated with the axiom and the set of nonterminal symbols of the grammar under consideration.

  $I_{N_A} = A$

  $\forall N_i \in \Sigma_N, i \neq A \rightarrow I_{N_i} = \emptyset$

- On the other hand, the initial conditions of the PNEP for shallow parsing are:

  $\forall N_i, I_{N_i} = i$

In this way, the PNEP produces every possible derivation sub-tree beginning from each non-terminal, as if they were axioms of a virtually independent grammar. However, those sub-trees have to be concatenated and, after that, joined to the same parent node (virtual root node of the fictitious axiom). We get this behavior thanks to splicing rules [Choudhary and Krithivasan, 2005, Manea and Mitrana, 2007] in the following way: (1) a special node marks the end and the beginning of the sub-trees with the symbol %, (2) a group of nodes apply splicing rules to concatenate couples of sub-trees, taking the beginning of the first one and the end of the second as the splicing point.

To be more precise, a special node is responsible of the first step. Its specification in jNEP is the following:

```
<NODE initCond="">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="insertion" actionType="RIGHT" symbol="%"/>
    <RULE ruleType="insertion" actionType="LEFT" symbol="%"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="2" permittingContext="SET_OF_VALID_TERMINALS"
                    forbiddingContext=""/>
    <OUTPUT type="RegularLangMembershipFilter"
            regularExpression="%%.*|%.*%|.*%%"/>
  </FILTERS>
</NODE>
```

During the second step a special sub-NEP with splicing rules concatenate the sub-trees. We could choose a specialized node (just one node) or a set of nodes depending on the degree of parallelism we prefer. The needed splicing rule could be defined as follows:

```
<RULE ruleType="splicingChoudhary" wordX="terminal1" wordY="%"
                                   wordU="%" wordV="terminal2"/>
```

Where *terminal2* follows *terminal1* in the analyzed sentence at one or more points. It should be remembered that % marks the end and beginning of the derivation trees. If the sentence has n words, there are n-1 rules/points for concatenation. It is important to note that only splicing rules that create a valid sub-sentence are actually concatenated. [2]

For example, if the sentence to parse is a_b_c_d, we would need the following rules:

```
<RULE ruleType="splicingChoudhary" wordX="a" wordY="%"
                                   wordU="%" wordV="b"/>
<RULE ruleType="splicingChoudhary" wordX="b" wordY="%"
                                   wordU="%" wordV="c"/>
<RULE ruleType="splicingChoudhary" wordX="c" wordY="%"
                                   wordU="%" wordV="d"/>
```

They could concatenate two sub-sentences like b_c and d, resulting in b_c_d.

Concerning the PNEP's topology, most edges are analogous to the basic PNEP presented in previous sections. Every deriving node is connected to the pruning node and to the node responsable for marking the sub-trees. Finally, every node in the splicing sub-NEP is connected to the marking node and the output node. They are also connected to an auxiliary node which helps to synchronize the concatenation steps by receiving all the concatenations, pruning them and sending back to the splicing nodes for a new concatenation cycle. In summary, this way, during the first steps the sub-trees are created, after that, they pass to the marking node. Later on, all possible concatenations are computed and, finally, the parsing trees enter the ouput node.

4.4.3 Our PNEP for the FreeLing's Spanish grammar

The jNEP configuration file for our PNEP adapted to the FreeLing's grammar is large. It has almost two hundred nodes and some nodes have tens of rules. The entire file is presented in appendix D, we will show, however, some of its details. Let the sentence to be parsed be "Él es ingeniero". The output node has the following definition:

[2] In fact, we are using Choudhary splicing rules [Choudhary and Krithivasan, 2005] with a little modification to ignore the symbols that belong to the trace of the derivation.

```
<NODE initCond="">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="deletion" actionType="RIGHT" symbol=""/>
  </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="RegularLangMembershipFilter"
             regularExpression="%[0-9\-]*(PP3MS000|PP\*)[0-9\-]*(VSIP3S0|VSI\*)
                                 [0-9\-]*(NCMS000|NCMS\*|NCMS00\*)%"/>
      <OUTPUT type="1" permittingContext=""
              forbiddingContext="PP*_PP3MS000_VSI*_VSIP3S0
                                 _NCMS*_NCMS00*_NCMS000"/>
    </FILTERS>
</NODE>
```

We have previously explained that the input sentence includes part-of-speech tags instead of actual Spanish words. This sequence of tags, together with the indexes of the rules that will be used to build the derivation tree, are in the input filter for the output node. We can also see some tags written as regular expressions. We have added this kind of tags because FreeLing uses also regular expressions to reduce the size of the grammar.

As an example, we show the specification of one of the deriving nodes. We can see below that the non-terminal *grup-verb* has many rules, the one with trace ID 70-7 is actually needed to parse our example.

```
<NODE initCond="grup-verb" id="70">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="grup-verb" string="70-0_grup-ve [...]
    <RULE ruleType="leftMostParsing" symbol="grup-verb" string="70-1_grup-ve [...]
    <RULE ruleType="leftMostParsing" symbol="grup-verb" string="70-7_verb" [...]
    [...]
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="grup-verb" forbiddingContext=""/>
  </FILTERS>
</NODE>
```

The output of jNEP is also large. However, we can show at least the main dynamic of the process. Figure 4.5 shows it. Comments between brackets help to understand it.

As jNEP shows, the output node contains more than one derivation tree. We design the PNEP in this way, because ambiguous grammars have more than one possible derivation tree for the same sentence. In that case, our PNEP will produce all the possible derivation trees, while FreeLing is only able to show the most likely. By checking the context-free rules applied, it is easy to realize that figure 4.6 corresponds also to the first output of jNEP running our PNEP for shallow parsing.

4.4.4 Final comments

Formal syntactical analysis techniques for natural languages (LL, LR, Early families, for example) suffer from inefficiency when they try to build derivation trees for complete natural language sentences. Shallow parsing is an approach focused on the basic components of the sentence instead on its complete structure. It is extensively used to overcome performance difficulties. FreeLing is one of the most popular free packages and it includes grammars for different natural languages and shallow parsers for them. Some of the main characteristics of shallow parsing are summarized below:

- It actually builds a set of derivation trees that are shown to the user as if they were children of a fictitious pseudo-axiom that does not belong to the grammar.

Figure 4.5: jNEP output for "Él es ingeniero".

```
***************NEP INITIAL CONFIGURATION***************
--- Evolutionary Processor 0 ---
[THE INITIAL WORD OF EVERY DERIVATION NODE IS ITS CORRESPONDING
NON-TERMINAL IN THE GRAMMAR]
[...]
--- Evolutionary Processor 70 ---
grup-verb
[...]
--- Evolutionary Processor 112 ---
sn
[...]
--- Evolutionary Processor 190 ---
[THE OUTPUT NODE IS EMPTY]
*************** NEP CONFIGURATION - EVOLUTIONARY STEP - TOTAL STEPS: 1 ***************
[FIRST EXPANSION OF THE TREES]
[...]
--- Evolutionary Processor 70 ---
70-6_verb-pass 70-7_verb 70-0_grup-verb_patons_patons_patons[...]
[...]
--- Evolutionary Processor 112 ---
112-104_grup-nom 112-103_grup-nom-ms 112-97_pron-mp 112-95_pron-ns[...]
[...]
*************** NEP CONFIGURATION - COMMUNICATION STEP - TOTAL STEPS:  2 ***************
--- Evolutionary Processor 0 ---
[THE FIRST TREES WITH ONLY TERMINALS APPEAR AT THE BEGINNING OF
SPLICING SUB-NET]
--- Evolutionary Processor 178 ---
57-3_NCMS00* 151-35_VSI* 1-2_PP3MS000 99-0_NCMS* 121-2_VSI*
[...]
[THE REST GO TO THE PRUNING NODE]
--- Evolutionary Processor 189 ---
112-87_psubj-mp_indef-mp 8-3_s-a-ms 44-6_prep_s-a-fp [...]
*************** NEP CONFIGURATION - COMMUNICATION STEP - TOTAL STEPS:  4 ***************
[THE PROCESS OF MARKING THE END AND THE BEGINNING STARTS]
[...]
--- Evolutionary Processor 178 ---
1-2_PP3MS000_% %_151-35_VSI* 57-3_NCMS00*_% %_1-2_PP3MS000 %_99-0_NCMS* 99-0_NCMS*_% 151-35_VSI*_% 121-2_VSI*_%
%_121-2_VSI* %_57-3_NCMS00*
[...]
*************** NEP CONFIGURATION - EVOLUTIONARY STEP - TOTAL STEPS: 7 ***************
[THE SPLICING SUB-NET STARTS TO CONCATENATE THE SUB-TREES]
[...]
--- Evolutionary Processor 178 ---
156-3_1-2_PP3MS000_% 77-13_57-3_NCMS00*_% %_70-7_151-35_VSI* 34-11_99-0_NCMS*_% %_111-4_1-2_PP3MS000
111-4_1-2_PP3MS000_% 70-7_151-35_VSI*_% %_77-13_57-3_NCMS00* %_34-11_99-0_NCMS* %_156-3_1-2_PP3MS000
[...]
--- Evolutionary Processor 187 ---
%_121-2_VSI*_99-0_NCMS*_% %_% %_151-35_VSI*_% %_99-0_NCMS*_% %_121-2_VSI*_% %_151-35_VSI*_99-0_NCMS*_%
--- Evolutionary Processor 188 ---
%_121-2_VSI*_57-3_NCMS00*_% %_151-35_VSI*_57-3_NCMS00*_% %_% %_151-35_VSI*_% %_121-2_VSI*_% %_57-3_NCMS00*_%
[...]
*************** NEP CONFIGURATION - COMMUNICATION STEP - TOTAL STEPS:  18 ***************
[THE OUTPUT NODE RECEIVES THE RIGHT DERIVATION TREE. IT IS THE SAME AS THE ONE OUTPUT BY FREELING]
--- Evolutionary Processor 190 ---
[THE FIRST ONE IS THE OUTPUT DESIRED]
%_112-99_111-4_1-2_PP3MS000_70-7_151-35_VSI*_112-103_77-13_57-3_NCMS00*_% %_1-2_PP3MS000_151-35_VSI*_57-3_NCMS00*_%
[...]
```

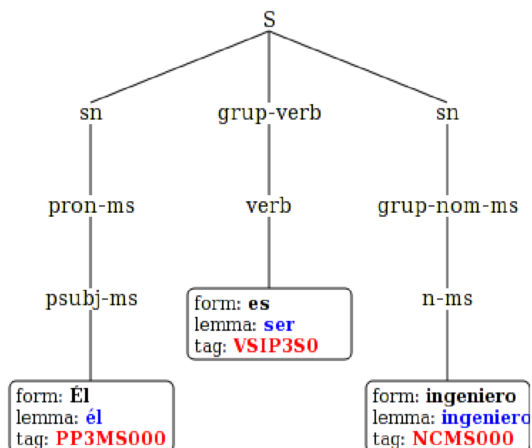


Figure 4.6: Shallow parsing tree for "Él es ingeniero"

- It is not a pure formal technique, so several tricks are frequently used to save resources. One of them is the use of regular expressions instead of terminal symbols. Each rule, in this case, represents the set of rules whose terminals match the regular expressions. The morphological analyzers have also to take into account this kind of matching.

We have added to PNEPs (an extension to NEPs for parsing any kind of context free grammars) some features to deal with these characteristics. We have also added them to jNEP (a NEP simulator written in Java and able to run on parallel platforms). We have also used the FreeLing grammar for Spanish to shallow parse some very simple examples. This way, we have demonstrated that PNEP can be easily adapted to treat with grammars oriented to shallow parsing, which are very common in practical contexts.

However, PNEPs have an important advantage over complete parsing and shallow parsing techniques; it produces all possible parsing trees in linear time, under the assumptions of the model, no matter if the parsing is complete or shallow.

In the future we plan to incorporate syntactical analysis (both, complete and shallow) to IBERIA corpus [Porta et al., 2011] for Scientific Spanish by means of PNEPs. Further on, it would be possible to extend PNEPs with formal representations able to handle semantics (attribute grammars, for example) We also plan to use this new model as a tool for compiler design and as a new approach to tackle some tasks in the semantic level of natural language processing.

Chapter 5

Automatic modelling

5.1 Automatic modelling of habituation

As we discussed previously in section 2.2.2, Grammatical Evolution is a system capable of generating programs in an arbitrary language. However if we have a deeper look at the GE algorithm, it is easy to show that there is no reason to restrict the set of languages to be used in GE to the subset of *programming* languages. In fact, we can make an evolutionary search in any language that can be defined as a context-free grammar or any extension of those presented in chapter 3. Therefore, the target expression in the algorithm can represent not only computer programs, but also any other thing that can be described using the language defined by the formal grammar.

With this simple idea in mind, we can think of using GE to find a model by performing an automatic search within the space of expressions defined by a formal grammar. Before going on, it would be better to disambiguate the concept of *model* and *modelling* by defining what we want to refer when using these words. *Model* can be used with different meanings in many scientific and technical areas, as well as philosophy of science. We will use *Model* in a broad sense, mainly referring to the most common meaning in the practice of science and engineering. From now on, with *Model* we mean a formal description of a system's behaviour and, therefore, the art of modelling is just the task of building an explanation of a system's behaviour and communicating it in a formal language.

Some of those works presented in section 2.2.2.2 made use of GE under a similar perspective to that presented above. Specially, Brabazon et al. [2002a], Brabazon et al. [2002b], Brabazon and O'Neill [2002], Brabazon and O'Neill [2003], Brabazon and O'Neill [2004], O'Neill et al. [2001b], O'Neill et al. [2001a], O'Neill et al. [2002], which are about financial prediction and Moore and Hahn [2003], Moore and Hahn [2004], Moore et al. [2005] where GE is applied to hierarchical Petri net modeling of complex genetic systems and McKinney et al. [2006] where GE helps to identify nonlinear dynamical systems. These studies bring GE outside the boundaries of programming languages and make a search for an expression that represents a model for a real system.

Given these general ideas, in the next section we will define with more details the methodology that we used to find a model of Habituation [del Rosal et al., 2007]. We understand that this methodology can be easily extrapolated to any other modelling context and, thus, it can be study in future works as a general *Automatic Modelling* technique.

5.1.0.1 General prior assumptions and constraints

Following the ideas discussed during the state of the art introduction, our general perspective is the following: Firstly, we understand that habituation, given its filtering role of stimuli and its simplicity, is the most appropriate type of learning to begin with. Secondly, the functionality that provides the habituation process (filtering not relevant stimuli for processing) has to be integrated in the models of a more complex type of learning, with all its details, if we want success in building a model that comprehends all the characteristics needed. Finally, to overcome the difficulties cited, it could be useful to get support from the latest developments in computer science. With this in mind, we will start our search of a fully featured habituation model.

Although our method of modelling is said to be *automatic*, some prior assumptions are needed to at least create the grammar. Those assumptions rest on the knowledge about the system to model that the researcher has previously acquired. In our case, we followed the line of research represented by Alonso et al. [2005a], Alonso et al. [2005b] and del Rosal et al. [2006]. These works present a model of habituation already discussed in Chapter 2. As any other current model of habituation is not completely satisfactory, since it does not fulfil all the main properties of habituation. From some of the general decisions and assumptions made in that line of research, we prepaired our GE system for searching a model of habituation, trying to improve its results.

We pass to enumerate these premises together with other practical constraints. All of them are taken from Alonso et al. [2005a], Alonso et al. [2005b] and del Rosal et al. [2006], except for the last one that is specific to our research's context;

1. As habituation features have to do with time (see 2.4.1.3), it is needed to represent the variables of the model in time and take into account its variation through time. For that purpose we use iterated functions as the formalism of the model, where the independent variable is time. Iterated functions work as follows; based on the specification of an initial condition f(0), each one of their values is always a function of the previous one; in other words, f(t+1) = g(f(t)). In addition, this kind of function is easy to manipulate, interpret and study.

2. Given that this model is intended to be integrated in a more general model of associative learning, it should have the ability to manage more than one stimulus, as well as the ability to manage its interactions. Thus, the architecture has to consider these features by permitting variables of one stimulus affects another stimulus' variables.

3. Each stimulus has 2 variables that describes the influence of each one on the organism's responses and, therefore, control the habituation mechanism. In addition, there is another variable defined for each possible pair of stimuli. They can influence each other, resulting in a highly coupled system. Finally an input variable to the system is needed to represent the advent of the stimulus;

   (a) **Function** $A_i(t)$; It defines the value of variable "A" at the time step "t" for the stimulus "i". It is related to the organism's response to the stimulus. The response's strength is directly proportional to its value.

   (b) **Function** $D_i(t)$; An auxiliary variable. It helps in the task of producing behaviour such as habituation.

   (c) **Function** $T_{ij}(t)$; There exists one variable T_{ij} for each pair of stimuli (i,j). It represents the interactions between stimuli.

(d) **Variable** $S_i(t)$; It is the input of the system. Values above zero means that the stimulus is present. The bigger its value, the bigger the stimulus' intensity.

4. **Initial conditions**; We decided to maintain the same initial conditions as in del Rosal et al. [2006]. $A_i(0) = T_{ij}(0) = 0$ and $D_i(0) = 1$. It makes sense that $A_i(0) = T_{ij}(0) = 0$ since the former represents the response's strength of the organism and the latter stands for an amount of some kind of interaction between "i" and "j". The initial condition of $D_i(0)$ is in some way arbitrary in our context.

5. **Complexity**; Since the model is to be used by the researcher as a representation of the mechanism that produces habituation, it can not be too complex so that the researcher can manipulate it as a tool of prediction and theoretical discussion. Therefore, the sizes of the expressions that defines the functions can not be too big. That constraint was added to the grammar.

It is important to note that in Alonso et al. [2005b], Alonso et al. [2005a] and del Rosal et al. [2006] variable D_i and T_{ij} have a clear conceptualization. D_i stands for the "Availability" of the stimulus' response, while T_{ij} represents the association between stimulus "i" and "j". However, in our case, the role of D_i and T_{ij} is not known at the moment of defining the grammar, but it is created through GE. Thus, we have only assumed the basic characteristics of those variables, without doing any interpretation.

We have followed the scheme of those prior works because we think that it provides a reasonable number of variables and it has all the elements that a model could need to reproduce all the characteristics of habituation and be prepaired to get integrated into big models of more complex learning. Moreover, it made our automatic generated model easily comparable with the last habituation model in the literature. Nevertheless, the number of assumptions and constraints that can be added is somewhat arbitrary. The researcher's needs and practical considerations guide the number and nature of these conditions. On one extreme point, the researcher could decide to leave the GE system search without restriction, only specifying the representation formalism of the model. On the other extreme, the researcher could implement plenty of conditions to the model, limiting the creativity of the evolutionary search.

5.1.0.2 Proposed solution through Grammatical Evolution

After doing the considerations of the previous section, now we need to implement them in the Grammatical Evolution system. This consists of a formal definition of the grammar and the fitness function so that we can implement them in a computer program. With this definition every detail of the *Automatic Modelling* task becomes explicit.

The Christiansen grammar As mentioned before, the model consists of 3 iterated functions $(A_i(t), D_i(t), T_{ij}(t))$. They can contain the arithmetic operators and variables that the grammar will explicit below. Before introducing the formal grammar, we will give some comments on those operators and variables to make the grammar easier to understand.

1. **Functions;** The 3 iterated functions are defined in the following manner; $A_i(t)$ = <arithmetic expression>. Where the expression can contain the operators and variables considered in the grammar. Since they are iterated functions, the variables inside the expression get the value of the previous step

t-1 instead of *t*. For clarity, we do not write the time step specification (t-1) in the expression's variables, it remains implicit.

2. **Notation;** The arithmetic expressions are written in prefix notation. Each element of the expression is separated by a comma just for practical reasons.

3. **Operator symbols and their meanings;**

   (a) Binary operators

   i. **+** → Addition operator.
   ii. **-** → Subtraction operator.
   iii. ***** → Product operator.
   iv. **/** → Division operator.
   v. **max** → Returns the value of the largest operand.
   vi. **min** → Returns the value of the smallest operand.
   vii. **pow** → Exponentiation operator where the first value is the base and the second is the exponent.

   (b) Unary operators

   i. **l** → Natural logarithm operator.
   ii. **abs** → Absolute value operator.
   iii. **int** → Returns the integer part of the operand.
   iv. **sqrt** → Square root operator.
   v. **exp** → Returns the number e raised to the power of the operand.
   vi. **sum<n>** → A summation operator with some special constraints. This operator is explained in detail below.

4. **Variables and indices;** Variables can have two kind of subscript indices. Those referring to the value of the variable for a single stimulus; i and j. And those referring to an array that contains the variable values for every stimuli; indices of type <n>. Any symbol apart from i and j can be used as the variable's index, for example D_k or A_c.

   Given the features of function $T_{ij}(t)$ the variables that can be included in its arithmetic expression vary from those in $A_i(t)$ and $D_i(t)$. In the case of the last two, variables with subscript index i or <n> can be included. On the other hand, variables in $T_{ij}(t)$ can have index i but also j for obvious reasons. To be more explicit;

   (a) **Legal variables for functions** $A_i(t)$ **and** $D_i(t)$ –>

   i. A_i
   ii. D_i
   iii. S_i
   iv. $A_{<n>}$
   v. $D_{<n>}$
   vi. $S_{<n>}$
   vii. $T_{i<n>}$

   (b) **Legal variables for function** $T_{ij}(t)$ –>

   i. All the legal variables for $A_i(t)$ and $D_i(t)$.
   ii. A_j
   iii. D_j

iv. S_j
v. T_{ij}
vi. $T_{<n>j}$

5. **Summation operator;** The summation operator, introduced earlier, performs an addition over all the values in an array. In practice, an addition over all the values of a variable with index of type <n> (see previous point "*Variables and indices*). For example, the expression sum_k, D_k evaluates to the summation of every stimuli's D. The more complex expression $sum_k, D_i, sum_l, A_k, A_l,$, which in a classic notation is $\sum_{k=0}^{k=n} D_i \sum_{l=0}^{l=n} A[k]A[l]$ (where n is the number of stimuli), has the usual meaning.

6. **Features outside the capabilities of context-free grammars;** We can see below that our grammar is just a context-free grammar except for two elements. These two are easily performed due to the Christiansen grammar's capabilities, which provide the possibility of defining any computational language.

   (a) **Summation indices;** As it is easy to show, a variable with subscript index of type <n> does not make sense outside the scope of a summation operator. To avoid this situation, our Christiansen grammar has rules that produces that kind of variables only while the non-terminal is inside the scope of a summation operator, which is a content-dependent property.

   (b) **Model complexity;** As mentioned in section 5.1.0.1, the complexity of the model should not be too large. To limit the complexity, the Christiansen grammar has an attribute (↑c) that stores the number of operators of each expression. If that number goes over a given threshold, the expression is dismissed (function *tooComplex*(c) in the grammar).

Finally, we present the grammar below. The grammar's notation is the same as the one used in section 2.2.3.

- <Model>(g) ::= A$_i$(t)=<ExpTypeAD>$_A$(↓g) : D$_i$(t)=<ExpTypeAD>$_B$(↓g) : T$_{ij}$(t)=<ExpTypeT>(↓g)
  {

  <ExpTypeAD>$_A$.↓g = <Model>.g

  <ExpTypeAD>$_B$.↓g = <Model>.g

  <ExpTypeT>.↓g = <Model>.g

  }

Rules concerning <ExpTypeT> are analogous to those concerning <ExpTypeAD>. They just differ in the variables they can have.

Below, rules for <ExpTypeAD>;

- <ExpTypeAD>(↓g, ↑c) ::= <BinaryOp>(↓g),<ExpTypeAD>$_A$(↓g, ↑c),<ExpTypeAD>$_B$(↓g, ↑c)
  {

  <BinaryOp>.↓g = <ExpTypeAD>.↓g

  <ExpTypeAD>$_A$.↓g = <ExpTypeAD>.↓g

  <ExpTypeAD>$_B$.↓g = <ExpTypeAD>.↓g

<ExpTypeAD>.↑c = <ExpTypeAD>$_A$.↑c + <ExpTypeAD>$_B$.↑c + 1

tooComplex(<ExpTypeAD>.↑c)

}

- <ExpTypeAD>(↓g, ↑c) ::= <UnaryOp>(↓g),<ExpTypeAD>$_A$(↓g, ↑c)
  {

  <UnaryOp>.↓g = <ExpTypeAD>.↓g

  <ExpTypeAD>$_A$.↓g = <ExpTypeAD>.↓g

  <ExpTypeAD>.↑c = <ExpTypeAD>$_A$.↑c + 1

  tooComplex(<ExpTypeAD>.↑c)

  }

- <ExpTypeAD>(↓g, ↑c) ::= sum<indexAD>(↓g, ↑new_g),<ExpTypeAD>$_A$(↓g, ↑c)
  {

  <indexAD>.↓g = <ExpTypeAD>.↓g

  <ExpTypeAD>$_A$.↓g = <indexAD>.↑new_g

  <ExpTypeAD>.↑c = <ExpTypeAD>$_A$.↑c + 1

  tooComplex(<ExpTypeAD>.↑c)

  }

In the next point, we are inside the scope of a summation operator. The following rule's semantic actions introduce a new rule to the grammar ↑new_g. That rule permits variables with subscript of type <n>, i. e. arrays. We include these kind of variables at this point and not by default in order to avoid array variables outside the scope of a summation operator.

- <indexAD>(↓g, ↑new_g) ::= a | b | c | d | e | f | g | h | k | m | o | p | q | r | s | t | u | w | x | y | z
  {

  rule1 = <ExpTypeAD>(↓g) ::= A$_{<n>(\downarrow g)}$ | D$_{<n>(\downarrow g)}$ | S$_{<n>(\downarrow g)}$ | T$_{i<n>(\downarrow g)}$
  {

  <n>.↓g = <ExpTypeAD>.↓g

  }

  rule2 = <n> ::= *"THE ACTUAL INDEX"*

  <indexAD>.↑new_g = <indexAD>.↓g + rule1 + rule2

  }

- <ExpTypeAD>(↓g, ↑c) ::= <RealNumber>(↓g)
  {

  <RealNumber>.↓g = <ExpTypeAD>.↓g

  <ExpTypeAD>.↑c = 0

  }

- <ExpTypeAD>(↓g, ↑c) ::= $S_i|A_i|D_i$
  {

<ExpTypeAD>.↑c = 0

}

Rules for <ExpTypeT>;

- <ExpTypeT>(↓g, ↑c) ::= <BinaryOp>(↓g),<ExpTypeT>$_A$(↓g, ↑c),<ExpTypeT>$_B$(↓g, ↑c)
  {

  <BinaryOp>.↓g = <ExpTypeT>.↓g

  <ExpTypeT>$_A$.↓g = <ExpTypeT>.↓g

  <ExpTypeT>$_B$.↓g = <ExpTypeT>.↓g

  <ExpTypeT>.↑c = <ExpTypeT>$_A$.↑c + <ExpTypeT>$_B$.↑c + 1

  tooComplex(<ExpTypeT>.↑c)

  }

- <ExpTypeT>(↓g, ↑c) ::= <UnaryOp>(↓g),<ExpTypeT>$_A$(↓g, ↑c)
  {

  <UnaryOp>.↓g = <ExpTypeT>.↓g

  <ExpTypeT>$_A$.↓g = <ExpTypeT>.↓g

  <ExpTypeT>.↑c = <ExpTypeT>$_A$.↑c + 1

  tooComplex(<ExpTypeT>.↑c)

  }

- <ExpTypeT>(↓g, ↑c) ::= sum<indexT>(↓g, ↑new_g),<ExpTypeT>$_A$(↓g, ↑c)
  {

  <indexT>.↓g = <ExpTypeT>.↓g

  <ExpTypeT>$_A$.↓g = <indexT>.↑new_g

  <ExpTypeT>.↑c = <ExpTypeT>$_A$.↑c + 1

  tooComplex(<ExpTypeT>.↑c)

  }

- <indexT>(↓g, ↑new_g) ::= a | b | c | d | e | f | g | h | k | m | o | p | q | r | s | t | u | w | x | y | z
  {

  rule1 = <ExpTypeT>(↓g) ::= A$_{<n>(\downarrow g)}$ | D$_{<n>(\downarrow g)}$ | S$_{<n>(\downarrow g)}$ | T$_{i<n>(\downarrow g)}$ | T$_{<n>j(\downarrow g)}$
  {

  <n>.↓g = <ExpTypeT>.↓g

  }

  rule2 = <n> ::= *"THE ACTUAL INDEX"*

  <indexT>.↑new_g = <indexT>.↓g + rule1 + rule2

  }

- <ExpTypeT>(↓g, ↑c) ::= <RealNumber>(↓g)
  {

<RealNumber>.↓g = <ExpTypeT>.↓g

<ExpTypeT>.↑c = 0

}

- <ExpTypeT>(↓g, ↑c) ::= $S_i|A_i|D_i|S_j|A_j|D_j|T_{ij}$
  {

  <ExpTypeT>.↑c = 0

  }

Non-specific rules;

- <BinaryOp> := + | - | * | / | max | min | pow
- <UnaryOp> ::= l | abs | int | sqrt | exp
- <RealNumber>(↓g) ::= <IntPart>(↓g).<DecPart>(↓g)
  {

  <IntPart>.↓g = <RealNumber>.↓g

  <DecPart>.↓g = <RealNumber>.↓g

  }
- <IntPart> ::= 0 | 1 | 2 | ...... | 8 | 9 | 1<IntPart>$_A$ | 2<IntPart>$_A$ | .... | 9<IntPart>$_A$
  {

  <IntPart>$_A$.↓g = <IntPart>.↓g

  }
- <DecPart> ::= 0 | 1 | 2 | ...... | 8 | 9 | <DecPart>$_A$1 | <DecPart>$_A$2 | .... | <DecPart>$_A$9
  {

  <DecPart>$_A$.↓g = <DecPart>.↓g

  }

Figure 5.1 shows an example derivation tree for the expression

"l,sumb,+,Ab,*,Tib,Di,"

The attribute's computations are not explicitly shown but commented in red.

The fitness function To asses the quality of the models generated by the GE algorithm, we compared the simulated data provided by the models against a set of empirical data. We tried to cover most of the habituation's main properties presented in table 2.4. Below, the list of the empirical data's references and its corresponding property is shown.

1. **Rankin and Broster [1992], experiment 1**; Properties 1, 2, 4 and 7.
2. **Rankin et al. [1990]**; Property 5.

Figure 5.1: Derivation tree for the expression ”l,sumb,+,Ab,*,Tib,Di,”

During the comparison, we understood that the values of $A_i(t)$ represent the response strength of the organism for stimulus "i". To be more specific, we compared $A_i(t)$ values, when the stimulus was presented at t-1 against the response strength values reported by the experiments for that stimulus presentation.

The measure for that comparison was based on the summation of the absolute differences between the empiric and simulated points. Indeed the points used in the comparison were not the direct values but the proportion of the first response strength, following the usual data management in the literature. Since negative or zero values distort proportion calculations and a negative *response strength* does not make sense, simulated point's arrays that report negative values were transformed before calculating its fitness as follows; $P^{Transformed} = P + |\min(P)| + 0.001$, where P stands for the array of simulated points to asses and $\min(P)$ means the minimum value of that vector.

The mathematical expression for the fitness function is;

$$Fitness = \sum_{i=0}^{i=n} \sum_{k=1}^{k=m} \left| \frac{P_i[k]}{P_i[0]} - \frac{E_i[k]}{E_i[0]} \right| * weight(i,k)$$

Leaving apart the *weight* function, E[k] stands for the empirical points' vector. P[k] symbolize the simulated points' vector. The subscript index "i" represents each experiment in the data set. Finally, "n" is the number of experiments, while "m" is the number of points of the given experiment.

The *weight* function was added to increase the influence of some especially important points in the experiments. Every point has a *weight* of 1 by default, except for a few points that strongly characterized habituation. These points are those that represent recovery in Rankin and Broster [1992] (experiment 1), which received a weight of 3, and the one that reflects dishabituation in the experiment taken from Rankin et al. [1990], which received a weight of 6. These values are somehow arbitrary and could be subject to further tuning. Although these singular points are very few considering the whole amount of points, they are very important in the habituation curve. Without the *weight* function, they would not have had an appropriate influence in the fitness function.

5.1.0.3 Experiments

During the preliminary development and testing of the application that performs the evolutionary search, we could see that the problem that we were trying to solve is far from being trivial. We assumed that we would need a preliminary long study on evolutionary parameters tuning before the GE system could find significant habituation models. Moreover, we discovered that the task of evaluating the models' expressions is highly time-consuming, therefore, the parameters tuning and the time efficiency optimization of the algorithm would be key aspects of our research.

With this in mind, we pass to present specific info about the experiments run.

Common parameters to every experiment

- **Population;** 10000 of individuals.
- **Genotypes' length;** All our experiments run with constant genotypes' length. Therefore, operators *Duplication* and *Elision* were off. The constant number of integers or codons was 1000.
- **Wrapping operator;** We established 3 as the maximum number of wrappings.

- **Integers' range;** The integers' range was between 0 and 256
- **Probability of mutation;** 100%, which means that each genotype mutates one of its codons every generation.
- **Probability of crossover;** 100%
- **Termination condition;** Find a model with fitness below 1 or getting the maximum number of generations set.
- **Parent selection;** Fitness proportional, where the worst "n" individuals do not take part in the selection, see below.
- **Survivor selection;** The worst n individuals are eliminated, where $n = population's_size * generational_gap$, see below.

5.1.0.4 Experiments program

During our experiments we tried to find the best combination of evolutionary parameter values, in terms of their influence to reach better fitness values. Only a subset of them varied: mutation, cross-over and the generational gap (in a steady-state population model). The termination condition was the existence of a fitness below 1 (a very small number that reflects an almost perfect matching between the empiric and simulated data) or reaching the maximum number of generations, which is determined by the maximum number of fitness evaluations, an amount between 30000 and 120000 depending on the CPU performance. With this we tried to avoid very long experiments (a normal experiment lasted 4-6 days).

We tried to find the best parameter combination considering the following values: mutation rates of 10%, 50% or 100%, generational gaps of 1% and 50% and cross-over rates of 10%, 50% or 100%. The best combination of values was mutation=50%, cross-over=10% and GG=1%. We ran at least four experiments for each combination. It is worthy to note at this point that, in our program, mutation rate is not defined in the traditional way. Rather than applying mutation to every bit with a given probability, our 50% rate gives the probability that a single codon in the genotype will be mutated. This corresponds to a much smaller rate under the traditional interpretation.

5.1.0.5 Results

The first result that we have to highlight is that the found models reproduce some basic properties of habituation but are not significant if we compare them with those in the literature. As an example, the model of Alonso et al. [2005b] receives a fitness of 29 approximately, on the other hand, none of the found models have a fitness below 33. We will have a deeper look on these models later.

Together with the unsatisfactory fitness performance, we found a big problem in our experiments: most of them stopped after reaching the maximum generation and show that the whole population had converged to a single individual. This behaviour could mean that the algorithm has problems to maintain the desirable diversity in the population during the evolutionary search. As a consequence, that convergence causes the algorithm to get stuck in a local optima.

5.1.0.6 A look on two of the models

As mentioned above, the evolutionary search has not found a model with similar qualities to those of the literature, however, some of the models found are capable of reproducing some of the characteristics presented in section 2.4.1.3. We will focus on

two models that we named *Evolutionary Model A* and *Evolutionary Model B*. Both of them show a simple strength's response decrement as the stimulus is presented, although with a different shape to the one found in the empiric data. Specifically, Model A reproduces spontaneous recovery (again with a different shape) and Model B shows dishabituation. Unfortunately, they neglect the rest of the basic properties of habituation.

To asses the models, we compared them, regarding their principal advantages, with the empiric data and other models in the literature. Model A is compared with the empiric data in figure 5.2, concerning spontaneous recovery of habituation. Figure 5.3 opposed it to Alonso et al. [2005b]'s model. On the other hand, Model B is compared against empiric data and del Rosal et al. [2006] (an improvement of Alonso et al. [2005b] that permits dishabituation behaviour) with regards to an experiment of simple habituation and dishabituation in figures 5.4 and 5.5 respectively.

Their defining expressions are;

- **Evolutionary Model A;**
  A(t)= Di ,

  D(t)= - , Si , * , Di , ln , 2 . 6 7 ,

  T(t)= Ai ,

  Fitness: 35.01259054550772

- **Evolutionary Model B;**

  A(t)= Di ,

  D(t)= - , sum z , * , S z , 4 . 7 , max , Di , sqrt , v , / , Di , sqrt , 3 4 3 . 8 3 6 1 8 5 ,

  T(t)= Aj ,

  Fitness: 36.52531288078

We can conclude that the models found are not yet valuable for our purposes. Some of them can reproduce only a couple of the features that define habituation in a general way. In addition, their shapes on those features are sometimes dissimilar to those in the empiric data.

5.2 Automatic programming of NEPs

This section shows the platform with which we implement a general methodology to automatically design NEPs to solve specific problems. We use CGE/AGE (see section 2.2.3) and jNEP (see section 3.1), two applications we have previously developed. Firstly, we will give a proof of viability where we are interested in linking all the modules and generating the initial population. Building this platform is relevant, because our methodology includes several non trivial steps, such as designing a grammar and implementing and using a simulator. For this first proof we have choosen a well known problem that other authors have solved by means of NEPs.

5.2.1 Motivation

Conventional personal computers are based on the well known von Neumann architecture, that can be considered as an implementation of the Turing machine. A great effort is being devoted to the design of new abstract computing devices, which can be seen as alternative architectures to design new families of computers. Some of them, inspired in the way used by Nature to solve difficult tasks efficiently, are called *natural or unconventional computers*. Some of the natural phenomena

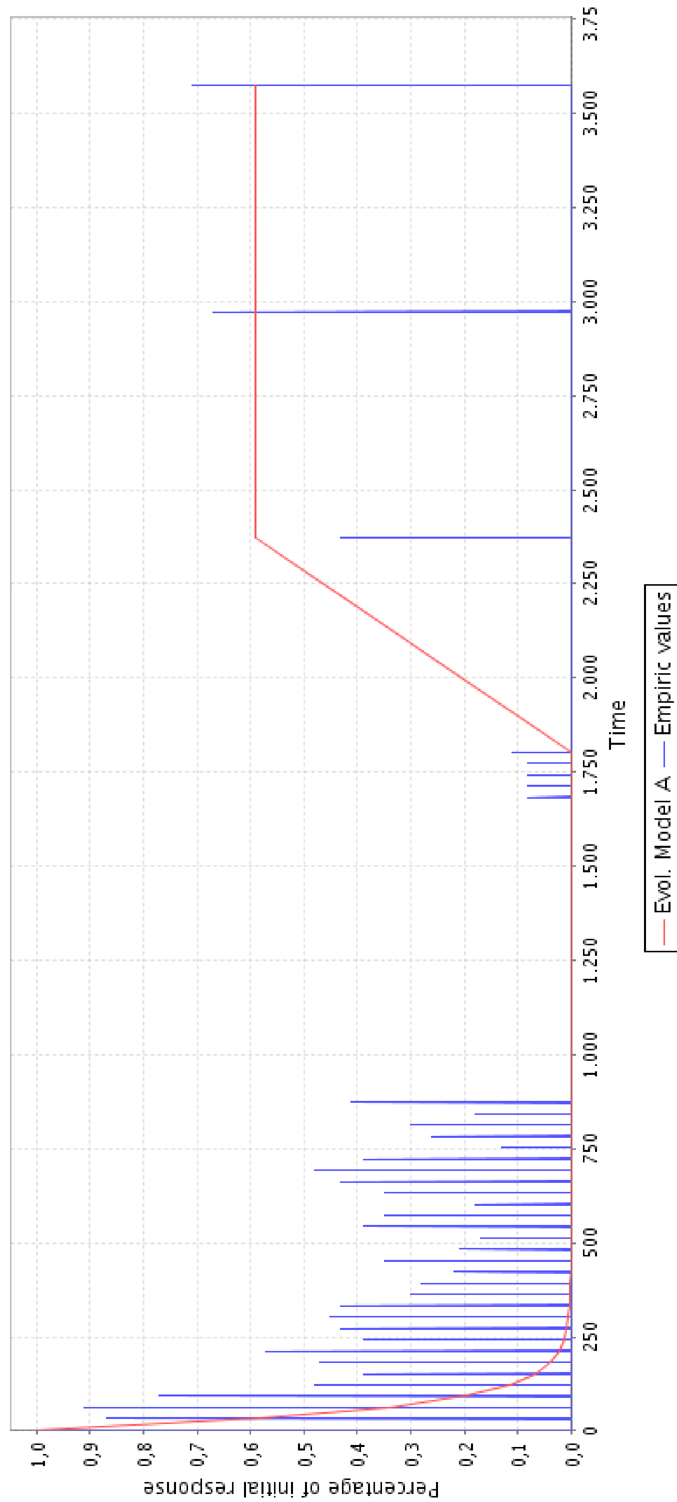


Figure 5.2: *Empiric data against Evol. Model A;* The figure shows the empiric data of Rankin and Broster [1992] (experiment 1, ISI 30) together with the data produced by Model A in the same experimental conditions. We can see a very basic response's strength decrement and spontaneous recovery for Model A.

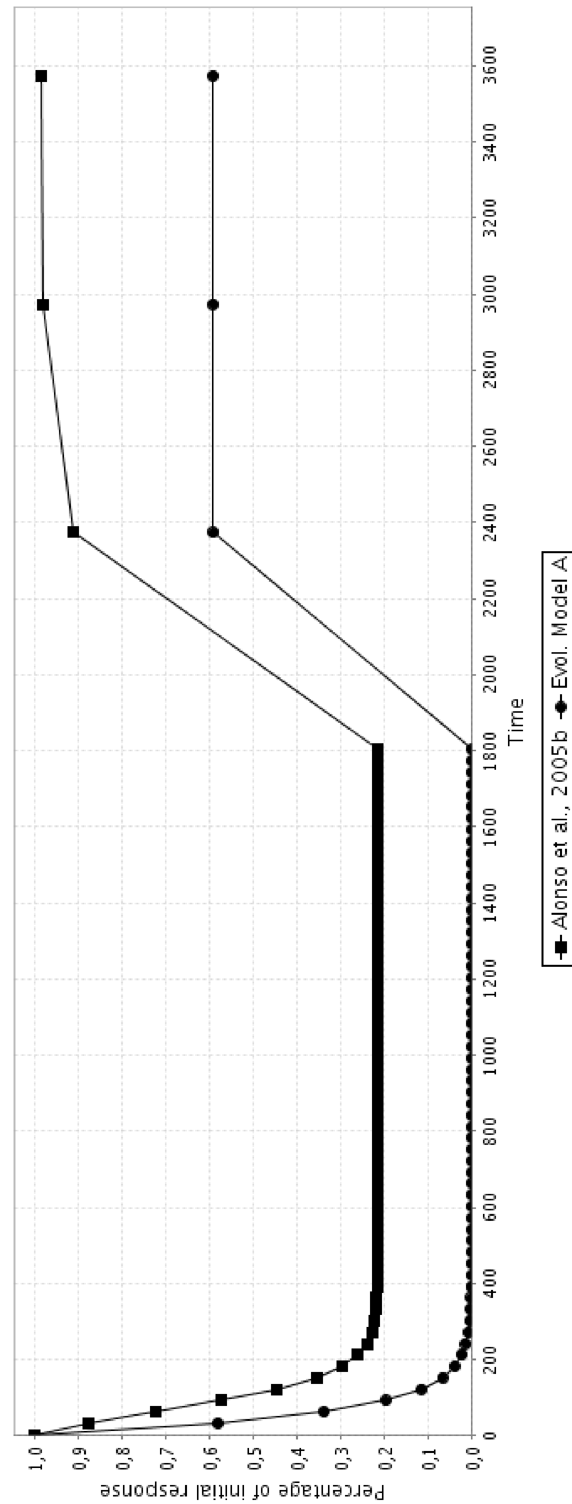


Figure 5.3: *Alonso et al. [2005b] against Evol. Model A;* The figure shows the data produced by Alonso et al. [2005b] (Rankin and Broster [1992], experiment 1, ISI 30) together with the data produced by Model A in the same experimental conditions.

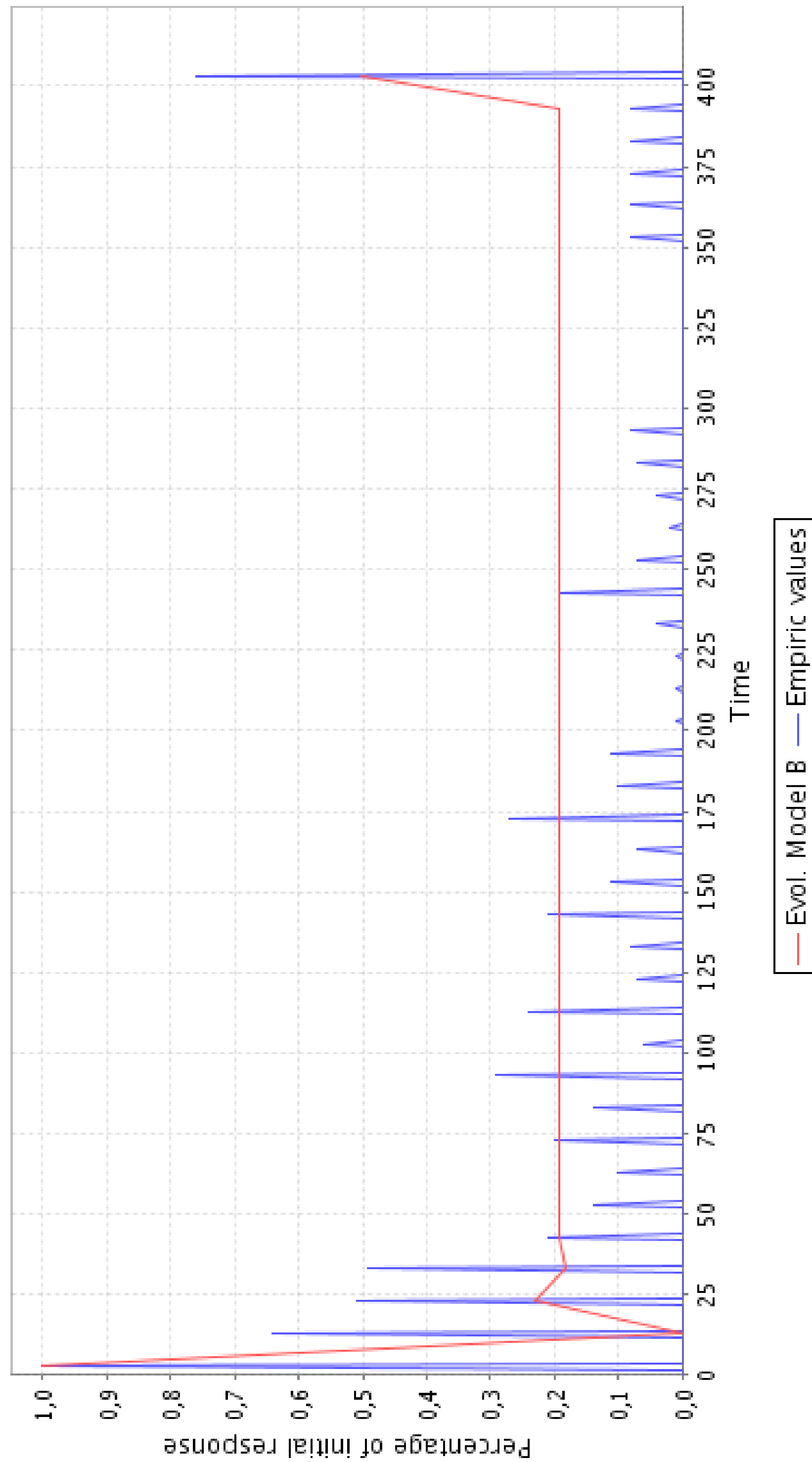


Figure 5.4: *Empiric data against Evol. Model B;* The figure shows the empiric data of Rankin et al. [1990] together with the data produced by Model A in the same experimental conditions. We can see a very basic response's strength decrement and dishabituation for Model A. The second stimulus that produces dishabituation appears only once at second 397, none of its values are shown in the figure for clarity.

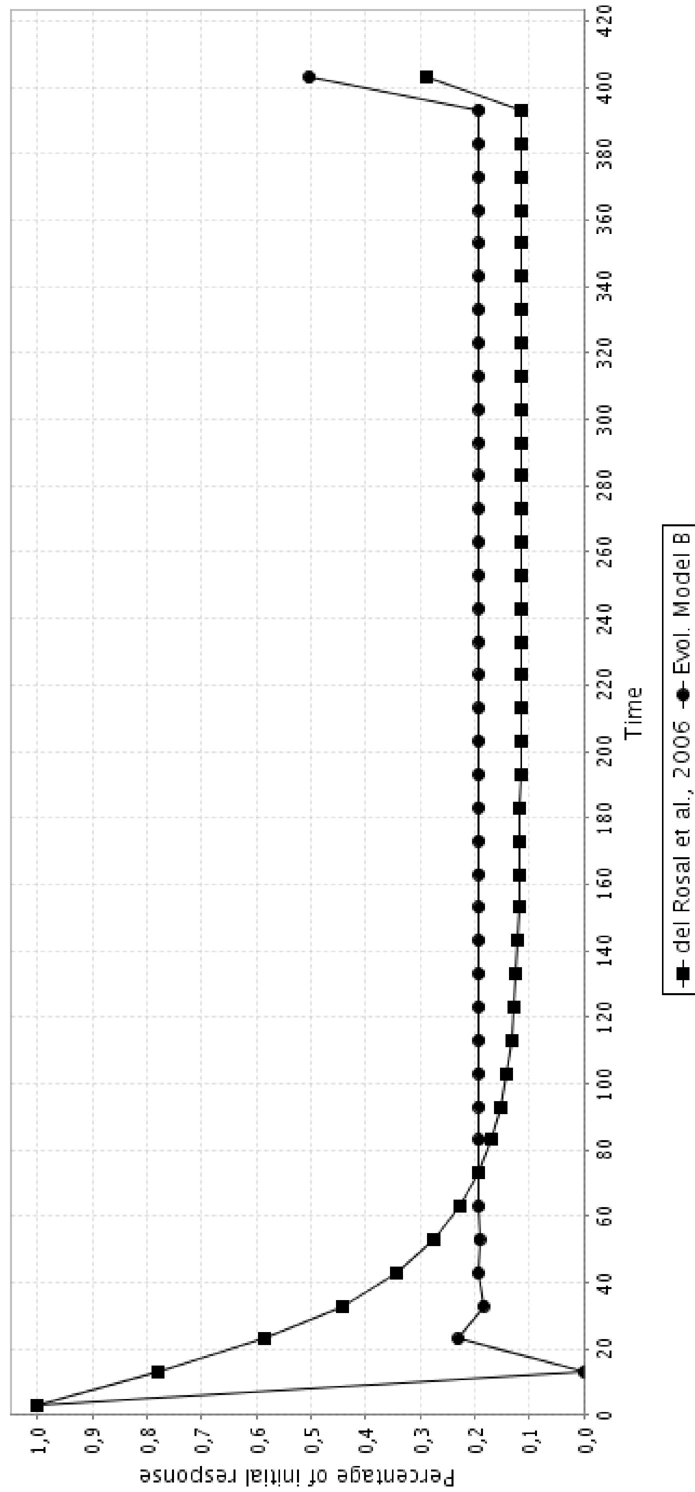


Figure 5.5: *del Rosal et al. [2006] against Evol. Model B;* The figure shows the data produced by del Rosal et al. [2006] (Rankin et al. [1990]) together with the data produced by Model B in the same experimental conditions. The second stimulus that produces dishabituation appears only once at second 397, none of its values are shown in the figure for clarity.

inspiring these devices are: the role of membranes in the behaviour of cells, the structure of genetic information, biological networks that perform natural processes in parallel, and the way in which species evolve.

Any computer scientist has a clear idea about how to program *conventional* (von Neumann) computers by means of different high level programming languages and their corresponding compilers, which translate programs into machine code. On the other hand, imagining how to program unconventional computers is quite difficult. We will show a new platform using evolutionary computation for the automatic programming of a family of natural computers called Networks of Evolutionary Processors or NEPs, which we have already discussed in the previous section 2.3. The task of automatically writing programs can be seen as a search problem: finding the best in a set of candidate programs automatically generated. Any general search technique can be used to solve this problem.

NEPs are abstract devices with a complex structure, because some of their components depend on others. This dependence makes it difficult to use genetic techniques to search NEPs because, in this circumstance, genetic operators usually produce a great number of incorrect individuals (either syntactically or semantically). Fortunately, we have previously explained new evolutionary automatic programming algorithms as powerful tools to design complex systems: Attribute Grammar Evolution, AGE, and Christiansen Grammar Evolution, CGE (see section 2.2.3). Both techniques, AGE and CGE, wholly describe the candidate solutions, both syntactically and semantically, by means, respectively, of attribute and Christiansen grammars; thus improving the performance of other approaches, because they reduce the search space by excluding non-promising individuals with syntactic or semantic errors. For that reason, we have chosen AGE and CGE to study the possibility of automatically program NEPs.

Below, we will show the steps we have taken to implement a real platform to automatically design (or program) NEPs by means of our genetic algorithms (AGE/CGE). We have chosen a well known family of NEPs, able to solve a very simple problem [Csuhaj-Varju et al., 2005]: the application of context free rules by classic NEPs. It is worth noticing that context free rules are not allowed in the classic family of NEPs. In this family, it is just allowed to replace a symbol by a single symbol (rather than a string of symbols, as in context free grammars).

Our final goal is to test our techniques for the automatic design of NEPs to solve given tasks. At the end, our experiments result in the proposal of a methodology to automatically design NEPs.

Figure 5.6 graphically describes the different blocks which can be considered to propose a general way to program natural computers similar to NEPs to solve a given problem.

This method takes as inputs the following elements:

- The *target problem* to be solved.
- The *computing device* that will be used to solve the problem.

And it consists of the following modules:

- An *evolutionary engine*, used as an automatic programming algorithm. This engine has to handle candidate solutions with a complex structure. We propose using AGE or CGE.
- A *formal description of the computing device* being programmed. Mainly, a formal grammar.
- A *simulator for the computing device* that will be used to compute the fitness function.

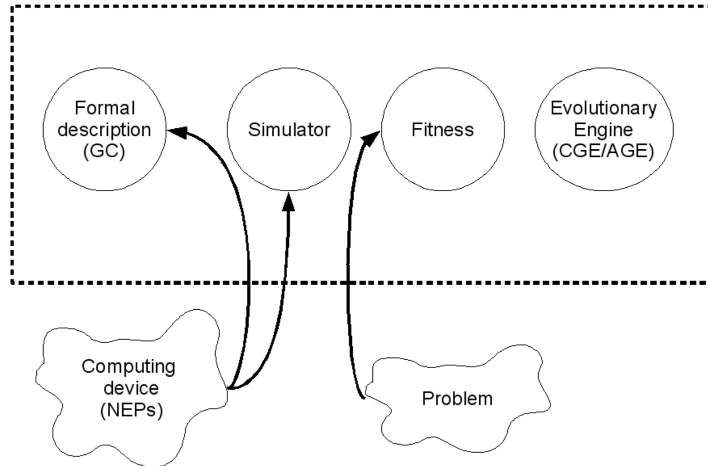


Figure 5.6: Blocks of a general way to program natural computers

- The *fitness function*, which must fulfill two roles:
  1. Simulate the generated solution (in this case, a particular NEP).
  2. Measure how well the solution solves the target problem.

As a first step, we will present the skeleton of our implementation of this methodology for NEPs and explain the generation of a correct initial population, since it is the first mandatory step to test its viability. In this previous work, we designed a static (non adaptable) simple context free grammar (without attributes) that could be used as the kernel for further Christiansen or attribute versions. This grammar generates a family of NEPs that includes that of Csuhaj-Varju et al. [2005]. To reduce the size of the search space, we have fixed all the components of the NEPs except the sets of rules and filters. We also have used an empty fitness function, because our goal is just to generate a valid initial population. Later on, we will include our simulator (jNEP, del Rosal et al. [2008]) and create a real fitness function used to evolve the initial individuals. This way, the platform will be completed and it will be able to find NEPs that solve the aforementioned problem.

5.2.2 Automatic programming of NEPs

According to the general methodology previously outlined, we have decided to use jNEP (see section 3.1) as the simulator for the computing device. The grammar and the fitness function depend on this choice, because, as explained in section 3.1, jNEP uses XML files describing the NEP being simulated as inputs. Therefore, our evolutionary individuals will be valid XML descriptions of the NEPs. Below, we will show the context free grammar we firstly used to generate the initial population. It will be the kernel for the Christiansen grammar we will finally use in further experiments. AGE/CGE are able to include semantic constraints when generating the populations to ensure that only syntactically and semantically valid individuals belong to them. When we design AGE/CGE experiments, we have to tune also the *amount of semantic constraints* we add to the grammar. For the sake of simplicity, we have decided to remove these restrictions and use a context free grammar in these first proofs.

In this work we have not used all the options available for describing NEPs by means of jNEP XML files. jNEP accepts all the variants and constructs found in

the literature. In order to reduce the huge search space defined by the full grammar, we have decided to force some structural and functioning characteristics of NEPs. In the future we will test the system with more general NEPs. The features of the NEPs generated are explained in the following section.

5.2.3 The NEPs to search for

Our long-term goal is to solve a moderately complex problem presented and solved in Csuhaj-Varju et al. [2005]. That paper shows how a NEP simulate the application of context free rules ($A \to \alpha, A \in V, \alpha \in V^*$ for alphabet V) in three steps. The first one (that is our goal) rotates the string where the rule is being applied until placing A in one of the string ends. In Csuhaj-Varju et al. [2005], this task is performed by means of a NEP with 4 nodes sequentially connected.

As we have explained before, we have decided to reduce the characteristics and functionality of general NEPs. In this first proof we have used

- rules of all the kinds described in Martín-Vide and Mitrana [2005], Castellanos et al. [2003], Martin-Vide et al. [2003], Castellanos et al. [2001]
- filters based on *random context conditions*, that is, the four usual types of filters described in Martín-Vide and Mitrana [2005]
- the same graph structure as in Csuhaj-Varju et al. [2005]
- we have bounded the number of symbols of the alphabet (see the grammar below)

This is the context free grammar we have used (notice that symbols “[” and “]” are used to enclose non terminal symbols in the right hand side of the rules while the XML markers for tags “<” and “>” have to appear literally):

NEP ::= <?xml version=”1.0”?> <NEP nodes=“[nodes]”> [alphabetTag] [graphTag] [processorsTag] [stoppingConditionsTag] </NEP>

nodes ::= 5

alphabetTag ::= <ALPHABET symbols=“a_b_c_o_p_q_r_s_t_u_v_w_x_y_z”/>

graphTag ::= <GRAPH> <EDGE vertex1=“0” vertex2=“1”/> <EDGE vertex1=“1” vertex2=“2”/> <EDGE vertex1=“2” vertex2=“3”/> <EDGE vertex1=“3” vertex2=“4”/> </GRAPH>

processorsTag ::= <EVOLUTIONARY_PROCESSORS> [nodeTagInit] [nodeTag] [nodeTag] [nodeTag] [nodeTag] </EVOLUTIONARY_PROCESSORS>

nodeTagInit ::= <NODE initCond=“input”> [evolutionaryRulesTag] [nodeFiltersTag] </NODE>

nodeTag ::= <NODE initCond=“”> [evolutionaryRulesTag] [nodeFiltersTag] </NODE>

evolutionaryRulesTag ::= <EVOLUTIONARY_RULES> [ruleTag] </EVOLUTIONARY_RULES>

ruleTag ::= <RULE ruleType=“[ruleType]” actionType=“[actionType]” symbol=“[symbol]” newSymbol=“[symbol]”/> [ruleTag]

ruleTag ::= λ

ruleType ::= insertion | deletion | substitution

actionType ::= LEFT | RIGHT | ANY

nodeFiltersTag ::= <FILTERS>[inputFilterTag] [outputFilterTag]</FILTERS>

nodeFiltersTag ::= <FILTERS>[inputFilterTag]</FILTERS>

nodeFiltersTag ::= <FILTERS>[outputFilterTag]</FILTERS>

nodeFiltersTag ::= <FILTERS></FILTERS>

inputFilterTag ::= <INPUT [filterSpec]/>

outputFilterTag ::= <OUTPUT [filterSpec]/>

filterSpec ::= type = “[filterType]” permittingContext = “[symbolList]” forbiddingContext = “[symbolList]”

filterType ::= 1 | 2 | 3 | 4
wordList ::= [symbolList] [wordList] | λ
symbolList ::= [auxList] | λ
auxList ::= [symbol] | [symbol]_[auxList]
symbol ::= a|b|c|o|p|q|r|s|t|u|v|w|x|y|z
stoppingConditionsTag ::= <STOPPING_CONDITION> <CONDITION type = "NonEmptyNodeStoppingCondition" nodeID = "[lastNodeID]"/> <CONDITION type = "MaximumStepsStoppingCondition" maximum = "20"/> </STOPPING_CONDITION>

5.2.4 Testing the framework

We have introduced the previous grammar in our CGE engine and run a preliminary evolutionary search. The framework creates a lot of different valid NEPs as expected. One of these NEPs is shown below. It is helpful to remember that this first work does not pretend to find NEPs which solve the problem, instead, we are only interested in creating valid, well-structured individuals within the evolutionary engine.

An example of the generated NEPs This NEP fullfils the functional and structural constraints imposed by the grammar. It has 4 nodes, connected sequentially. The first one contains the input word that should be rotated by the NEP. One of the stopping conditions stops the NEP when a word reaches the last node, finishing the rotation. Others stop it after a given number of steps is executed, or when nothing has changed in two consecutive configurations.

However, rules and filters are almost unrestricted. In this example, the first node adds the symbol "t" at the end of its words and deletes any appearance of the symbol "p". There is no output filter in the first node and no input filter in the second. So, all the words enter the second node. The second node has no rule. Nothing changes from this point and the NEP stops.

The XML file for this NEP is shown below:

```
<?xml version="1.0"?>
  <NEP nodes="5">
  <ALPHABET symbols="a_b_c_o_p_q_r_s_t_u_v_w_x_y_z"/>
  <GRAPH>
    <EDGE vertex1="0" vertex2="1"/>
    <EDGE vertex1="1" vertex2="2"/>
    <EDGE vertex1="2" vertex2="3"/>
    <EDGE vertex1="3" vertex2="4"/>
  </GRAPH>
  <EVOLUTIONARY_PROCESSORS>
    <NODE initCond="input">
      <EVOLUTIONARY_RULES> <RULE ruleType="substitution" actionType="ANY"
                              symbol="z" newSymbol="u"/>
        <RULE ruleType="deletion" actionType="LEFT" symbol="p" newSymbol="u"/>
        <RULE ruleType="insertion" actionType="RIGHT" symbol="t" newSymbol="v"/>
      </EVOLUTIONARY_RULES> <FILTERS> </FILTERS>
    </NODE>
    <NODE initCond="">
      <EVOLUTIONARY_RULES></EVOLUTIONARY_RULES>
      <FILTERS></FILTERS>
    </NODE>
    <NODE initCond="">
      <EVOLUTIONARY_RULES> </EVOLUTIONARY_RULES>
      <FILTERS> <OUTPUT type="1" permittingContext="c" forbiddingContext="y"/>
```

```
      </FILTERS>
    </NODE>
    <NODE initCond="">
      <EVOLUTIONARY_RULES> </EVOLUTIONARY_RULES>
      <FILTERS>
        <INPUT type="2" permittingContext="" forbiddingContext="" />
        <OUTPUT type="1" permittingContext="" forbiddingContext="a_w_t" />
      </FILTERS>
    </NODE>
    <NODE initCond="">
      <EVOLUTIONARY_RULES> </EVOLUTIONARY_RULES>
      <FILTERS>
        <INPUT type="4" permittingContext="" forbiddingContext="" />
        <OUTPUT type="1" permittingContext="" forbiddingContext="" />
      </FILTERS>
    </NODE>
  </EVOLUTIONARY_PROCESSORS>
  <STOPPING_CONDITION>
    <CONDITION type="NonEmptyNodeStoppingCondition" nodeID="4"/>
    <CONDITION type="MaximumStepsStoppingCondition" maximum="15"/>
  </STOPPING_CONDITION>
</NEP>
```

We also show the output that jNEP generates while simulating this NEP.

```
XML CONFIGURATION FILE LOADED AND PARSED SUCCESSFULLY...
GRAPH INFO PARSED SUCCESSFULLY...
STOPPING CONDITIONS INFO PARSED SUCCESSFULLY...
EVOLUTIONARY PROCESSORS INFO PARSED SUCCESSFULLY...
****  NEP INITIAL CONFIGURATION  ****
        --- Evolutionary Processor 0 ---
input
        --- Evolutionary Processor 1 ---
...
        --- Evolutionary Processor 4 ---

****  NEP CONFIGURATION - EVOLUTIONARY STEP - TOTAL STEPS: 1  ****
        --- Evolutionary Processor 0 ---
input_t input
        --- Evolutionary Processor 1 ---
...
        --- Evolutionary Processor 4 ---

****  NEP CONFIGURATION - COMMUNICATION STEP - TOTAL STEPS:  2  ****
        --- Evolutionary Processor 0 ---

        --- Evolutionary Processor 1 ---
input_t input
        --- Evolutionary Processor 2 ---
...

****  NEP CONFIGURATION - EVOLUTIONARY STEP - TOTAL STEPS: 3  ****
        --- Evolutionary Processor 0 ---

        --- Evolutionary Processor 1 ---
```

```
        --- Evolutionary Processor 2 ---

        --- Evolutionary Processor 3 ---

        --- Evolutionary Processor 4 ---

...

---------------------- NEP has stopped!!! ----------------------

Stopping condition found:
net.e_delrosal.jnep.stopping.NoChangesStoppingCondition

------------------------------------------------------------------

We are glad you used jNEP
```

In short, we have implemented an initial context free grammar for a family of NEPs able to solve a given problem: the application of context free rules. This family of NEPs has been taken from the literature [Csuhaj-Varju et al., 2005]. The same graph structure has been used, while the number of symbols of the alphabet has been limited. To find the proper rules and filters, our grammar generates valid XML files to be input into jNEP. Different initial populations have been successfully generated.

We still have to work on the following points in order to check if our general methodology is applicable to this problem. In the future, we plan to:

- Design and implement a proper fitness function.
- Add additional semantic constraints to our grammar, to drive the search.
- Design experiments to find, by means of CGE/AGE, different solutions to this problem.
- Compare the solucions automatically designed to the one presented in (Csuhaj-Varju et al. [2005])

5.2.5 The complete solution.

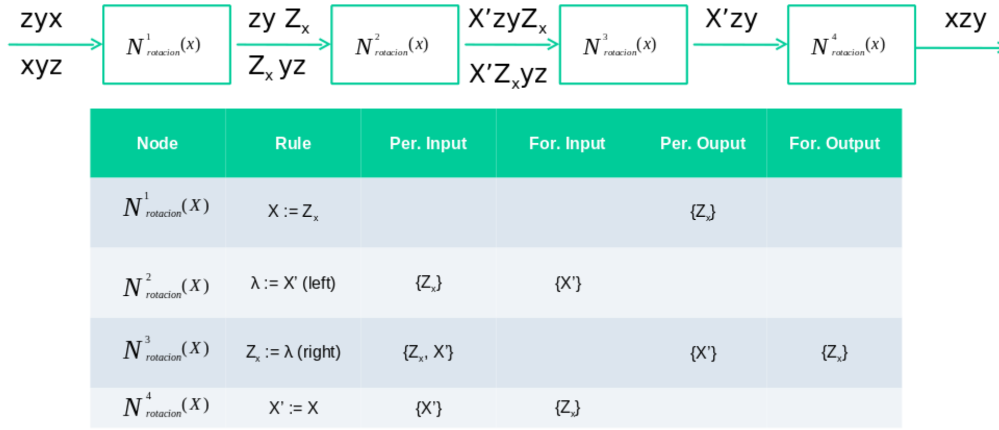
Below, we will present our results in solving a real problem with this platform and improves our previous work [del Rosal et al., 2011] (in which we have tested the feasibility of generating syntactically correct NEPs by means of a simple context free grammar) by adding semantics and a complete fitness function.

5.2.5.1 Introduction to NEPs to rotate strings

As mentioned before, we wanted to solve a problem presented in Csuhaj-Varju et al. [2005]. In that paper, a NEP simulates the application of context free rules ($A \rightarrow \alpha, A \in V, \alpha \in V^*$ for alphabet V) in three steps. The first one rotates the string where the rule is being applied until placing A in one of the string ends. This task is performed by means of a sub-NEP with 4 nodes connected as a linear chain. We will focus on this first step.

The computation of the rotating sub-NEP can be summarized as follows: let us call "s" the symbol being rotated. The first node of the NEP receives a word where the symbol "s" is at the end. This node substitutes "s" by an auxiliary symbol ("s_{a1}"). Then, the new word is sent to the second node, where a new auxiliary

Figure 5.7: Simplified scheme of the rotation NEP presented in Csuhaj-Varju et al. [2005]



Node	Rule	Per. Input	For. Input	Per. Ouput	For. Output
$N^1_{rotacion}(X)$	X := Z_x			{Z_x}	
$N^2_{rotacion}(X)$	λ := X' (left)	{Z_x}	{X'}		
$N^3_{rotacion}(X)$	Z_x := λ (right)	{Z_x, X'}		{X'}	{Z_x}
$N^4_{rotacion}(X)$	X' := X	{X'}	{Z_x}		

symbol ("s_{a2}") is added to the beginning of the word. The last two nodes remove "s_{a1}" and substitute "s_{a2}" by the original "s". These nodes use filters that reject those words without the auxiliary symbols. At this point the rotation of "s" has finished. This cycle could be repeated as many times as needed until finding the symbol to which we want to apply the rule. Thus, this sub-NEP works as a chain of nodes working sequentially. It should be noted that the complete NEP has a different sub-NEP to rotate each non terminal symbol in the original grammar. Figure 5.7 shows a scheme of one of these sub-NEPs dedicated to rotate the symbol "x". Furthermore, the nodes' components are detailed. The upper trace corresponds to the string "zyx", which is rotated since it contains "x" at the end. However, the lower trace shows the computation for the string "xyz" which has no "x" symbol at the end and, therefore, the filters do not allow it to pass to the last nodes.

5.2.5.2 Our solution

In the following sections we will describe each component of the methodology introduced in 5.2.1 to solve the problem under consideration.

In these first tests we have reduced the search space corresponding to any kind of NEP. We have only considered NEPs with the following characteristics:

- They have the same graph structure as in Csuhaj-Varju et al. [2005]: a linear chain.
- In addition, we have reduced the size of the alphabet

Nevertheless we are not considering any additional constraint to

- the rules described in Martín-Vide and Mitrana [2005], Castellanos et al. [2003], Martin-Vide et al. [2003], Castellanos et al. [2001]. However, we have limited the total number of rules in the NEP, so as to avoid too complex, inefficient NEPs.
- the filters based on *random context conditions* described in Martín-Vide and Mitrana [2005]

The grammar and the fitness function. The complex structure of the individuals that we have to generate in our experiments (NEPs that belong to a particular family) made us design a Christiansen Grammar to describe them. As we will explain below, our fitness function invokes jNEP to check if the generated NEP properly processes some input strings. jNEP takes as input a XML configuration file that describes the NEP that is being simulated. This is why our Christiansen Grammar actually generates the XML files that represent the NEPs and that can be read by jNEP as inputs.

As mentioned in section 2.2.3, Cristiansen grammars are Attribute Grammars in which the first attribute of each non-terminal symbol is the actual Grammar applicable to this symbol. We follow the notation introduced in Watt and Madsen [1977]: inherited and synthesized attributes are preceeded by down and up arrows ($\downarrow$ and $\uparrow$), respectively. Attributes are enclosed in round brackets next to their non terminal symbols. Each rule has its corresponding semantic actions (to compute the values of its attributes) enclosed in brackets after its right hand side. Figure 5.8 describes the grammar. Some actions are not shown for simplicity.

On the other hand, we evaluate the fitness of each phenotype by means of the function that we describe below. First, jNEP simulates the NEP corresponding to the genotype to check if it properly processes some input strings. Then, the function checks if a specific symbol can be rotated in different strings: it checks if the symbol “c” can be moved from the end of a string to the beginning. It also checks if the solution works for a set of different strings. The fitness function returns a value between zero and one. If a NEP can rotate a large amount of strings, the fitness function returns a value close to one and vice versa. In order to obtain a smoother and more progressive fitness function (which is always desirable in an evolutionary search), we increased the value of the individual if it can perform sequences of sub-tasks. We have implemented this criterion assigning higher fitness values to those NEPs that can communicate strings across their chain of nodes.

The detailed computation is described below (returned values between 0 and 1):

1. The NEP is run once for each string of the set {“abc”, “aabbcc”, “aac”, “bbc”, “cca”, “bcb”}.

2. The right set of outputs is {“cab”, “caabbc”, “caa”, “cbb”, ””, ””}. A value proportional to the number of matches is returned (see below).

3. The ability to perform sequences of sub-tasks is evaluated for every string as follows:

    - If the penultimate node of the chain contains one or more strings, 0.0416 is added.
    - When the last node contains one or more strings, 0.0416 is added.
    - If the last node contains the desired output, 0.083 is added.

5.2.5.3 Experiments and results

After eight runs of two thousand generations with populations of one thousand individuals, most of the experiments found a perfect (maximum fitness value) or almost perfect solution.

The detailed parameters are the following:

- Population: 1000.
- Codons: 0-256.
- Maximum wrappings operations: 2.

Figure 5.8: The Christiansen Grammar

In the first rule the father inherits the original Christiansen Grammar with their children.

[NEP](g) ::= <?xml version="1.0"?><NEP nodes="[nodes](↓g)">[alphabetTag](↓g) [graphTag](↓g) [processorsTag](↓g)[stoppingConditionsTag](↓g)</NEP>
{

[nodes].↓g = [NEP].g

[alphabetTag].↓g = [nodes].↑g_new

[graphTag].↓g = [nodes].↑g_new

[processorsTag].↓g = [nodes].↑g_new

[stoppingConditionsTag].↓g = [nodes].↑g_new

}

[nodes](↓g) ::= 5

[alphabetTag] ::=<ALPHABET symbols="a_b_c_u_v_w_x_y_z"/>

[graphTag] ::= <GRAPH><EDGE vertex1="0" vertex2="1"/><EDGE vertex1="1" vertex2="2"/><EDGE vertex1="2" vertex2="3"/><EDGE vertex1="3" vertex2="4"/></GRAPH>

The following rule derives the processors. It computes the total number of rules (by means of the expansion of the non terminal [inputNodeTag]) and limits it to 20: thus it is not possible to generate phenotypes with more than 20 rules.

[processorsTag](↓g) ::= <EVOLUTIONARY_PROCESSORS> [inputNodeTag](↓g,↓counterInit,↑counterFinal) [nodeTag]$_1$(↓g,↓counterInit,↑counterFinal) [nodeTag]$_2$(↓g,↓counterInit,↑counterFinal) [nodeTag]$_3$(↓g,↓counterInit,↑counterFinal) [nodeTag]$_4$(↓g,↓counterInit,↑counterFinal) </EVOLUTIONARY_PROCESSORS>
{

EVERY CHILD INHERITS THE CHRISTIANSEN GRAMMAR AS IN PREVIOUS RULES

[inputNodeTag].↓counterInit = 0

[nodeTag]$_i$.↑counterFinal = [nodeTag]$_{i+1}$.↓counterFinal

}

[inputNodeTag](↓g,↓counterInit,↑counterFinal) ::= <NODE initCond="`input word to rotate`"> [evolutionaryRulesTag](↓g,↓counterInit,↑counterFinal) [nodeFiltersTag](↓g,) </NODE>
{

EVERY CHILD INHERITS THE CHRISTIANSEN GRAMMAR AS IN PREVIOUS RULES

[evolutionaryRulesTag].↓counterInit = [inputNodeTag].↓counterInit

[inputNodeTag].↑counterFinal = [evolutionaryRulesTag].↑counterFinal

}

[nodeTag](↓g,↓counterInit,↑counterFinal) ::= <NODE initCond=""> [evolutionaryRulesTag](↓g,↓counterInit,↑counterFinal) [nodeFiltersTag](↓g) </NODE>
{

Semantic actions equivalent to the previous one.

}

[evolutionaryRulesTag](↓g,↓counterInit,↑counterFinal) ::= <EVOLUTIONARY_RULES> [ruleTag](↓g,↓counterInit,↑counterFinal) </EVOLUTIONARY_RULES>
{

EVERY CHILD INHERITS THE CHRISTIANSEN GRAMMAR AS IN PREVIOUS RULES

[ruleTag].↓counterInit = [evolutionaryRulesTag].↓counterInit

[evolutionaryRulesTag].↑counterFinal = [ruleTag].↑counterFinal

}

[ruleTag]$_a$(↓g,↓counterInit,↑counterFinal) ::= <RULE ruleType="[ruleType](↓g)" actionType="[actionType](↓g)" symbol="[symbol](↓g)" newSymbol="[symbol](↓g)"/> [ruleTag]$_b$(↓g,.↓counterInit,↑counterFinal)
{

EVERY CHILD INHERITS THE CHRISTIANSEN GRAMMAR AS IN PREVIOUS RULES

[ruleTag]$_b$.↓counterInit = [ruleTag]$_a$.↓counterInit +1

[ruleTag]$_a$.↑counterFinal = [ruleTag]$_b$.↑counterFinal

if ([ruleTag]$_b$.↑counterFinal > 20) dismissPhenotype();

}

[ruleTag](↓g,↓counterInit,↑counterFinal) ::= λ
{

[ruleTag].↑counterFinal = [ruleTag].↓counterInit

}

[ruleType] ::= insertion | deletion | substitution

[actionType] ::= LEFT | RIGHT | ANY

[nodeFiltersTag] ::= [inputFilterTag] [outputFilterTag]

[nodeFiltersTag] ::= [inputFilterTag]

[nodeFiltersTag] ::= [outputFilterTag]

[nodeFiltersTag] ::= λ

[inputFilterTag] ::= <INPUT [filterSpec]/>

[outputFilterTag] ::= <OUTPUT [filterSpec]/>

[filterSpec] ::= type="[filterType]" permittingContext="[symbolList]" forbiddingContext="[symbolList]"

[filterType] ::= 1 | 2 | 3 | 4

[wordList] ::= [symbolList] [wordList] |λ

[symbolList] ::=
`a string of the alphabet's symbols separated by the character '_'`

The following rule derives the stopping condition. We consider three conditions:

- The NEP stops when some string enters the output node
- The computation finishes after a maximum number of steps has been taken
- or when a maximum number of strings has been generated

[stoppingConditionsTag] ::= <STOPPING_CONDITION><CONDITION type="NonEmptyNodeStoppingCondition" nodeID="4"/> <CONDITION type="MaximumStepsStoppingCondition" maximum="8"/> <CONDITION type="MaximumSizeStoppingCondition" maximum="100"/></STOPPING_CONDITION>

- Mutation probability: 100% (each genotype mutates one of its codons in every generation)
- Crossover probability: 95%.
- Generational replacement: elitist.
- Initial genotype length: 200.

Below, we show the jNEP input file for one of the solutions found. We have omitted some elements of the configuration that have no effect on the computation.

```
<?xml version="1.0"?>
  <NEP nodes="5">
```

There are three symbols in the strings that can be rotated a,b,c, the rest can be used as auxiliary symbols by the NEP.

```
  <ALPHABET symbols="a_b_c_o_p_q_r_s_t_u_v_w_x_y_z"/>
  <GRAPH>
    <EDGE vertex1="0" vertex2="1"/>
    <EDGE vertex1="1" vertex2="2"/>
    <EDGE vertex1="2" vertex2="3"/>
    <EDGE vertex1="3" vertex2="4"/>
  </GRAPH>

  <EVOLUTIONARY_PROCESSORS>
```

Remember that our fitness function firstly checks if the symbol c is properly rotated in a set of strings. The input string is placed at this first node. Those strings that finish with an a will change it by b. Therefore, this node can only transfer strings that end with the symbols b and c (the symbol being rotated) at the end.

```
    <NODE initCond="input">
      <EVOLUTIONARY_RULES>
       <RULE ruleType="substitution" actionType="RIGHT" symbol="a" newSymbol="b"/>
      </EVOLUTIONARY_RULES>
      <FILTERS> </FILTERS>
    </NODE>
```

The second node substitutes every b at the end by the auxiliary symbol y. Therefore, after this node, all the strings will finish with c (the symbol being rotated) or y. We can consider, thus, that this node has marked the non-rotating strings .

```
    <NODE initCond="">
      <EVOLUTIONARY_RULES>
        <RULE ruleType="substitution" actionType="RIGHT" symbol="b" newSymbol="y"/>
      </EVOLUTIONARY_RULES>
      <FILTERS> </FILTERS>
    </NODE>
```

The third node adds the symbol that is being rotated (c) in the left side of the string.

```
    <NODE initCond="">
      <EVOLUTIONARY_RULES>
        <RULE ruleType="insertion" actionType="LEFT" symbol="c"/>
      </EVOLUTIONARY_RULES>
      <FILTERS> </FILTERS>
    </NODE>
```

This node, finally, deletes the rotating symbol from its original position. The non-rotating strings can not pass this point since the output filter forbids the symbol y.

```
    <NODE initCond="">
      <EVOLUTIONARY_RULES>
        <RULE ruleType="deletion" actionType="RIGHT" symbol="c"/>
      </EVOLUTIONARY_RULES>
      <FILTERS>
        <OUTPUT type="3" permittingContext="" forbiddingContext="y" />
      </FILTERS>
    </NODE>
    <NODE initCond="">
      <EVOLUTIONARY_RULES>
      </EVOLUTIONARY_RULES>
      <FILTERS> </FILTERS>
    </NODE>
  </EVOLUTIONARY_PROCESSORS>

  <STOPPING_CONDITION>
    <CONDITION type="NonEmptyNodeStoppingCondition" nodeID="4"/>
    <CONDITION type="MaximumStepsStoppingCondition" maximum="8"/>
    <CONDITION type="MaximumSizeStoppingCondition" maximum="100"/>
  </STOPPING_CONDITION>

</NEP>
```

Figure 5.7 shows the description of the solution proposed in Csuhaj-Varju et al. [2005]. It is worth mentioning that the solution proposed in Csuhaj-Varju et al. [2005] follows a different approach: at the first step, the rotating symbol is replaced by a *label* (an auxiliary symbol). This symbol causes the string to pass the following filters. The next node discards any string without this label. In the last stage, the auxiliary symbol is deleted and the rotating symbol is inserted at the beginning.

It is amazing that our solution makes the opposite, it marks the non-rotating symbols with the label y to discard them later. Furthermore, the NEP described in Csuhaj-Varju et al. [2005] needs to perform more tasks and its rotating sub-NEP needs one more auxiliary symbol for good coordination with the rest of the NEP.

The next paragraphs show the outputs corresponding to the input string *abc* and to a non-rotating string, respectively. Firstly, the succesfull rotation of *abc*.

```
***************                NEP INITIAL CONFIGURATION                ***************
--- Evolutionary Processor 0 ---
a_b_c
--- Evolutionary Processor 1 ---

--- Evolutionary Processor 2 ---

--- Evolutionary Processor 3 ---

--- Evolutionary Processor 4 ---

*************** NEP CONFIGURATION - EVOLUTIONARY STEP - TOTAL STEPS: 1 ***************
--- Evolutionary Processor 0 ---
a_b_c
--- Evolutionary Processor 1 ---

--- Evolutionary Processor 2 ---

--- Evolutionary Processor 3 ---

--- Evolutionary Processor 4 ---

*************** NEP CONFIGURATION - COMMUNICATION STEP - TOTAL STEPS:  2 ***************
```

```
--- Evolutionary Processor 0 ---

--- Evolutionary Processor 1 ---
a_b_c
--- Evolutionary Processor 2 ---

--- Evolutionary Processor 3 ---

--- Evolutionary Processor 4 ---

*************** NEP CONFIGURATION - EVOLUTIONARY STEP - TOTAL STEPS: 3 ***************
--- Evolutionary Processor 0 ---

--- Evolutionary Processor 1 ---
a_b_c
--- Evolutionary Processor 2 ---

--- Evolutionary Processor 3 ---

--- Evolutionary Processor 4 ---

*************** NEP CONFIGURATION - COMMUNICATION STEP - TOTAL STEPS:  4 ***************
--- Evolutionary Processor 0 ---
a_b_c
--- Evolutionary Processor 1 ---

--- Evolutionary Processor 2 ---
a_b_c
--- Evolutionary Processor 3 ---

--- Evolutionary Processor 4 ---

*************** NEP CONFIGURATION - EVOLUTIONARY STEP - TOTAL STEPS: 5 ***************
--- Evolutionary Processor 0 ---
a_b_c
--- Evolutionary Processor 1 ---

--- Evolutionary Processor 2 ---
c_a_b_c
--- Evolutionary Processor 3 ---

--- Evolutionary Processor 4 ---

*************** NEP CONFIGURATION - COMMUNICATION STEP - TOTAL STEPS:  6 ***************
--- Evolutionary Processor 0 ---

--- Evolutionary Processor 1 ---
c_a_b_c a_b_c
--- Evolutionary Processor 2 ---

--- Evolutionary Processor 3 ---
c_a_b_c
--- Evolutionary Processor 4 ---

*************** NEP CONFIGURATION - EVOLUTIONARY STEP - TOTAL STEPS: 7 ***************
--- Evolutionary Processor 0 ---

--- Evolutionary Processor 1 ---
c_a_b_c a_b_c
--- Evolutionary Processor 2 ---

--- Evolutionary Processor 3 ---
c_a_b
--- Evolutionary Processor 4 ---

*************** NEP CONFIGURATION - COMMUNICATION STEP - TOTAL STEPS:  8 ***************
--- Evolutionary Processor 0 ---
```

```
c_a_b_c a_b_c
--- Evolutionary Processor 1 ---

--- Evolutionary Processor 2 ---
c_a_b c_a_b_c a_b_c
--- Evolutionary Processor 3 ---

--- Evolutionary Processor 4 ---
c_a_b
---------------------- NEP has stopped!!! ----------------------
```

Secondly, the string *abc* can not be rotated because it does not contain the symbol "c" at the end.

```
***************                NEP INITIAL CONFIGURATION                ***************
--- Evolutionary Processor 0 ---
b_c_b
--- Evolutionary Processor 1 ---

--- Evolutionary Processor 2 ---

--- Evolutionary Processor 3 ---

--- Evolutionary Processor 4 ---

*************** NEP CONFIGURATION - EVOLUTIONARY STEP - TOTAL STEPS: 1 ***************
--- Evolutionary Processor 0 ---
b_c_b
--- Evolutionary Processor 1 ---

--- Evolutionary Processor 2 ---

--- Evolutionary Processor 3 ---

--- Evolutionary Processor 4 ---

*************** NEP CONFIGURATION - COMMUNICATION STEP - TOTAL STEPS:  2 ***************
--- Evolutionary Processor 0 ---

--- Evolutionary Processor 1 ---
b_c_b
--- Evolutionary Processor 2 ---

--- Evolutionary Processor 3 ---

--- Evolutionary Processor 4 ---

*************** NEP CONFIGURATION - EVOLUTIONARY STEP - TOTAL STEPS: 3 ***************
--- Evolutionary Processor 0 ---

--- Evolutionary Processor 1 ---
b_c_y
--- Evolutionary Processor 2 ---

--- Evolutionary Processor 3 ---

--- Evolutionary Processor 4 ---

*************** NEP CONFIGURATION - COMMUNICATION STEP - TOTAL STEPS:  4 ***************
--- Evolutionary Processor 0 ---
b_c_y
--- Evolutionary Processor 1 ---

--- Evolutionary Processor 2 ---
b_c_y
--- Evolutionary Processor 3 ---
```

```
--- Evolutionary Processor 4 ---

*************** NEP CONFIGURATION - EVOLUTIONARY STEP - TOTAL STEPS: 5 ***************
--- Evolutionary Processor 0 ---
b_c_y
--- Evolutionary Processor 1 ---

--- Evolutionary Processor 2 ---
c_b_c_y
--- Evolutionary Processor 3 ---

--- Evolutionary Processor 4 ---

*************** NEP CONFIGURATION - COMMUNICATION STEP - TOTAL STEPS:  6 ***************
--- Evolutionary Processor 0 ---

--- Evolutionary Processor 1 ---
b_c_y c_b_c_y
--- Evolutionary Processor 2 ---

--- Evolutionary Processor 3 ---
c_b_c_y
--- Evolutionary Processor 4 ---

*************** NEP CONFIGURATION - EVOLUTIONARY STEP - TOTAL STEPS: 7 ***************
--- Evolutionary Processor 0 ---

--- Evolutionary Processor 1 ---
b_c_y c_b_c_y
--- Evolutionary Processor 2 ---

--- Evolutionary Processor 3 ---
c_b_c_y
--- Evolutionary Processor 4 ---

*************** NEP CONFIGURATION - COMMUNICATION STEP - TOTAL STEPS:  8 ***************
--- Evolutionary Processor 0 ---
b_c_y c_b_c_y
--- Evolutionary Processor 1 ---

--- Evolutionary Processor 2 ---
b_c_y c_b_c_y
--- Evolutionary Processor 3 ---
c_b_c_y
--- Evolutionary Processor 4 ---

----------------------- NEP has stopped!!! -----------------------
```

5.2.5.4 Conclusions and further research lines

During this work, we have, for the first time, tackled a non trivial problem by means of the platform we are proposing to automatically design NEPs. Although we have simplified in some way the problem under consideration, and constrained different elements of the NEPs being evolved in order to reduce the search space, we have found interesting solutions similar to those described in literature. These solutions could be considered as valid alternatives. We are, then, optimistic with respect to the feasibility of using our platform to solve more general problems in more general domains. In the future we have to generalize different aspects of the work described above:

- We have to find NEPs to rotate any symbol of the alphabet.
- We have to search with a more general family of NEPs by removing some of the constraints used in this work.

- We have to find NEPs to solve the complete problem of applying context-free rules.
- We have to tackle different non trivial problems.
- We are interested in adding to our platform a general way for describing the problem under consideration and for including it in the fitness function in a more standard way.

5.3 A methodology for automatic modelling

As we have worked in modelling and designing complex systems in sections 5.1 and 5.2, we have implicitly outlined a general methodology for automatic modelling in science and engineering. We think it is important to detail such methodology because it is an interesting way to exploit the features of Grammatical Evolution to manage complex formalisms. In addition, it could be a good alternative for automatically modelling in many research contexts, where no clear knowledge or insight of the complex system to model is available to the researcher.

Modelling method Before running the evolutionary algorithm, we need to prepair its principal components. Each component will reflect the modelling aims, assumptions and practical matters of the researcher. Below we enumerate the list of components and explain how they have to be set.

1 **Grammar**: The grammar introduced in the GE system determines the expression space within which the evolutionary algorithm will search for our model. Therefore, it is the mechanism by which the researcher can introduce all kinds of conditions. The GE's extensions presented in section 2.2.3 that increase the power of context-free grammar are especially useful for this purpose. These conditions can be divided in 3 categories.

   (a) **Model's general architecture**: Basically, it comprehends the formalism used to express the model. For instance, differential equations or formal automata.

   (b) **Researcher's assumptions and bias**: Due to theoretical reasons or model assumptions we may want to introduce some restrictions to the grammar. For instance, we may limit the number of variables and their possible relations or force some neural clusters if we are using neural networks.

   (c) **Other constraints**: For practical or technical reasons we may need to introduce some constraints. For example; avoiding overly complex expressions that are not easy to interpret by imposing a maximun of terminals in the expression.

2 **Fitness function**: To asses the quality of the expressions that the algorithm generates, it is mandatory to define a fitness function. This function has to be able to sort any model by quality in order to make comparisons possible. For example, in the case of our Habituation model, the fitness function consists of calculating the difference between empirical data on habituation and the simulated data produced by the model.

These two components must be set, later another two actions have to be developed so as to complete our task.

3 **Parameter tuning and search**: Once the previous components are set, it is possible to run the evolutionary search. This phase is not different from any other use of evolutionary algorithms. Most times we will need some parameter tuning before the algorithm can find valuable solutions.

4 **Interpretation and study of the model**: In some contexts, the fitness function does not provide enough information to conclude that the model found is worthy and appropriate. In other cases, the model found may fit the conditions imposed but also bring some other knowledge in terms of the way it performs the task. Therefore, it may be important to make some efforts in order to deeply understand the model.

For example, if we are in a scientific speculative or theorizing context, some study and interpretation of the model is needed to draw conclusions, otherwise the model significance is reduced to its suitability to a set of computational constraints and conditions (the fitness function). It is important to note that a model provides an explanation for a real system and, therefore, may have theoretical implications.

These 4 steps were described in a deeper way for our specific case of habituation.

Chapter 6

Conclussions and final comments

> "What matter if it be great or small? If it be called swamp or sky? A handbreadth of basis is enough for me, if it be actually basis and ground!
>
> A handbreadth of basis: thereon can one stand."
>
> *Thus spake Zarathustra*, by Friedrich Nietzsche.

Nature has given to computer scientists a huge field for inspiration and exploration. Novel algorithmic approaches can be derived from the way Nature works. In addition, researches have developed different ways to simulate natural phenomena in computers, leading to a better understading of the simulated system but also to new computing paradigms. Moreover, natural materials could be, in the near future, the basis for new computers. Promisingly, these new hardware paradigms will make it possible to solve problems that are intractable for classical computers. Indeed, the productive interplay between Computing and Nature is producing a great amount of innovation and new research lines, mainly due to its multidisciplinary root.

We have tried to contribute to this fresh and exciting new field throughout our investigation. Particularly, our contribution is focused on: a) developing and studying a complete framework for simulating and researching on Network of Evolutionary Processors, b) testing the suitability of NEPs to solve specific problems and c) exploring the potential of Grammatical Evolution algorithms to automatically model, design or program complex systems like those found in nature.

jNEP and its improvements permit to work on the programming and simulation of NEPs and, therefore, achieve a practical understanding of the NEP model. We have tried to provide a useful platform to the researchers in this field. For that purpose, we have not just created a complete simulator of NEPs but a graphical viewer which facilitates the study of them. To improve the productivity of researchers, a visual language based on AToM^3 and under-development high level language (NEPs-Lingua) has been also presented. Furthermore, jNEP is prepared to take advantage of the parallel nature of NEPs because it is able to run on parallel Java platforms. It is worth noticing that, to deepen in this key point of NEPs, author's colleagues has created and tested an adapted version of jNEP written in *ANSI C++* and capable of running on clusters. This new version is based on the *Message Passing Interface* libraries [Navarrete Navarrete et al., 2011].

We are convinced that jNEP will be useful to the researchers in the field and will be adapted to new NEPs variants thanks to its flexible design and its open source.

During our investigations, we have also made used of our own platform to study NEPs as computational tools to solve specific problems. Firstly, we focused on classical NP problems, since the intrinsic parallelism of NEPs make them specially suitable for that kind of problems. This is relevant because, as in the case of other bio-inspired computing models, NEPs can handle NP problems efficiently. Later on, we have proposed a NEP algorithm to parse context-free languages. We have seen how the special features of NEPs allows efficient top-down parsing. With this in mind, we have explored the possibility of applying the NEP model to the special difficulties of Natural Language Parsing. We have shown an example of a Parsing Network of Evolutionary Processor used for parsing an English grammar contained in a modern language toolkit (Freeling, Padró et al. [2010]).

Again, trying to ease the study, simulation and programming of Natural Computing devices, we have deeply explored Grammatical Evolution algorithms as general tools to automatically design/program bio-inspired devices. We have shown how Grammatical Evolution extensions, like Christiansen Grammar Evolution and Attribute Grammar Evolution, are capable of formally defining the constraints and features of any device and search for specific instances. The potential of these evolutionary algorithms as general tools for automatic designing/programming has been clearly identified and we have proposed the corresponding methodology. Finally, we have applied the methodology in two practical contexts.

However, after this work, there exist many aspects to improve and new research lines have appeared. We are planning to develop a complete compiler for NEP-Lingua; a code generator which translate NEPs-Lingua programs into the XML configuration files that jNEP uses as input. Furthermore, although its simple syntax seems to be expressive enough for the NEPs we have found in the literature, we are considering new extensions as parametrized sub-NEPs and others usually present in programming languages. It is also worth noticing that we are paying attention to further NEPs variants so as to include their components in the jNEP simulator.

Concerning Parsing Networks of Evolutionary Processors, we are planning to test our proposal with more realistic examples, to improve the accuracy and performance of the basic PNEP model and to incorporate syntactical analysis (both, complete and shallow) to the IBERIA corpus for Scientic Spanish [Porta et al., 2011]. Further on, we are considering to extend PNEPs with formal representations able to handle semantics (attribute grammars, for example). In the same way, we think this new parsing model can serve as a tool for compiler design and as a new approach to tackle some tasks in the semantic level of natural language processing.

Our studies on the automatic design and programming by means of Grammatical Evolution are also far from being finished. We have tested the framework in two cases and we are optimistic with respect to the feasibility of using our platform to solve more general problems in more general domains. In the future, we will try to generalize different aspects of the work described. As finding NEPs to rotate any symbol of the alphabet or searching within a more general family of NEPs by removing some of the constraints used in that first work. Moreover, we also pretend to tackle different non trivial problems. Finally, we are interested in adding to our platform a general way for describing the problem under consideration and for including it in the fitness function in a more standard way.

While classical computers are reaching their theoretical and practical limits, the new Natural Computing paradigms are destined to expand the frontiers of computer science. Molecular or quantum computers will substitute, soon or later, the classical von Neumann computer architecture implemented *in silico*. Meanwhile, Natural

Computing will keep creating innovative interplays between Nature and Computing which make us widen our conception of the potential and limits of computing. We hope that we have made a modest contribution to this exciting scientific journey.

Appendix A

jNEP configuration file for the three variables SAT problem

```
<?xml version="1.0"?>
<!-- NEP Config file-->
<!-- The character '_' is reserved since it is used to separate symbols within
words or within a set of symbols-->

<NEP nodes="9">

  <ALPHABET symbols="A_B_C_!A_!B_!C_AND_OR_(_)_[A=1]_[B=1]_[C=1]_[A=0]_[B=0]_[C=0]_#_UP_{_}_1"/>

  <GRAPH>
    <EDGE vertex1="0" vertex2="1"/>
    <EDGE vertex1="0" vertex2="2"/>
    <EDGE vertex1="0" vertex2="3"/>
    <EDGE vertex1="0" vertex2="4"/>
    <EDGE vertex1="0" vertex2="5"/>
    <EDGE vertex1="0" vertex2="6"/>
    <EDGE vertex1="0" vertex2="7"/>
    <EDGE vertex1="0" vertex2="8"/>
    <EDGE vertex1="1" vertex2="2"/>
    <EDGE vertex1="1" vertex2="3"/>
    <EDGE vertex1="1" vertex2="4"/>
    <EDGE vertex1="1" vertex2="5"/>
    <EDGE vertex1="1" vertex2="6"/>
    <EDGE vertex1="1" vertex2="7"/>
    <EDGE vertex1="1" vertex2="8"/>
    <EDGE vertex1="2" vertex2="3"/>
    <EDGE vertex1="2" vertex2="4"/>
    <EDGE vertex1="2" vertex2="5"/>
    <EDGE vertex1="2" vertex2="6"/>
    <EDGE vertex1="2" vertex2="7"/>
    <EDGE vertex1="2" vertex2="8"/>
    <EDGE vertex1="3" vertex2="4"/>
    <EDGE vertex1="3" vertex2="5"/>
    <EDGE vertex1="3" vertex2="6"/>
    <EDGE vertex1="3" vertex2="7"/>
    <EDGE vertex1="3" vertex2="8"/>
    <EDGE vertex1="4" vertex2="5"/>
    <EDGE vertex1="4" vertex2="6"/>
    <EDGE vertex1="4" vertex2="7"/>
    <EDGE vertex1="4" vertex2="8"/>
    <EDGE vertex1="5" vertex2="6"/>
    <EDGE vertex1="5" vertex2="7"/>
    <EDGE vertex1="5" vertex2="8"/>
```

```
  <EDGE vertex1="6" vertex2="7"/>
  <EDGE vertex1="6" vertex2="8"/>
  <EDGE vertex1="7" vertex2="8"/>
</GRAPH>

<STOPPING_CONDITION>
        <CONDITION type="NonEmptyNodeStoppingCondition" nodeID="1"/>
</STOPPING_CONDITION>

<EVOLUTIONARY_PROCESSORS>
  <!-- INPUT NODE -->
  <NODE initCond="{_(_A_)_AND_(_B_OR_C_)_}" auxiliaryWords="{_[A=1]_# {_[A=0]_# {_[B=1]_# {_[B=0]_# {_[C=1]_#
                                                                {_[C=0]_#">
    <EVOLUTIONARY_RULES>
      <RULE ruleType="splicing" wordX="{" wordY="(" wordU="{_[A=1]" wordV="#"/>
      <RULE ruleType="splicing" wordX="{" wordY="(" wordU="{_[A=0]" wordV="#"/>
      <RULE ruleType="splicing" wordX="{" wordY="[A=0]" wordU="{_[B=0]" wordV="#"/>
      <RULE ruleType="splicing" wordX="{" wordY="[A=0]" wordU="{_[B=1]" wordV="#"/>
      <RULE ruleType="splicing" wordX="{" wordY="[A=1]" wordU="{_[B=0]" wordV="#"/>
      <RULE ruleType="splicing" wordX="{" wordY="[A=1]" wordU="{_[B=1]" wordV="#"/>
      <RULE ruleType="splicing" wordX="{" wordY="[B=0]" wordU="{_[C=0]" wordV="#"/>
      <RULE ruleType="splicing" wordX="{" wordY="[B=0]" wordU="{_[C=1]" wordV="#"/>
      <RULE ruleType="splicing" wordX="{" wordY="[B=1]" wordU="{_[C=0]" wordV="#"/>
      <RULE ruleType="splicing" wordX="{" wordY="[B=1]" wordU="{_[C=1]" wordV="#"/>
    </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="4" permittingContext="" forbiddingContext="[A=1]_[B=1]_[C=1]_[A=0]_[B=0]_[C=0]_#_UP_{_}_1"/>
      <OUTPUT type="4" permittingContext="[C=1]_[C=0]" forbiddingContext=""/>
    </FILTERS>
  </NODE>
  <NODE initCond="">  <!-- OUTPUT NODE -->
    <EVOLUTIONARY_RULES>
    </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="1" permittingContext="" forbiddingContext="A_B_C_!A_!B_!C_AND_OR_(_)"/>
      <OUTPUT type="1" permittingContext="" forbiddingContext="[A=1]_[B=1]_[C=1]_[A=0]_[B=0]_[C=0]_#_UP_{_}_1"/>
    </FILTERS>
  </NODE>
  <NODE initCond="" auxiliaryWords="#_[A=0]_} #_[A=1]_} #_} #_1_)_}">  <!--COMP NODE -->
    <EVOLUTIONARY_RULES>
      <RULE ruleType="splicing" wordX="" wordY="A_OR_1_)_}" wordU="#" wordV="1_)_}"/>
      <RULE ruleType="splicing" wordX="" wordY="!A_OR_1_)_}" wordU="#" wordV="1_)_}"/>
      <RULE ruleType="splicing" wordX="" wordY="B_OR_1_)_}" wordU="#" wordV="1_)_}"/>
      <RULE ruleType="splicing" wordX="" wordY="!B_OR_1_)_}" wordU="#" wordV="1_)_}"/>
      <RULE ruleType="splicing" wordX="" wordY="C_OR_1_)_}" wordU="#" wordV="1_)_}"/>
      <RULE ruleType="splicing" wordX="" wordY="!C_OR_1_)_}" wordU="#" wordV="1_)_}"/>
      <RULE ruleType="splicing" wordX="" wordY="AND_(_1_)_}" wordU="#" wordV="}"/>
      <RULE ruleType="splicing" wordX="" wordY="[A=1]_(_1_)_}" wordU="#" wordV="[A=1]_}"/>
      <RULE ruleType="splicing" wordX="" wordY="[A=0]_(_1_)_}" wordU="#" wordV="[A=0]_}"/>
    </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="1" permittingContext="1" forbiddingContext=""/>
      <OUTPUT type="1" permittingContext="" forbiddingContext="#_1"/>
    </FILTERS>
  </NODE>
  <NODE initCond=""  auxiliaryWords="#_1_)_} #_)_}">  <!-- A=1 NODE -->
    <EVOLUTIONARY_RULES>
      <RULE ruleType="splicing" wordX="" wordY="A_)_}" wordU="#" wordV="1_)_}"/>
      <RULE ruleType="splicing" wordX="" wordY="(_!A_)_}" wordU="#" wordV="UP"/>
      <RULE ruleType="splicing" wordX="" wordY="OR_!A_)_}" wordU="#" wordV=")_}"/>
      <RULE ruleType="splicing" wordX="" wordY="B_)_}" wordU="#" wordV="UP"/>
      <RULE ruleType="splicing" wordX="" wordY="C_)_}" wordU="#" wordV="UP"/>
    </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="1" permittingContext="[A=1]" forbiddingContext="[A=0]_1"/>
      <OUTPUT type="1" permittingContext="" forbiddingContext="#_UP"/>
    </FILTERS>
```

```
  </NODE>
  <NODE initCond=""  auxiliaryWords="#_1_)_} #_)_}">  <!-- B=1 NODE -->
    <EVOLUTIONARY_RULES>
      <RULE ruleType="splicing" wordX="" wordY="B_)_}" wordU="#" wordV="1_)_}"/>
      <RULE ruleType="splicing" wordX="" wordY="(_!B_)_}" wordU="#" wordV="UP"/>
      <RULE ruleType="splicing" wordX="" wordY="OR_!B_)_}" wordU="#" wordV=")_}"/>
      <RULE ruleType="splicing" wordX="" wordY="A_)_}" wordU="#" wordV="UP"/>
      <RULE ruleType="splicing" wordX="" wordY="C_)_}" wordU="#" wordV="UP"/>
    </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="1" permittingContext="[B=1]" forbiddingContext="[B=0]_1"/>
      <OUTPUT type="1" permittingContext="" forbiddingContext="#_UP"/>
    </FILTERS>
  </NODE>
  <NODE initCond=""  auxiliaryWords="#_1_)_} #_)_}">  <!-- C=1 NODE -->
    <EVOLUTIONARY_RULES>
      <RULE ruleType="splicing" wordX="" wordY="C_)_}" wordU="#" wordV="1_)_}"/>
      <RULE ruleType="splicing" wordX="" wordY="(_!C_)_}" wordU="#" wordV="UP"/>
      <RULE ruleType="splicing" wordX="" wordY="OR_!C_)_}" wordU="#" wordV=")_}"/>
      <RULE ruleType="splicing" wordX="" wordY="A_)_}" wordU="#" wordV="UP"/>
      <RULE ruleType="splicing" wordX="" wordY="B_)_}" wordU="#" wordV="UP"/>
    </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="1" permittingContext="[C=1]" forbiddingContext="[C=0]_1"/>
      <OUTPUT type="1" permittingContext="" forbiddingContext="#_UP"/>
    </FILTERS>
  </NODE>
  <NODE initCond=""  auxiliaryWords="#_1_)_} #_)_}">  <!-- A=0 NODE -->
    <EVOLUTIONARY_RULES>
      <RULE ruleType="splicing" wordX="" wordY="OR_A_)_}" wordU="#" wordV=")_}"/>
      <RULE ruleType="splicing" wordX="" wordY="(_A_)_}" wordU="#" wordV="UP"/>
      <RULE ruleType="splicing" wordX="" wordY="!A_)_}" wordU="#" wordV="1"/>
      <RULE ruleType="splicing" wordX="" wordY="B_)_}" wordU="#" wordV="UP"/>
      <RULE ruleType="splicing" wordX="" wordY="C_)_}" wordU="#" wordV="UP"/>
    </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="1" permittingContext="[A=0]" forbiddingContext="[A=1]_1"/>
      <OUTPUT type="1" permittingContext="" forbiddingContext="#_UP"/>
    </FILTERS>
  </NODE>
  <NODE initCond=""  auxiliaryWords="#_1_)_} #_)_}">  <!-- B=0 NODE -->
    <EVOLUTIONARY_RULES>
      <RULE ruleType="splicing" wordX="" wordY="OR_B_)_}" wordU="#" wordV=")_}"/>
      <RULE ruleType="splicing" wordX="" wordY="(_B_)_}" wordU="#" wordV="UP"/>
      <RULE ruleType="splicing" wordX="" wordY="!B_)_}" wordU="#" wordV="1"/>
      <RULE ruleType="splicing" wordX="" wordY="A_)_}" wordU="#" wordV="UP"/>
      <RULE ruleType="splicing" wordX="" wordY="C_)_}" wordU="#" wordV="UP"/>
    </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="1" permittingContext="[B=0]" forbiddingContext="[B=1]_1"/>
      <OUTPUT type="1" permittingContext="" forbiddingContext="#_UP"/>
    </FILTERS>
  </NODE>
  <NODE initCond=""  auxiliaryWords="#_1_)_} #_)_}">  <!-- C=0 NODE -->
    <EVOLUTIONARY_RULES>
      <RULE ruleType="splicing" wordX="" wordY="OR_C_)_}" wordU="#" wordV=")_}"/>
      <RULE ruleType="splicing" wordX="" wordY="(_C_)_}" wordU="#" wordV="UP"/>
      <RULE ruleType="splicing" wordX="" wordY="!C_)_}" wordU="#" wordV="1"/>
      <RULE ruleType="splicing" wordX="" wordY="B_)_}" wordU="#" wordV="UP"/>
      <RULE ruleType="splicing" wordX="" wordY="A_)_}" wordU="#" wordV="UP"/>
    </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="1" permittingContext="[C=0]" forbiddingContext="[C=1]_1"/>
      <OUTPUT type="1" permittingContext="" forbiddingContext="#_UP"/>
    </FILTERS>
  </NODE>
</EVOLUTIONARY_PROCESSORS>
```

```
</NEP>
```

Appendix B

jNEP configuration file for the hamiltonian path problem

```
<NEP nodes="8">

  <ALPHABET symbols="i_0_1_2_3_4_5_6"/>

  <GRAPH>
    <EDGE vertex1="0" vertex2="1"/>
    <EDGE vertex1="0" vertex2="3"/>
    <EDGE vertex1="0" vertex2="6"/>
    <EDGE vertex1="1" vertex2="2"/>
    <EDGE vertex1="1" vertex2="3"/>
    <EDGE vertex1="2" vertex2="1"/>
    <EDGE vertex1="2" vertex2="3"/>
    <EDGE vertex1="3" vertex2="2"/>
    <EDGE vertex1="3" vertex2="4"/>
    <EDGE vertex1="4" vertex2="1"/>
    <EDGE vertex1="4" vertex2="5"/>
    <EDGE vertex1="5" vertex2="1"/>
    <EDGE vertex1="5" vertex2="2"/>
    <EDGE vertex1="5" vertex2="6"/>
    <EDGE vertex1="6" vertex2="7"/>
  </GRAPH>

  <EVOLUTIONARY_PROCESSORS>
    <NODE initCond="i">
      <EVOLUTIONARY_RULES>
        <RULE ruleType="insertion" actionType="RIGHT" symbol="0" newSymbol=""/>
      </EVOLUTIONARY_RULES>
      <FILTERS>
        <INPUT type="2" permittingContext="i_0_1_2_3_4_5_6" forbiddingContext=""/>
        <OUTPUT type="2" permittingContext="i_0_1_2_3_4_5_6" forbiddingContext=""/>
      </FILTERS>
    </NODE>

    <NODE initCond="">
      <EVOLUTIONARY_RULES>
        <RULE ruleType="insertion" actionType="RIGHT" symbol="1" newSymbol=""/>
      </EVOLUTIONARY_RULES>
      <FILTERS>
        <INPUT type="2" permittingContext="i_0_1_2_3_4_5_6" forbiddingContext=""/>
        <OUTPUT type="2" permittingContext="i_0_1_2_3_4_5_6" forbiddingContext=""/>
      </FILTERS>
```

```
    </NODE>

    <NODE initCond="">
      <EVOLUTIONARY_RULES>
        <RULE ruleType="insertion" actionType="RIGHT" symbol="2" newSymbol=""/>
      </EVOLUTIONARY_RULES>
      <FILTERS>
        <INPUT type="2" permittingContext="i_0_1_2_3_4_5_6" forbiddingContext=""/>
        <OUTPUT type="2" permittingContext="i_0_1_2_3_4_5_6" forbiddingContext=""/>
      </FILTERS>
    </NODE>

    <NODE initCond="">
      <EVOLUTIONARY_RULES>
        <RULE ruleType="insertion" actionType="RIGHT" symbol="3" newSymbol=""/>
      </EVOLUTIONARY_RULES>
      <FILTERS>
        <INPUT type="2" permittingContext="i_0_1_2_3_4_5_6" forbiddingContext=""/>
        <OUTPUT type="2" permittingContext="i_0_1_2_3_4_5_6" forbiddingContext=""/>
      </FILTERS>
    </NODE>

    <NODE initCond="">
      <EVOLUTIONARY_RULES>
        <RULE ruleType="insertion" actionType="RIGHT" symbol="4" newSymbol=""/>
      </EVOLUTIONARY_RULES>
      <FILTERS>
        <INPUT type="2" permittingContext="i_0_1_2_3_4_5_6" forbiddingContext=""/>
        <OUTPUT type="2" permittingContext="i_0_1_2_3_4_5_6" forbiddingContext=""/>
      </FILTERS>
    </NODE>

    <NODE initCond="">
      <EVOLUTIONARY_RULES>
        <RULE ruleType="insertion" actionType="RIGHT" symbol="5" newSymbol=""/>
      </EVOLUTIONARY_RULES>
      <FILTERS>
        <INPUT type="2" permittingContext="i_0_1_2_3_4_5_6" forbiddingContext=""/>
        <OUTPUT type="2" permittingContext="i_0_1_2_3_4_5_6" forbiddingContext=""/>
      </FILTERS>
    </NODE>

    <NODE initCond="">
      <EVOLUTIONARY_RULES>
        <RULE ruleType="insertion" actionType="RIGHT" symbol="6" newSymbol=""/>
      </EVOLUTIONARY_RULES>
      <FILTERS>
        <INPUT type="2" permittingContext="i_0_1_2_3_4_5_6" forbiddingContext=""/>
        <OUTPUT type="2" permittingContext="i_0_1_2_3_4_5_6" forbiddingContext=""/>
      </FILTERS>
    </NODE>

    <NODE initCond="">
      <EVOLUTIONARY_RULES>
      </EVOLUTIONARY_RULES>
      <FILTERS>
        <INPUT type="SetMembershipFilter" wordSet="i_0_1_2_3_4_5_6"/>
      </FILTERS>
    </NODE>
  </EVOLUTIONARY_PROCESSORS>

  <STOPPING_CONDITION>
    <CONDITION type="NonEmptyNodeStoppingCondition" nodeID="7"/>
  </STOPPING_CONDITION>
</NEP>
```

Appendix C

jNEP configuration file for the coloring problem

Some comments are inserted in square brackets.

```
<NEP nodes="51">

<ALPHABET symbols="b1_r1_g1_b2_r2_g2_b3_r3_g3_b4_r4_g4_b5_r5_g5_B1_R1_G1_B2_R2_G2_B3_R3_G3
                   _B4_R4_G4_B5_R5_G5_a1_a2_a3_a4_a5_X1_X2_X3_X4_X5_X6_X8_X9"/>

<GRAPH>
  <EDGE vertex1="0" vertex2="1"/>
  <EDGE vertex1="0" vertex2="2"/>
  <EDGE vertex1="0" vertex2="3"/>
  <EDGE vertex1="0" vertex2="4"/>
  <EDGE vertex1="0" vertex2="5"/>
  <EDGE vertex1="0" vertex2="6"/>
  <EDGE vertex1="0" vertex2="7"/>
  <EDGE vertex1="0" vertex2="8"/>
  <EDGE vertex1="0" vertex2="9"/>
  <EDGE vertex1="0" vertex2="10"/>
  <EDGE vertex1="0" vertex2="11"/>
  <EDGE vertex1="0" vertex2="12"/>
  <EDGE vertex1="0" vertex2="13"/>
  <EDGE vertex1="0" vertex2="14"/>
  <EDGE vertex1="0" vertex2="15"/>
  <EDGE vertex1="0" vertex2="16"/>
  <EDGE vertex1="0" vertex2="17"/>
  <EDGE vertex1="0" vertex2="18"/>
  <EDGE vertex1="0" vertex2="19"/>
  <EDGE vertex1="0" vertex2="20"/>
  <EDGE vertex1="0" vertex2="21"/>
  <EDGE vertex1="0" vertex2="22"/>
  <EDGE vertex1="0" vertex2="23"/>
  <EDGE vertex1="0" vertex2="24"/>
  <EDGE vertex1="0" vertex2="25"/>
  <EDGE vertex1="0" vertex2="26"/>
  <EDGE vertex1="0" vertex2="27"/>
  <EDGE vertex1="0" vertex2="28"/>
  <EDGE vertex1="0" vertex2="29"/>
  <EDGE vertex1="0" vertex2="30"/>
  <EDGE vertex1="0" vertex2="31"/>
  <EDGE vertex1="0" vertex2="32"/>
  <EDGE vertex1="0" vertex2="33"/>
  <EDGE vertex1="0" vertex2="34"/>
  <EDGE vertex1="0" vertex2="35"/>
  <EDGE vertex1="0" vertex2="36"/>
  <EDGE vertex1="0" vertex2="37"/>
```

```
<EDGE vertex1="0" vertex2="38"/>
<EDGE vertex1="0" vertex2="39"/>
<EDGE vertex1="0" vertex2="40"/>
<EDGE vertex1="0" vertex2="41"/>
<EDGE vertex1="0" vertex2="42"/>
<EDGE vertex1="0" vertex2="43"/>
<EDGE vertex1="0" vertex2="44"/>
<EDGE vertex1="0" vertex2="45"/>
<EDGE vertex1="0" vertex2="46"/>
<EDGE vertex1="0" vertex2="47"/>
<EDGE vertex1="0" vertex2="48"/>
<EDGE vertex1="0" vertex2="49"/>
<EDGE vertex1="0" vertex2="50"/>
<EDGE vertex1="1" vertex2="2"/>
<EDGE vertex1="1" vertex2="3"/>
<EDGE vertex1="1" vertex2="4"/>
<EDGE vertex1="1" vertex2="5"/>
<EDGE vertex1="1" vertex2="6"/>
<EDGE vertex1="1" vertex2="7"/>
<EDGE vertex1="1" vertex2="8"/>
<EDGE vertex1="1" vertex2="9"/>
<EDGE vertex1="1" vertex2="10"/>
<EDGE vertex1="1" vertex2="11"/>
<EDGE vertex1="1" vertex2="12"/>
<EDGE vertex1="1" vertex2="13"/>
<EDGE vertex1="1" vertex2="14"/>
<EDGE vertex1="1" vertex2="15"/>
<EDGE vertex1="1" vertex2="16"/>
<EDGE vertex1="1" vertex2="17"/>
<EDGE vertex1="1" vertex2="18"/>
<EDGE vertex1="1" vertex2="19"/>
<EDGE vertex1="1" vertex2="20"/>
<EDGE vertex1="1" vertex2="21"/>
<EDGE vertex1="1" vertex2="22"/>
<EDGE vertex1="1" vertex2="23"/>
<EDGE vertex1="1" vertex2="24"/>
<EDGE vertex1="1" vertex2="25"/>
<EDGE vertex1="1" vertex2="26"/>
<EDGE vertex1="1" vertex2="27"/>
<EDGE vertex1="1" vertex2="28"/>
<EDGE vertex1="1" vertex2="29"/>
<EDGE vertex1="1" vertex2="30"/>
<EDGE vertex1="1" vertex2="31"/>
<EDGE vertex1="1" vertex2="32"/>
<EDGE vertex1="1" vertex2="33"/>
<EDGE vertex1="1" vertex2="34"/>
<EDGE vertex1="1" vertex2="35"/>
<EDGE vertex1="1" vertex2="36"/>
<EDGE vertex1="1" vertex2="37"/>
<EDGE vertex1="1" vertex2="38"/>
<EDGE vertex1="1" vertex2="39"/>
<EDGE vertex1="1" vertex2="40"/>
<EDGE vertex1="1" vertex2="41"/>
<EDGE vertex1="1" vertex2="42"/>
<EDGE vertex1="1" vertex2="43"/>
<EDGE vertex1="1" vertex2="44"/>
<EDGE vertex1="1" vertex2="45"/>
<EDGE vertex1="1" vertex2="46"/>
<EDGE vertex1="1" vertex2="47"/>
<EDGE vertex1="1" vertex2="48"/>
<EDGE vertex1="1" vertex2="49"/>
<EDGE vertex1="1" vertex2="50"/>
```


[And so on, until we get a complete graph.]

[...]

```
</GRAPH>


  <EVOLUTIONARY_PROCESSORS>
    <NODE initCond="a1_a2_a3_a4_a5_X1">  <!-- NODO 0 INICIO -->
      <EVOLUTIONARY_RULES>
        <RULE ruleType="substitution" actionType="ANY" symbol="a1" newSymbol="b1"/>
        <RULE ruleType="substitution" actionType="ANY" symbol="a1" newSymbol="r1"/>
        <RULE ruleType="substitution" actionType="ANY" symbol="a1" newSymbol="g1"/>
        <RULE ruleType="substitution" actionType="ANY" symbol="a2" newSymbol="b2"/>
        <RULE ruleType="substitution" actionType="ANY" symbol="a2" newSymbol="r2"/>
        <RULE ruleType="substitution" actionType="ANY" symbol="a2" newSymbol="g2"/>
        <RULE ruleType="substitution" actionType="ANY" symbol="a3" newSymbol="b3"/>
        <RULE ruleType="substitution" actionType="ANY" symbol="a3" newSymbol="r3"/>
        <RULE ruleType="substitution" actionType="ANY" symbol="a3" newSymbol="g3"/>
        <RULE ruleType="substitution" actionType="ANY" symbol="a4" newSymbol="b4"/>
        <RULE ruleType="substitution" actionType="ANY" symbol="a4" newSymbol="r4"/>
        <RULE ruleType="substitution" actionType="ANY" symbol="a4" newSymbol="g4"/>
        <RULE ruleType="substitution" actionType="ANY" symbol="a5" newSymbol="b5"/>
        <RULE ruleType="substitution" actionType="ANY" symbol="a5" newSymbol="r5"/>
        <RULE ruleType="substitution" actionType="ANY" symbol="a5" newSymbol="g5"/>
      </EVOLUTIONARY_RULES>
      <FILTERS>
        <INPUT type="1" permittingContext="a1_a2_a3_a4_a5_X1" forbiddingContext=""/>
        <OUTPUT type="1" permittingContext="" forbiddingContext="a1_a2_a3_a4_a5"/>
      </FILTERS>
    </NODE>

    <NODE initCond=""> <!-- NODO 1 SALIDA -->
      <EVOLUTIONARY_RULES>
        <RULE ruleType="deletion" actionType="ANY" symbol="X9"/>
      </EVOLUTIONARY_RULES>
      <FILTERS>
        <INPUT type="1" permittingContext="X9" forbiddingContext="B1_R1_G1_B2_R2_G2_B3_R3_G3_B4_R4_G4_B5_R5_G5
                                                                  _a1_a2_a3_a4_a5_X1_X2_X3_X4_X5_X6_X8"/>
        <OUTPUT type="1" permittingContext="" forbiddingContext="b1_r1_g1_b2_r2_g2_b3_r3_g3_b4_r4_g4_b5_r5_g5"/>
      </FILTERS>
    </NODE>

    <NODE initCond=""> <!-- NODO 2 ARISTA t=1 Z  b ( 1 , 2 )  -->
      <EVOLUTIONARY_RULES>
        <RULE ruleType="substitution" actionType="ANY" symbol="b1" newSymbol="B1"/>
      </EVOLUTIONARY_RULES>
      <FILTERS>
        <INPUT type="1" permittingContext="X1" forbiddingContext="B1_R1_G1_B2_R2_G2_B3_R3_G3_B4_R4_G4_B5_R5_G5
                                                                  _a1_a2_a3_a4_a5_X2_X3_X4_X5_X6_X8_X9"/>
        <OUTPUT type="1" permittingContext="B1" forbiddingContext=""/>
      </FILTERS>
    </NODE>

    <NODE initCond=""> <!-- NODO 3 ARISTA t=1 Z r  ( 1 , 2 )  -->
      <EVOLUTIONARY_RULES>
        <RULE ruleType="substitution" actionType="ANY" symbol="r1" newSymbol="R1"/>
      </EVOLUTIONARY_RULES>
      <FILTERS>
        <INPUT type="1" permittingContext="X1" forbiddingContext="B1_R1_G1_B2_R2_G2_B3_R3_G3_B4_R4_G4_B5_R5_G5
                                                                  _a1_a2_a3_a4_a5_X2_X3_X4_X5_X6_X8_X9"/>
        <OUTPUT type="1" permittingContext="R1" forbiddingContext=""/>
      </FILTERS>
    </NODE>

    <NODE initCond=""> <!-- NODO 4 ARISTA t=1 Z g  ( 1 , 2 )  -->
      <EVOLUTIONARY_RULES>
        <RULE ruleType="substitution" actionType="ANY" symbol="g1" newSymbol="G1"/>
      </EVOLUTIONARY_RULES>
      <FILTERS>
        <INPUT type="1" permittingContext="X1" forbiddingContext="B1_R1_G1_B2_R2_G2_B3_R3_G3_B4_R4_G4_B5_R5_G5
```

```
                                                            _a1_a2_a3_a4_a5_X2_X3_X4_X5_X6_X8_X9"/>
      <OUTPUT type="1" permittingContext="G1" forbiddingContext=""/>
    </FILTERS>
  </NODE>

  <NODE initCond=""> <!-- NODO 5 ARISTA t=1 B  ( 1 , 2 ) -->
    <EVOLUTIONARY_RULES>
      <RULE ruleType="substitution" actionType="ANY" symbol="r2" newSymbol="R2"/>
      <RULE ruleType="substitution" actionType="ANY" symbol="g2" newSymbol="G2"/>
    </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="1" permittingContext="X1_B1" forbiddingContext=""/>
      <OUTPUT type="1" permittingContext="" forbiddingContext="b1_r1_g1_b2_r2_g2"/>
    </FILTERS>
  </NODE>

  <NODE initCond=""> <!-- NODO 6 ARISTA t=1 R  ( 1 , 2 ) -->
    <EVOLUTIONARY_RULES>
      <RULE ruleType="substitution" actionType="ANY" symbol="b2" newSymbol="B2"/>
      <RULE ruleType="substitution" actionType="ANY" symbol="g2" newSymbol="G2"/>
    </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="1" permittingContext="X1_R1" forbiddingContext=""/>
      <OUTPUT type="1" permittingContext="" forbiddingContext="b1_r1_g1_b2_r2_g2"/>
    </FILTERS>
  </NODE>

  <NODE initCond=""> <!-- NODO 7 ARISTA t=1 G  ( 1 , 2 ) -->
    <EVOLUTIONARY_RULES>
      <RULE ruleType="substitution" actionType="ANY" symbol="r2" newSymbol="R2"/>
      <RULE ruleType="substitution" actionType="ANY" symbol="b2" newSymbol="B2"/>
    </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="1" permittingContext="X1_G1" forbiddingContext=""/>
      <OUTPUT type="1" permittingContext="" forbiddingContext="b1_r1_g1_b2_r2_g2"/>
    </FILTERS>
  </NODE>

  <NODE initCond=""> <!-- NODO 8 ARISTA t=1 ( 1 , 2 ) -->
    <EVOLUTIONARY_RULES>
      <RULE ruleType="substitution" actionType="ANY" symbol="R1" newSymbol="r1"/>
      <RULE ruleType="substitution" actionType="ANY" symbol="B1" newSymbol="b1"/>
      <RULE ruleType="substitution" actionType="ANY" symbol="G1" newSymbol="g1"/>
      <RULE ruleType="substitution" actionType="ANY" symbol="R2" newSymbol="r2"/>
      <RULE ruleType="substitution" actionType="ANY" symbol="B2" newSymbol="b2"/>
      <RULE ruleType="substitution" actionType="ANY" symbol="G2" newSymbol="g2"/>
      <RULE ruleType="substitution" actionType="ANY" symbol="X1" newSymbol="X2"/>
    </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="1" permittingContext="X1" forbiddingContext="r1_b1_g1_r2_b2_g2"/>
      <OUTPUT type="1" permittingContext="X2" forbiddingContext="B1_R1_G1_B2_R2_G2_B3_R3_G3_B4_R4_G4_B5_R5_G5
                                                             _a1_a2_a3_a4_a5_X1_X3_X4_X5_X6_X8_X9"/>
    </FILTERS>
  </NODE>

  <NODE initCond=""> <!-- NODO 9 ARISTA t=2 Z b  ( 1 , 3 )  -->
    <EVOLUTIONARY_RULES>
      <RULE ruleType="substitution" actionType="ANY" symbol="b1" newSymbol="B1"/>
    </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="1" permittingContext="X2" forbiddingContext="B1_R1_G1_B2_R2_G2_B3_R3_G3_B4_R4_G4_B5_R5_G5
                                                            _a1_a2_a3_a4_a5_X1_X3_X4_X5_X6_X8_X9"/>
      <OUTPUT type="1" permittingContext="B1" forbiddingContext=""/>
    </FILTERS>
  </NODE>

  <NODE initCond=""> <!-- NODO 10 ARISTA t=2 Z r  ( 1 , 3 )  -->
```

```
  <EVOLUTIONARY_RULES>
    <RULE ruleType="substitution" actionType="ANY" symbol="r1" newSymbol="R1"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="X2" forbiddingContext="B1_R1_G1_B2_R2_G2_B3_R3_G3_B4_R4_G4_B5_R5_G5
                                                    _a1_a2_a3_a4_a5_X1_X3_X4_X5_X6_X8_X9"/>
    <OUTPUT type="1" permittingContext="R1" forbiddingContext=""/>
  </FILTERS>
</NODE>

<NODE initCond=""> <!-- NODO 11 ARISTA t=2 Z g  ( 1 , 3 )  -->
  <EVOLUTIONARY_RULES>
    <RULE ruleType="substitution" actionType="ANY" symbol="g1" newSymbol="G1"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="X2" forbiddingContext="B1_R1_G1_B2_R2_G2_B3_R3_G3_B4_R4_G4_B5_R5_G5
                                                    _a1_a2_a3_a4_a5_X1_X3_X4_X5_X6_X8_X9"/>
    <OUTPUT type="1" permittingContext="G1" forbiddingContext=""/>
  </FILTERS>
</NODE>

<NODE initCond=""> <!-- NODO 12 ARISTA t=2 B  ( 1 , 3 ) -->
  <EVOLUTIONARY_RULES>
    <RULE ruleType="substitution" actionType="ANY" symbol="r3" newSymbol="R3"/>
    <RULE ruleType="substitution" actionType="ANY" symbol="g3" newSymbol="G3"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="X2_B1" forbiddingContext=""/>
    <OUTPUT type="1" permittingContext="" forbiddingContext="b1_r1_g1_b3_r3_g3"/>
  </FILTERS>
</NODE>

<NODE initCond=""> <!-- NODO 13 ARISTA t=2 R  ( 1 , 3 ) -->
  <EVOLUTIONARY_RULES>
    <RULE ruleType="substitution" actionType="ANY" symbol="b3" newSymbol="B3"/>
    <RULE ruleType="substitution" actionType="ANY" symbol="g3" newSymbol="G3"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="X2_R1" forbiddingContext=""/>
    <OUTPUT type="1" permittingContext="" forbiddingContext="b1_r1_g1_b3_r3_g3"/>
  </FILTERS>
</NODE>

<NODE initCond=""> <!-- NODO 14 ARISTA t=2 G  ( 1 , 3 ) -->
  <EVOLUTIONARY_RULES>
    <RULE ruleType="substitution" actionType="ANY" symbol="r3" newSymbol="R3"/>
    <RULE ruleType="substitution" actionType="ANY" symbol="b3" newSymbol="B3"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="X2_G1" forbiddingContext=""/>
    <OUTPUT type="1" permittingContext="" forbiddingContext="b1_r1_g1_b3_r3_g3"/>
  </FILTERS>
</NODE>

<NODE initCond=""> <!-- NODO 15 ARISTA t=2 ( 1 , 3 ) -->
  <EVOLUTIONARY_RULES>
    <RULE ruleType="substitution" actionType="ANY" symbol="R1" newSymbol="r1"/>
    <RULE ruleType="substitution" actionType="ANY" symbol="B1" newSymbol="b1"/>
    <RULE ruleType="substitution" actionType="ANY" symbol="G1" newSymbol="g1"/>
    <RULE ruleType="substitution" actionType="ANY" symbol="R3" newSymbol="r3"/>
    <RULE ruleType="substitution" actionType="ANY" symbol="B3" newSymbol="b3"/>
    <RULE ruleType="substitution" actionType="ANY" symbol="G3" newSymbol="g3"/>
    <RULE ruleType="substitution" actionType="ANY" symbol="X2" newSymbol="X3"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="X2" forbiddingContext="r1_b1_g1_r3_b3_g3"/>
    <OUTPUT type="1" permittingContext="X3" forbiddingContext="B1_R1_G1_B2_R2_G2_B3_R3_G3_B4_R4_G4_B5_R5_G5
```

```
                                                                     _a1_a2_a3_a4_a5_X1_X2_X4_X5_X6_X8_X9"/>
  </FILTERS>
</NODE>

<NODE initCond=""> <!-- NODO 16 ARISTA t=3 Z b  ( 1 , 4 )  -->
  <EVOLUTIONARY_RULES>
    <RULE ruleType="substitution" actionType="ANY" symbol="b1" newSymbol="B1"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="X3" forbiddingContext="B1_R1_G1_B2_R2_G2_B3_R3_G3_B4_R4_G4_B5_R5_G5
                                                            _a1_a2_a3_a4_a5_X1_X2_X4_X5_X6_X8_X9"/>
    <OUTPUT type="1" permittingContext="B1" forbiddingContext=""/>
  </FILTERS>
</NODE>

<NODE initCond=""> <!-- NODO 17 ARISTA t=3 Z r  ( 1 , 4 )  -->
  <EVOLUTIONARY_RULES>
    <RULE ruleType="substitution" actionType="ANY" symbol="r1" newSymbol="R1"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="X3" forbiddingContext="B1_R1_G1_B2_R2_G2_B3_R3_G3_B4_R4_G4_B5_R5_G5
                                                            _a1_a2_a3_a4_a5_X1_X2_X4_X5_X6_X8_X9"/>
    <OUTPUT type="1" permittingContext="R1" forbiddingContext=""/>
  </FILTERS>
</NODE>

<NODE initCond=""> <!-- NODO 18 ARISTA t=3 Z g  ( 1 , 4 )  -->
  <EVOLUTIONARY_RULES>
    <RULE ruleType="substitution" actionType="ANY" symbol="g1" newSymbol="G1"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="X3" forbiddingContext="B1_R1_G1_B2_R2_G2_B3_R3_G3_B4_R4_G4_B5_R5_G5
                                                            _a1_a2_a3_a4_a5_X1_X2_X4_X5_X6_X8_X9"/>
    <OUTPUT type="1" permittingContext="G1" forbiddingContext=""/>
  </FILTERS>
</NODE>

<NODE initCond=""> <!-- NODO 19 ARISTA t=3 B  ( 1 , 4 ) -->
  <EVOLUTIONARY_RULES>
    <RULE ruleType="substitution" actionType="ANY" symbol="r4" newSymbol="R4"/>
    <RULE ruleType="substitution" actionType="ANY" symbol="g4" newSymbol="G4"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="X3_B1" forbiddingContext=""/>
    <OUTPUT type="1" permittingContext="" forbiddingContext="b1_r1_g1_b4_r4_g4"/>
  </FILTERS>
</NODE>

<NODE initCond=""> <!-- NODO 20 ARISTA t=3 R  ( 1 , 4 ) -->
  <EVOLUTIONARY_RULES>
    <RULE ruleType="substitution" actionType="ANY" symbol="b4" newSymbol="B4"/>
    <RULE ruleType="substitution" actionType="ANY" symbol="g4" newSymbol="G4"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="X3_R1" forbiddingContext=""/>
    <OUTPUT type="1" permittingContext="" forbiddingContext="b1_r1_g1_b4_r4_g4"/>
  </FILTERS>
</NODE>

<NODE initCond=""> <!-- NODO 21 ARISTA t=3 G  ( 1 , 4 ) -->
  <EVOLUTIONARY_RULES>
    <RULE ruleType="substitution" actionType="ANY" symbol="r4" newSymbol="R4"/>
    <RULE ruleType="substitution" actionType="ANY" symbol="b4" newSymbol="B4"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="X3_G1" forbiddingContext=""/>
    <OUTPUT type="1" permittingContext="" forbiddingContext="b1_r1_g1_b4_r4_g4"/>
```

```
  </FILTERS>
</NODE>

<NODE initCond=""> <!-- NODO 22 ARISTA t=3 ( 1 , 4 ) -->
  <EVOLUTIONARY_RULES>
    <RULE ruleType="substitution" actionType="ANY" symbol="R1" newSymbol="r1"/>
    <RULE ruleType="substitution" actionType="ANY" symbol="B1" newSymbol="b1"/>
    <RULE ruleType="substitution" actionType="ANY" symbol="G1" newSymbol="g1"/>
    <RULE ruleType="substitution" actionType="ANY" symbol="R4" newSymbol="r4"/>
    <RULE ruleType="substitution" actionType="ANY" symbol="B4" newSymbol="b4"/>
    <RULE ruleType="substitution" actionType="ANY" symbol="G4" newSymbol="g4"/>
    <RULE ruleType="substitution" actionType="ANY" symbol="X3" newSymbol="X4"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="X3" forbiddingContext="r1_b1_g1_r4_b4_g4"/>
    <OUTPUT type="1" permittingContext="X4" forbiddingContext="B1_R1_G1_B2_R2_G2_B3_R3_G3_B4_R4_G4_B5_R5_G5
                                                       _a1_a2_a3_a4_a5_X1_X2_X3_X5_X6_X8_X9"/>
  </FILTERS>
</NODE>

<NODE initCond=""> <!-- NODO 23 ARISTA t=4 Z b  ( 2 , 3 )  -->
  <EVOLUTIONARY_RULES>
    <RULE ruleType="substitution" actionType="ANY" symbol="b2" newSymbol="B2"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="X4" forbiddingContext="B1_R1_G1_B2_R2_G2_B3_R3_G3_B4_R4_G4_B5_R5_G5
                                                      _a1_a2_a3_a4_a5_X1_X2_X3_X5_X6_X8_X9"/>
    <OUTPUT type="1" permittingContext="B2" forbiddingContext=""/>
  </FILTERS>
</NODE>

<NODE initCond=""> <!-- NODO 24 ARISTA t=4 Z r  ( 2 , 3 )  -->
  <EVOLUTIONARY_RULES>
    <RULE ruleType="substitution" actionType="ANY" symbol="r2" newSymbol="R2"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="X4" forbiddingContext="B1_R1_G1_B2_R2_G2_B3_R3_G3_B4_R4_G4_B5_R5_G5
                                                      _a1_a2_a3_a4_a5_X1_X2_X3_X5_X6_X8_X9"/>
    <OUTPUT type="1" permittingContext="R2" forbiddingContext=""/>
  </FILTERS>
</NODE>

<NODE initCond=""> <!-- NODO 25 ARISTA t=4 Z g  ( 2 , 3 )  -->
  <EVOLUTIONARY_RULES>
    <RULE ruleType="substitution" actionType="ANY" symbol="g2" newSymbol="G2"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="X4" forbiddingContext="B1_R1_G1_B2_R2_G2_B3_R3_G3_B4_R4_G4_B5_R5_G5
                                                      _a1_a2_a3_a4_a5_X1_X2_X3_X5_X6_X8_X9"/>
    <OUTPUT type="1" permittingContext="G2" forbiddingContext=""/>
  </FILTERS>
</NODE>

<NODE initCond=""> <!-- NODO 26 ARISTA t=4 B  ( 2 , 3 )  -->
  <EVOLUTIONARY_RULES>
    <RULE ruleType="substitution" actionType="ANY" symbol="r3" newSymbol="R3"/>
    <RULE ruleType="substitution" actionType="ANY" symbol="g3" newSymbol="G3"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="X4_B2" forbiddingContext=""/>
    <OUTPUT type="1" permittingContext="" forbiddingContext="b2_r2_g2_b3_r3_g3"/>
  </FILTERS>
</NODE>

<NODE initCond=""> <!-- NODO 27 ARISTA t=4 R  ( 2 , 3 )  -->
  <EVOLUTIONARY_RULES>
    <RULE ruleType="substitution" actionType="ANY" symbol="b3" newSymbol="B3"/>
```

```
      <RULE ruleType="substitution" actionType="ANY" symbol="g3" newSymbol="G3"/>
    </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="1" permittingContext="X4_R2" forbiddingContext=""/>
      <OUTPUT type="1" permittingContext="" forbiddingContext="b2_r2_g2_b3_r3_g3"/>
    </FILTERS>
  </NODE>

  <NODE initCond=""> <!-- NODO 28 ARISTA t=4 G  ( 2 , 3 )  -->
    <EVOLUTIONARY_RULES>
      <RULE ruleType="substitution" actionType="ANY" symbol="r3" newSymbol="R3"/>
      <RULE ruleType="substitution" actionType="ANY" symbol="b3" newSymbol="B3"/>
    </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="1" permittingContext="X4_G2" forbiddingContext=""/>
      <OUTPUT type="1" permittingContext="" forbiddingContext="b2_r2_g2_b3_r3_g3"/>
    </FILTERS>
  </NODE>

  <NODE initCond=""> <!-- NODO 29 ARISTA t=4  ( 2 , 3 )  -->
    <EVOLUTIONARY_RULES>
      <RULE ruleType="substitution" actionType="ANY" symbol="R2" newSymbol="r2"/>
      <RULE ruleType="substitution" actionType="ANY" symbol="B2" newSymbol="b2"/>
      <RULE ruleType="substitution" actionType="ANY" symbol="G2" newSymbol="g2"/>
      <RULE ruleType="substitution" actionType="ANY" symbol="R3" newSymbol="r3"/>
      <RULE ruleType="substitution" actionType="ANY" symbol="B3" newSymbol="b3"/>
      <RULE ruleType="substitution" actionType="ANY" symbol="G3" newSymbol="g3"/>
      <RULE ruleType="substitution" actionType="ANY" symbol="X4" newSymbol="X5"/>
    </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="1" permittingContext="X4" forbiddingContext="r2_b2_g2_r3_b3_g3"/>
      <OUTPUT type="1" permittingContext="X5" forbiddingContext="B1_R1_G1_B2_R2_G2_B3_R3_G3_B4_R4_G4_B5_R5_G5
                                                                _a1_a2_a3_a4_a5_X1_X2_X3_X4_X6_X8_X9"/>
    </FILTERS>
  </NODE>

  <NODE initCond=""> <!-- NODO 30 ARISTA t=5 Z b  ( 2 , 4 )  -->
    <EVOLUTIONARY_RULES>
      <RULE ruleType="substitution" actionType="ANY" symbol="b2" newSymbol="B2"/>
    </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="1" permittingContext="X5" forbiddingContext="B1_R1_G1_B2_R2_G2_B3_R3_G3_B4_R4_G4_B5_R5_G5
                                                               _a1_a2_a3_a4_a5_X1_X2_X3_X4_X6_X8_X9"/>
      <OUTPUT type="1" permittingContext="B2" forbiddingContext=""/>
    </FILTERS>
  </NODE>

  <NODE initCond=""> <!-- NODO 31 ARISTA t=5 Z r  ( 2 , 4 )  -->
    <EVOLUTIONARY_RULES>
      <RULE ruleType="substitution" actionType="ANY" symbol="r2" newSymbol="R2"/>
    </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="1" permittingContext="X5" forbiddingContext="B1_R1_G1_B2_R2_G2_B3_R3_G3_B4_R4_G4_B5_R5_G5
                                                               _a1_a2_a3_a4_a5_X1_X2_X3_X4_X6_X8_X9"/>
      <OUTPUT type="1" permittingContext="R2" forbiddingContext=""/>
    </FILTERS>
  </NODE>

  <NODE initCond=""> <!-- NODO 32 ARISTA t=5 Z g  ( 2 , 4 )  -->
    <EVOLUTIONARY_RULES>
      <RULE ruleType="substitution" actionType="ANY" symbol="g2" newSymbol="G2"/>
    </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="1" permittingContext="X5" forbiddingContext="B1_R1_G1_B2_R2_G2_B3_R3_G3_B4_R4_G4_B5_R5_G5
                                                               _a1_a2_a3_a4_a5_X1_X2_X3_X4_X6_X8_X9"/>
      <OUTPUT type="1" permittingContext="G2" forbiddingContext=""/>
    </FILTERS>
```

```
</NODE>

<NODE initCond=""> <!-- NODO 33 ARISTA t=5 B  ( 2 , 4 )  -->
  <EVOLUTIONARY_RULES>
    <RULE ruleType="substitution" actionType="ANY" symbol="r4" newSymbol="R4"/>
    <RULE ruleType="substitution" actionType="ANY" symbol="g4" newSymbol="G4"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="X5_B2" forbiddingContext=""/>
    <OUTPUT type="1" permittingContext="" forbiddingContext="b2_r2_g2_b4_r4_g4"/>
  </FILTERS>
</NODE>

<NODE initCond=""> <!-- NODO 34 ARISTA t=5 R  ( 2 , 4 )  -->
  <EVOLUTIONARY_RULES>
    <RULE ruleType="substitution" actionType="ANY" symbol="b4" newSymbol="B4"/>
    <RULE ruleType="substitution" actionType="ANY" symbol="g4" newSymbol="G4"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="X5_R2" forbiddingContext=""/>
    <OUTPUT type="1" permittingContext="" forbiddingContext="b2_r2_g2_b4_r4_g4"/>
  </FILTERS>
</NODE>

<NODE initCond=""> <!-- NODO 35 ARISTA t=5 G  ( 2 , 4 )  -->
  <EVOLUTIONARY_RULES>
    <RULE ruleType="substitution" actionType="ANY" symbol="r4" newSymbol="R4"/>
    <RULE ruleType="substitution" actionType="ANY" symbol="b4" newSymbol="B4"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="X5_G2" forbiddingContext=""/>
    <OUTPUT type="1" permittingContext="" forbiddingContext="b2_r2_g2_b4_r4_g4"/>
  </FILTERS>
</NODE>

<NODE initCond=""> <!-- NODO 36 ARISTA t=5  ( 2 , 4 )  -->
  <EVOLUTIONARY_RULES>
    <RULE ruleType="substitution" actionType="ANY" symbol="R2" newSymbol="r2"/>
    <RULE ruleType="substitution" actionType="ANY" symbol="B2" newSymbol="b2"/>
    <RULE ruleType="substitution" actionType="ANY" symbol="G2" newSymbol="g2"/>
    <RULE ruleType="substitution" actionType="ANY" symbol="R4" newSymbol="r4"/>
    <RULE ruleType="substitution" actionType="ANY" symbol="B4" newSymbol="b4"/>
    <RULE ruleType="substitution" actionType="ANY" symbol="G4" newSymbol="g4"/>
    <RULE ruleType="substitution" actionType="ANY" symbol="X5" newSymbol="X6"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="X5" forbiddingContext="r2_b2_g2_r4_b4_g4"/>
    <OUTPUT type="1" permittingContext="X6" forbiddingContext="B1_R1_G1_B2_R2_G2_B3_R3_G3_B4_R4_G4_B5_R5_G5
                                                             _a1_a2_a3_a4_a5_X1_X2_X3_X4_X5_X8_X9"/>
  </FILTERS>
</NODE>

<NODE initCond=""> <!-- NODO 37 ARISTA t=6 Z b  ( 2 , 5 )  -->
  <EVOLUTIONARY_RULES>
    <RULE ruleType="substitution" actionType="ANY" symbol="b2" newSymbol="B2"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="X6" forbiddingContext="B1_R1_G1_B2_R2_G2_B3_R3_G3_B4_R4_G4_B5_R5_G5
                                                            _a1_a2_a3_a4_a5_X1_X2_X3_X4_X5_X8_X9"/>
    <OUTPUT type="1" permittingContext="B2" forbiddingContext=""/>
  </FILTERS>
</NODE>

<NODE initCond=""> <!-- NODO 38 ARISTA t=6 Z r  ( 2 , 5 )  -->
  <EVOLUTIONARY_RULES>
    <RULE ruleType="substitution" actionType="ANY" symbol="r2" newSymbol="R2"/>
  </EVOLUTIONARY_RULES>
```

```
    <FILTERS>
      <INPUT type="1" permittingContext="X6" forbiddingContext="B1_R1_G1_B2_R2_G2_B3_R3_G3_B4_R4_G4_B5_R5_G5
                                                         _a1_a2_a3_a4_a5_X1_X2_X3_X4_X5_X8_X9"/>
      <OUTPUT type="1" permittingContext="R2" forbiddingContext=""/>
    </FILTERS>
  </NODE>

  <NODE initCond=""> <!-- NODO 39 ARISTA t=6 Z g  ( 2 , 5 )  -->
    <EVOLUTIONARY_RULES>
      <RULE ruleType="substitution" actionType="ANY" symbol="g2" newSymbol="G2"/>
    </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="1" permittingContext="X6" forbiddingContext="B1_R1_G1_B2_R2_G2_B3_R3_G3_B4_R4_G4_B5_R5_G5
                                                         _a1_a2_a3_a4_a5_X1_X2_X3_X4_X5_X8_X9"/>
      <OUTPUT type="1" permittingContext="G2" forbiddingContext=""/>
    </FILTERS>
  </NODE>

  <NODE initCond=""> <!-- NODO 40 ARISTA t=6 B  ( 2 , 5 )  -->
    <EVOLUTIONARY_RULES>
      <RULE ruleType="substitution" actionType="ANY" symbol="r5" newSymbol="R5"/>
      <RULE ruleType="substitution" actionType="ANY" symbol="g5" newSymbol="G5"/>
    </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="1" permittingContext="X6_B2" forbiddingContext=""/>
      <OUTPUT type="1" permittingContext="" forbiddingContext="b2_r2_g2_b5_r5_g5"/>
    </FILTERS>
  </NODE>

  <NODE initCond=""> <!-- NODO 41 ARISTA t=6 R  ( 2 , 5 )  -->
    <EVOLUTIONARY_RULES>
      <RULE ruleType="substitution" actionType="ANY" symbol="b5" newSymbol="B5"/>
      <RULE ruleType="substitution" actionType="ANY" symbol="g5" newSymbol="G5"/>
    </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="1" permittingContext="X6_R2" forbiddingContext=""/>
      <OUTPUT type="1" permittingContext="" forbiddingContext="b2_r2_g2_b5_r5_g5"/>
    </FILTERS>
  </NODE>

  <NODE initCond=""> <!-- NODO 42 ARISTA t=6 G  ( 2 , 5 )  -->
    <EVOLUTIONARY_RULES>
      <RULE ruleType="substitution" actionType="ANY" symbol="r5" newSymbol="R5"/>
      <RULE ruleType="substitution" actionType="ANY" symbol="b5" newSymbol="B5"/>
    </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="1" permittingContext="X6_G2" forbiddingContext=""/>
      <OUTPUT type="1" permittingContext="" forbiddingContext="b2_r2_g2_b5_r5_g5"/>
    </FILTERS>
  </NODE>

  <NODE initCond=""> <!-- NODO 43 ARISTA t=6  ( 2 , 5 )  -->
    <EVOLUTIONARY_RULES>
      <RULE ruleType="substitution" actionType="ANY" symbol="R2" newSymbol="r2"/>
      <RULE ruleType="substitution" actionType="ANY" symbol="B2" newSymbol="b2"/>
      <RULE ruleType="substitution" actionType="ANY" symbol="G2" newSymbol="g2"/>
      <RULE ruleType="substitution" actionType="ANY" symbol="R5" newSymbol="r5"/>
      <RULE ruleType="substitution" actionType="ANY" symbol="B5" newSymbol="b5"/>
      <RULE ruleType="substitution" actionType="ANY" symbol="G5" newSymbol="g5"/>
      <RULE ruleType="substitution" actionType="ANY" symbol="X6" newSymbol="X8"/>
    </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="1" permittingContext="X6" forbiddingContext="r2_b2_g2_r5_b5_g5"/>
      <OUTPUT type="1" permittingContext="X8" forbiddingContext="B1_R1_G1_B2_R2_G2_B3_R3_G3_B4_R4_G4_B5_R5_G5
                                                          _a1_a2_a3_a4_a5_X1_X2_X3_X4_X5_X6_X9"/>
    </FILTERS>
  </NODE>
```

```
<NODE initCond=""> <!-- NODO 44 ARISTA t=8 Z b  ( 4 , 5 )  -->
  <EVOLUTIONARY_RULES>
    <RULE ruleType="substitution" actionType="ANY" symbol="b4" newSymbol="B4"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="X8" forbiddingContext="B1_R1_G1_B2_R2_G2_B3_R3_G3_B4_R4_G4_B5_R5_G5
                                                             _a1_a2_a3_a4_a5_X1_X2_X3_X4_X5_X6_X9"/>
    <OUTPUT type="1" permittingContext="B4" forbiddingContext=""/>
  </FILTERS>
</NODE>

<NODE initCond=""> <!-- NODO 45 ARISTA t=8 Z r  ( 4 , 5 )  -->
  <EVOLUTIONARY_RULES>
    <RULE ruleType="substitution" actionType="ANY" symbol="r4" newSymbol="R4"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="X8" forbiddingContext="B1_R1_G1_B2_R2_G2_B3_R3_G3_B4_R4_G4_B5_R5_G5
                                                             _a1_a2_a3_a4_a5_X1_X2_X3_X4_X5_X6_X9"/>
    <OUTPUT type="1" permittingContext="R4" forbiddingContext=""/>
  </FILTERS>
</NODE>

<NODE initCond=""> <!-- NODO 46 ARISTA t=8 Z g  ( 4 , 5 )  -->
  <EVOLUTIONARY_RULES>
    <RULE ruleType="substitution" actionType="ANY" symbol="g4" newSymbol="G4"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="X8" forbiddingContext="B1_R1_G1_B2_R2_G2_B3_R3_G3_B4_R4_G4_B5_R5_G5
                                                             _a1_a2_a3_a4_a5_X1_X2_X3_X4_X5_X6_X9"/>
    <OUTPUT type="1" permittingContext="G4" forbiddingContext=""/>
  </FILTERS>
</NODE>

<NODE initCond=""> <!-- NODO 47 ARISTA t=8 B  ( 4 , 5 )  -->
  <EVOLUTIONARY_RULES>
    <RULE ruleType="substitution" actionType="ANY" symbol="r5" newSymbol="R5"/>
    <RULE ruleType="substitution" actionType="ANY" symbol="g5" newSymbol="G5"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="X8_B4" forbiddingContext=""/>
    <OUTPUT type="1" permittingContext="" forbiddingContext="b4_r4_g4_b5_r5_g5"/>
  </FILTERS>
</NODE>

<NODE initCond=""> <!-- NODO 48 ARISTA t=8 R  ( 4 , 5 )  -->
  <EVOLUTIONARY_RULES>
    <RULE ruleType="substitution" actionType="ANY" symbol="b5" newSymbol="B5"/>
    <RULE ruleType="substitution" actionType="ANY" symbol="g5" newSymbol="G5"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="X8_R4" forbiddingContext=""/>
    <OUTPUT type="1" permittingContext="" forbiddingContext="b4_r4_g4_b5_r5_g5"/>
  </FILTERS>
</NODE>

<NODE initCond=""> <!-- NODO 49 ARISTA t=8 G  ( 4 , 5 )  -->
  <EVOLUTIONARY_RULES>
    <RULE ruleType="substitution" actionType="ANY" symbol="r5" newSymbol="R5"/>
    <RULE ruleType="substitution" actionType="ANY" symbol="b5" newSymbol="B5"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="X8_G4" forbiddingContext=""/>
    <OUTPUT type="1" permittingContext="" forbiddingContext="b4_r4_g4_b5_r5_g5"/>
  </FILTERS>
</NODE>
```

```
    <NODE initCond=""> <!-- NODO 50 ARISTA t=8  ( 4 , 5 )  -->
      <EVOLUTIONARY_RULES>
        <RULE ruleType="substitution" actionType="ANY" symbol="R4" newSymbol="r4"/>
        <RULE ruleType="substitution" actionType="ANY" symbol="B4" newSymbol="b4"/>
        <RULE ruleType="substitution" actionType="ANY" symbol="G4" newSymbol="g4"/>
        <RULE ruleType="substitution" actionType="ANY" symbol="R5" newSymbol="r5"/>
        <RULE ruleType="substitution" actionType="ANY" symbol="B5" newSymbol="b5"/>
        <RULE ruleType="substitution" actionType="ANY" symbol="G5" newSymbol="g5"/>
        <RULE ruleType="substitution" actionType="ANY" symbol="X8" newSymbol="X9"/>
      </EVOLUTIONARY_RULES>
      <FILTERS>
        <INPUT type="1" permittingContext="X8" forbiddingContext="r4_b4_g4_r5_b5_g5"/>
        <OUTPUT type="1" permittingContext="X9" forbiddingContext="B1_R1_G1_B2_R2_G2_B3_R3_G3_B4_R4_G4_B5_R5_G5
                                                                  _a1_a2_a3_a4_a5_X1_X2_X3_X4_X5_X6_X8"/>
      </FILTERS>
    </NODE>

  </EVOLUTIONARY_PROCESSORS>

  <STOPPING_CONDITION>
    <CONDITION type="MaximumStepsStoppingCondition" maximum="100"/>
  </STOPPING_CONDITION>
</NEP>
```

Appendix D

jNEP configuration file for the shallow parsing PNEP

Below, a broad summary of the jNEP configuration file for the shallow parsing PNEP explained in section 4.4 is presented. A few highly repetitive elements are ommitted for the sake of simplicity. Some comments are inserted in square brackets. It is worth to note that the Spanish grammar mentioned in some comments is the one used by FreeLing Padró et al. [2010], besides, althouh the grammar is for the Spanish language, it was designed by a Catalan speaker, thus, some terminology used to name the symbols is written in Catalan.

```
<?xml version="1.0"?>
<NEP nodes="192">
<GRAPH>
  <EDGE vertex1="0" vertex2="190"/>
  <EDGE vertex1="0" vertex2="178"/>
  <EDGE vertex1="1" vertex2="190"/>
  <EDGE vertex1="1" vertex2="178"/>
  <EDGE vertex1="2" vertex2="190"/>
  <EDGE vertex1="2" vertex2="178"/>

  [An so on until every non-terminal node is connected to the splicing sub-net and the pruning node.]
  [...]

  [The topology of the splicing sub-net is defined below.]

  <EDGE vertex1="178" vertex2="179"/>
  <EDGE vertex1="178" vertex2="180"/>
  <EDGE vertex1="178" vertex2="181"/>
  <EDGE vertex1="178" vertex2="182"/>
  <EDGE vertex1="178" vertex2="183"/>
  <EDGE vertex1="178" vertex2="184"/>
  <EDGE vertex1="178" vertex2="185"/>
  <EDGE vertex1="178" vertex2="186"/>
  <EDGE vertex1="178" vertex2="187"/>
  <EDGE vertex1="178" vertex2="188"/>
  <EDGE vertex1="179" vertex2="189"/>
  <EDGE vertex1="180" vertex2="189"/>
  <EDGE vertex1="181" vertex2="189"/>
  <EDGE vertex1="182" vertex2="189"/>
  <EDGE vertex1="183" vertex2="189"/>
  <EDGE vertex1="184" vertex2="189"/>
  <EDGE vertex1="185" vertex2="189"/>
  <EDGE vertex1="186" vertex2="189"/>
  <EDGE vertex1="187" vertex2="189"/>
  <EDGE vertex1="188" vertex2="189"/>
  <EDGE vertex1="178" vertex2="191"/>
```

```
  <EDGE vertex1="179" vertex2="191"/>
  <EDGE vertex1="180" vertex2="191"/>
  <EDGE vertex1="181" vertex2="191"/>
  <EDGE vertex1="182" vertex2="191"/>
  <EDGE vertex1="183" vertex2="191"/>
  <EDGE vertex1="184" vertex2="191"/>
  <EDGE vertex1="185" vertex2="191"/>
  <EDGE vertex1="186" vertex2="191"/>
  <EDGE vertex1="187" vertex2="191"/>
  <EDGE vertex1="188" vertex2="191"/>
</GRAPH>

<EVOLUTIONARY_PROCESSORS>
  <NODE initCond="a-ms" id="0">
    <EVOLUTIONARY_RULES>
      <RULE ruleType="leftMostParsing" symbol="a-ms" string="0-0_A0CMS0"
                                       nonTerminals="a-ms_psubj-ms_paton-fp_[...]"/>
      <RULE ruleType="leftMostParsing" symbol="a-ms" string="0-1_AQCMSP"
                                       nonTerminals="[All the grammar non-terminals]"/>
      [And the same for every possible derivation of "a-ms"]
      [...]
    </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="1" permittingContext="a-ms" forbiddingContext=""/>
    </FILTERS>
  </NODE>
  <NODE initCond="psubj-ms" id="1">
    <EVOLUTIONARY_RULES>
      <RULE ruleType="leftMostParsing" symbol="psubj-ms" string="1-0_PP2CS00P"
                                       nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="psubj-ms" string="1-1_PP3NS000"
                                       nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="psubj-ms" string="1-2_PP3MS000"
                                       nonTerminals="[All the grammar non-terminals]"/>
    </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="1" permittingContext="psubj-ms" forbiddingContext=""/>
    </FILTERS>
  </NODE>
  <NODE initCond="paton-fp" id="2">
    <EVOLUTIONARY_RULES>
      <RULE ruleType="leftMostParsing" symbol="paton-fp" string="2-0_PP3FPA00"
                                       nonTerminals="[All the grammar non-terminals]"/>
    </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="1" permittingContext="paton-fp" forbiddingContext=""/>
    </FILTERS>
  </NODE>
  <NODE initCond="grup-complex-spec-ms" id="3">
    <EVOLUTIONARY_RULES>
      <RULE ruleType="leftMostParsing" symbol="grup-complex-spec-ms" string="3-0_pos-ms_indef-ms"
                                       nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="grup-complex-spec-ms" string="3-1_j-ms_indef-ms"
                                       nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="grup-complex-spec-ms" string="3-2_j-ms_indef-ms_indef-ms"
                                       nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="grup-complex-spec-ms" string="3-3_j-ms_num-ms"
                                       nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="grup-complex-spec-ms" string="3-4_pos-ms_num-ms"
                                       nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="grup-complex-spec-ms" string="3-5_dem-ms_num-ms"
                                       nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="grup-complex-spec-ms" string="3-6_indef-ms_pos-ms"
                                       nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="grup-complex-spec-ms" string="3-7_indef-ms_indef-ms"
                                       nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="grup-complex-spec-ms" string="3-8_RG_indef-ms_j-ms"
```

```
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-complex-spec-ms" string="3-9_indef-ms_j-ms"
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-complex-spec-ms" string="3-10_indef-ms_dem-ms"
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-complex-spec-ms" string="3-11_indef-ms_num-ms"
                                    nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="grup-complex-spec-ms" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="paton-fs" id="4">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="paton-fs" string="4-0_PP3FSA00"
                                    nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="paton-fs" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="grup-complex-spec-mp" id="5">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="grup-complex-spec-mp" string="5-0_num-mp_num-mp_num-mp"
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-complex-spec-mp" string="5-1_num-mp_num-mp"
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-complex-spec-mp" string="5-2_pos-mp_indef-mp"
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-complex-spec-mp" string="5-3_j-mp_indef-mp"
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-complex-spec-mp" string="5-4_j-mp_indef-mp_indef-mp"
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-complex-spec-mp" string="5-5_j-mp_num-mp"
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-complex-spec-mp" string="5-6_pos-mp_num-mp"
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-complex-spec-mp" string="5-7_dem-mp_num-mp"
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-complex-spec-mp" string="5-8_indef-mp_pos-mp"
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-complex-spec-mp" string="5-9_indef-mp_indef-mp"
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-complex-spec-mp" string="5-10_RG_indef-mp_j-mp"
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-complex-spec-mp" string="5-11_indef-mp_j-mp"
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-complex-spec-mp" string="5-12_indef-mp_dem-mp"
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-complex-spec-mp" string="5-13_indef-mp_num-mp"
                                    nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="grup-complex-spec-mp" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="psubj-mp" id="6">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="psubj-mp" string="6-0_PP2CP00P"
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="psubj-mp" string="6-1_PP3MP000"
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="psubj-mp" string="6-2_PP2MP000"
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="psubj-mp" string="6-3_PP1MP000"
                                    nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
```

```
  <FILTERS>
    <INPUT type="1" permittingContext="psubj-mp" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="coord" id="7">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="coord" string="7-0_CC"
                                     nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="coord" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="s-adj" id="8">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="s-adj" string="8-0_s-a-fp"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="s-adj" string="8-1_s-a-mp"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="s-adj" string="8-2_s-a-fs"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="s-adj" string="8-3_s-a-ms"
                                     nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="s-adj" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="a-fp" id="9">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="a-fp" string="9-0_AOCFP0"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="a-fp" string="9-1_AQCFPP"
                                     nonTerminals="[All the grammar non-terminals]"/>
    [And the same for every possible derivation of "a-fp"]
    [...]
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="a-fp" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="relatiu" id="10">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="relatiu" string="10-0_prep_cuyo-fp_grup-nom-fp"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="relatiu" string="10-1_prep_cuyo-mp_grup-nom-mp"
                                     nonTerminals="[All the grammar non-terminals]"/>
    [And the same for every possible derivation of "relatiu"]
    [...]
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="relatiu" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="infinitiu" id="11">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="infinitiu" string="11-0_infaux-ser"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="infinitiu" string="11-1_inf-pas"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="infinitiu" string="11-2_inf"
                                     nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="infinitiu" forbiddingContext=""/>
  </FILTERS>
</NODE>
```

```
<NODE initCond="vaux" id="12">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="vaux" string="12-0_VAS*"
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="vaux" string="12-1_VAM*"
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="vaux" string="12-2_VAI*"
                                    nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="vaux" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="parti-mp" id="13">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="parti-mp" string="13-0_VMP00PM"
                                    nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="parti-mp" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="parti-ms" id="14">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="parti-ms" string="14-0_VMP00SM"
                                    nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="parti-ms" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="cual-s" id="15">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="cual-s" string="15-0_PR0CS000"
                                    nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="cual-s" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="a-mp" id="16">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="a-mp" string="16-0_AOCMP0"
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="a-mp" string="16-1_AQCMPP"
                                    nonTerminals="[All the grammar non-terminals]"/>
    [And the same for every possible derivation of "a-mp"]
    [...]
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="a-mp" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="grup-verb-inf" id="17">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="grup-verb-inf" string="17-0_infinitiu"
                                    nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="grup-verb-inf" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="a-fs" id="18">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="a-fs" string="18-0_AOCFS0"
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="a-fs" string="18-1_AQCFSP"
```

```
                                  nonTerminals="[All the grammar non-terminals]"/>
    [And the same for every possible derivation of "a-fs"]
    [...]
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="a-fs" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="cual-p" id="19">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="cual-p" string="19-0_PR0CP000"
                                  nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="cual-p" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="gerundi" id="20">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="gerundi" string="20-0_geraux-ser"
                                  nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="gerundi" string="20-1_ger-pas"
                                  nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="gerundi" string="20-2_ger"
                                  nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="gerundi" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="exc-fs" id="21">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="exc-fs" string="21-0_DE0CN0"
                                  nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="exc-fs" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="espec-ms-E" id="22">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="espec-ms-E" string="22-0_DD0MS0"
                                  nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="espec-ms-E" string="22-1_DI0MS0"
                                  nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="espec-ms-E" string="22-2_DA0MS0"
                                  nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="espec-ms-E" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="grup-sp-inf" id="23">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="grup-sp-inf" string="23-0_prepc-ms_grup-verb-inf"
                                  nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-sp-inf" string="23-1_prep_grup-verb-inf"
                                  nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="grup-sp-inf" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="exc-fp" id="24">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="exc-fp" string="24-0_DE0CN0"
                                  nonTerminals="[All the grammar non-terminals]"/>
```

```
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="exc-fp" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="espec-ms" id="25">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="espec-ms" string="25-0_j-ms"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="espec-ms" string="25-1_indef-ms"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="espec-ms" string="25-2_exc-ms"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="espec-ms" string="25-3_int-ms"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="espec-ms" string="25-4_pos-ms"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="espec-ms" string="25-5_dem-ms"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="espec-ms" string="25-6_num-ms"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="espec-ms" string="25-7_cuantif"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="espec-ms" string="25-8_grup-complex-spec-ms"
                                     nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="espec-ms" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="paton-mp" id="26">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="paton-mp" string="26-0_PP3MPA00"
                                     nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="paton-mp" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="espec-mp" id="27">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="espec-mp" string="27-0_j-mp"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="espec-mp" string="27-1_indef-mp"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="espec-mp" string="27-2_exc-mp"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="espec-mp" string="27-3_int-mp"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="espec-mp" string="27-4_pos-mp"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="espec-mp" string="27-5_dem-mp"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="espec-mp" string="27-6_num-mp"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="espec-mp" string="27-7_cuantif"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="espec-mp" string="27-8_grup-complex-spec-mp"
                                     nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="espec-mp" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="numero" id="28">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="numero" string="28-0_SPS*_numero_CC_numero"
```

```
                                  nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="numero" string="28-1_Zd"
                                  nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="numero" string="28-2_Z"
                                  nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="numero" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="adv-interrog" id="29">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="adv-interrog" string="29-0_PT000000"
                                  nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="adv-interrog" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="verb-pass" id="30">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="verb-pass" string="30-0_vaux_parti-ser_parti-flex"
                                  nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="verb-pass" string="30-1_vser_parti-flex"
                                  nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="verb-pass" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="prep" id="31">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="prep" string="31-0_CS"
                                  nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="prep" string="31-1_SPS00"
                                  nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="prep" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="numero-part" id="32">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="numero-part" string="32-0_Zd"
                                  nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="numero-part" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="paton-ms" id="33">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="paton-ms" string="33-0_PP3MSA00"
                                  nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="paton-ms" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="sn-tmp" id="34">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="sn-tmp" string="34-0_nom-tmp-fp"
                                  nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="sn-tmp" string="34-1_s-a-fp_nom-tmp-fp"
                                  nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="sn-tmp" string="34-2_quant-fp_nom-tmp-fp"
                                  nonTerminals="[All the grammar non-terminals]"/>
```

```
    <RULE ruleType="leftMostParsing" symbol="sn-tmp" string="34-3_quant-fp_s-a-fp_nom-tmp-fp"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="sn-tmp" string="34-4_nom-tmp-mp"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="sn-tmp" string="34-5_s-a-mp_nom-tmp-mp"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="sn-tmp" string="34-6_quant-mp_nom-tmp-mp"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="sn-tmp" string="34-7_quant-mp_s-a-mp_nom-tmp-mp"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="sn-tmp" string="34-8_s-a-fs_nom-tmp-fs"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="sn-tmp" string="34-9_quant-fs_nom-tmp-fs"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="sn-tmp" string="34-10_quant-fs_s-a-fs_nom-tmp-fs"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="sn-tmp" string="34-11_nom-tmp-ms"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="sn-tmp" string="34-12_s-a-ms_nom-tmp-ms"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="sn-tmp" string="34-13_quant-ms_nom-tmp-ms"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="sn-tmp" string="34-14_quant-ms_s-a-ms_nom-tmp-ms"
                                     nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="sn-tmp" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="prel" id="35">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="prel" string="35-0_PR0CN000"
                                     nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="prel" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="grup-complex-spec-fp" id="36">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="grup-complex-spec-fp" string="36-0_num-fp_num-fp_num-fp"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-complex-spec-fp" string="36-1_pos-fp_indef-fp"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-complex-spec-fp" string="36-2_j-fp_indef-fp"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-complex-spec-fp" string="36-3_j-fp_indef-fp_indef-fp"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-complex-spec-fp" string="36-4_j-fp_num-fp"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-complex-spec-fp" string="36-5_pos-fp_num-fp"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-complex-spec-fp" string="36-6_dem-fp_num-fp"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-complex-spec-fp" string="36-7_indef-fp_pos-fp"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-complex-spec-fp" string="36-8_indef-fp_indef-fp"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-complex-spec-fp" string="36-9_RG_indef-fp_j-fp"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-complex-spec-fp" string="36-10_indef-fp_j-fp"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-complex-spec-fp" string="36-11_indef-fp_dem-fp"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-complex-spec-fp" string="36-12_num-fp_num-fp"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-complex-spec-fp" string="36-13_indef-fp_num-fp"
```

```
                                    nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="grup-complex-spec-fp" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="grup-complex-spec-fs" id="37">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="grup-complex-spec-fs" string="37-0_pos-fs_indef-fs"
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-complex-spec-fs" string="37-1_j-fs_indef-fs"
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-complex-spec-fs" string="37-2_j-fs_indef-fs_indef-fs"
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-complex-spec-fs" string="37-3_j-fs_num-fs"
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-complex-spec-fs" string="37-4_pos-fs_num-fs"
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-complex-spec-fs" string="37-5_dem-fs_num-fs"
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-complex-spec-fs" string="37-6_indef-fs_pos-fs"
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-complex-spec-fs" string="37-7_indef-fs_indef-fs"
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-complex-spec-fs" string="37-8_RG_indef-fs_j-fs"
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-complex-spec-fs" string="37-9_indef-fs_j-fs"
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-complex-spec-fs" string="37-10_indef-fs_dem-fs"
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-complex-spec-fs" string="37-11_num-fs_num-fs"
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-complex-spec-fs" string="37-12_indef-fs_num-fs"
                                    nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="grup-complex-spec-fs" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="exc-ms" id="38">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="exc-ms" string="38-0_DE0CN0"
                                    nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="exc-ms" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="parti" id="39">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="parti" string="39-0_VMP00SM"
                                    nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="parti" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="exc-mp" id="40">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="exc-mp" string="40-0_DE0CN0"
                                    nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="exc-mp" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="neg" id="41">
```

```
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="neg" string="41-0_RN"
                                     nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="neg" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="prepc-ms" id="42">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="prepc-ms" string="42-0_SPCMS"
                                     nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="prepc-ms" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="F-term" id="43">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="F-term" string="43-0_Fat"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="F-term" string="43-1_Fit"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="F-term" string="43-2_Fp"
                                     nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="F-term" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="grup-sp" id="44">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="grup-sp" string="44-0_SPS00_sn_CC_sn"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-sp" string="44-1_prepc-ms_W"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-sp" string="44-2_prepc-ms_s-a-ms"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-sp" string="44-3_prep_ptonic"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-sp" string="44-4_prep_sn"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-sp" string="44-5_prep_sadv"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-sp" string="44-6_prep_s-a-fp"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-sp" string="44-7_prep_s-a-fs"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-sp" string="44-8_prep_s-a-mp"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-sp" string="44-9_prep_s-a-ms"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-sp" string="44-10_prep_data"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-sp" string="44-11_prep_numero"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-sp" string="44-12_prepc-ms_pposs-ms"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-sp" string="44-13_prepc-ms_pindef-ms"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-sp" string="44-14_prepc-ms_grup-nom-ms"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-sp" string="44-15_PP3CS000"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-sp" string="44-16_PP2CS000"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-sp" string="44-17_PP1CS000"
```

```
                                    nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="grup-sp" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="ptonic" id="45">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="ptonic" string="45-0_PP3CN000"
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="ptonic" string="45-1_PP2CS000"
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="ptonic" string="45-2_PP1CS000"
                                    nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="ptonic" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="j-ms" id="46">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="j-ms" string="46-0_DA0NS0"
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="j-ms" string="46-1_DA0MS0"
                                    nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="j-ms" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="forma-inf" id="47">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="forma-inf" string="47-0_VMN0000"
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="forma-inf" string="47-1_VSN0000"
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="forma-inf" string="47-2_VAN0000"
                                    nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="forma-inf" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="data" id="48">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="data" string="48-0_data_coord_data"
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="data" string="48-1_espec-ms_W"
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="data" string="48-2_W"
                                    nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="data" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="pinterrog-fp" id="49">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="pinterrog-fp" string="49-0_PT0FP000"
                                    nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="pinterrog-fp" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="j-mp" id="50">
  <EVOLUTIONARY_RULES>
```

```
    <RULE ruleType="leftMostParsing" symbol="j-mp" string="50-0_DA0MP0"
                                     nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="j-mp" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="psubj-fs" id="51">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="psubj-fs" string="51-0_PP2CS00P"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="psubj-fs" string="51-1_PP3FS000"
                                     nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="psubj-fs" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="parti-ser" id="52">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="parti-ser" string="52-0_VSP00SM"
                                     nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="parti-ser" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="pinterrog-fs" id="53">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="pinterrog-fs" string="53-0_PT0FS000"
                                     nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="pinterrog-fs" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="num-mp" id="54">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="num-mp" string="54-0_Z"
                                     nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="num-mp" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="num-ms" id="55">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="num-ms" string="55-0_Z"
                                     nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="num-ms" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="numero-nopart" id="56">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="numero-nopart" string="56-0_Zd_SPS00"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="numero-nopart" string="56-1_Z"
                                     nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="numero-nopart" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="n-ms" id="57">
  <EVOLUTIONARY_RULES>
```

```
    <RULE ruleType="leftMostParsing" symbol="n-ms" string="57-0_NCCN00*"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="n-ms" string="57-1_NCMN00*"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="n-ms" string="57-2_NCCS00*"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="n-ms" string="57-3_NCMS00*"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="n-ms" string="57-4_NC0000*"
                                     nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="n-ms" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="n-mp" id="58">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="n-mp" string="58-0_NCCN00*"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="n-mp" string="58-1_NCMN00*"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="n-mp" string="58-2_NCCP00*"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="n-mp" string="58-3_NCMP00*"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="n-mp" string="58-4_NC0000*"
                                     nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="n-mp" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="pron" id="59">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="pron" string="59-0_pinterrog"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="pron" string="59-1_psubj-s"
                                     nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="pron" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="quant-fs" id="60">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="quant-fs" string="60-0_indef-fs"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="quant-fs" string="60-1_Zd_SPS00"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="quant-fs" string="60-2_Z"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="quant-fs" string="60-3_num-fs"
                                     nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="quant-fs" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="quant-fp" id="61">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="quant-fp" string="61-0_indef-fp"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="quant-fp" string="61-1_Zd_SPS00"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="quant-fp" string="61-2_Z"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="quant-fp" string="61-3_num-fp"
```

```
                                  nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="quant-fp" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="psubj-fp" id="62">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="psubj-fp" string="62-0_PP2CP00P"
                                  nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="psubj-fp" string="62-1_PP3FP000"
                                  nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="psubj-fp" string="62-2_PP2FP000"
                                  nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="psubj-fp" string="62-3_PP1FP000"
                                  nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="psubj-fp" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="num-fp" id="63">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="num-fp" string="63-0_Z"
                                  nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="num-fp" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="pinterrog-s" id="64">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="pinterrog-s" string="64-0_PT0CS000"
                                  nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="pinterrog-s" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="nom-fs-E" id="65">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="nom-fs-E" string="65-0_NCFS00*"
                                  nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="nom-fs-E" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="num-fs" id="66">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="num-fs" string="66-0_Z"
                                  nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="num-fs" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="pinterrog-p" id="67">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="pinterrog-p" string="67-0_PT0CP000"
                                  nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="pinterrog-p" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="pinterrog" id="68">
```

```
    <EVOLUTIONARY_RULES>
      <RULE ruleType="leftMostParsing" symbol="pinterrog" string="68-0_PT0CN000"
                                       nonTerminals="[All the grammar non-terminals]"/>
    </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="1" permittingContext="pinterrog" forbiddingContext=""/>
    </FILTERS>
  </NODE>
  <NODE initCond="pdem-mp" id="69">
    <EVOLUTIONARY_RULES>
      <RULE ruleType="leftMostParsing" symbol="pdem-mp" string="69-0_PD0CP000"
                                       nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="pdem-mp" string="69-1_PD0MP000"
                                       nonTerminals="[All the grammar non-terminals]"/>
    </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="1" permittingContext="pdem-mp" forbiddingContext=""/>
    </FILTERS>
  </NODE>
  <NODE initCond="grup-verb" id="70">
    <EVOLUTIONARY_RULES>
      <RULE ruleType="leftMostParsing" symbol="grup-verb" string="70-0_grup-verb_patons_patons_patons"
                                       nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="grup-verb" string="70-1_grup-verb_patons_patons"
                                       nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="grup-verb" string="70-2_grup-verb_patons"
                                       nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="grup-verb" string="70-3_patons_grup-verb"
                                       nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="grup-verb" string="70-4_morfema-verbal_patons_grup-verb"
                                       nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="grup-verb" string="70-5_morfema-verbal_grup-verb"
                                       nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="grup-verb" string="70-6_verb-pass"
                                       nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="grup-verb" string="70-7_verb"
                                       nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="grup-verb" string="70-8_morf-pron_grup-verb"
                                       nonTerminals="[All the grammar non-terminals]"/>
    </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="1" permittingContext="grup-verb" forbiddingContext=""/>
    </FILTERS>
  </NODE>
  <NODE initCond="pdem-ms" id="71">
    <EVOLUTIONARY_RULES>
      <RULE ruleType="leftMostParsing" symbol="pdem-ms" string="71-0_PD0CS000"
                                       nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="pdem-ms" string="71-1_PD0NS000"
                                       nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="pdem-ms" string="71-2_PD0MS000"
                                       nonTerminals="[All the grammar non-terminals]"/>
    </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="1" permittingContext="pdem-ms" forbiddingContext=""/>
    </FILTERS>
  </NODE>
  <NODE initCond="grup-nom-mp" id="72">
    <EVOLUTIONARY_RULES>
      <RULE ruleType="leftMostParsing" symbol="grup-nom-mp" string="72-0_espec-mp_parti-mp"
                                       nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="grup-nom-mp" string="72-1_n-mp_pdem-mp"
                                       nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="grup-nom-mp" string="72-2_n-mp_pos-mp"
                                       nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="grup-nom-mp" string="72-3_pnum-mp_pnum-mp_pnum-mp"
                                       nonTerminals="[All the grammar non-terminals]"/>
```

```
      <RULE ruleType="leftMostParsing" symbol="grup-nom-mp" string="72-4_pnum-mp_pnum-mp"
                                       nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="grup-nom-mp" string="72-5_grup-c-nom-mp_s-a-mp"
                                       nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="grup-nom-mp" string="72-6_grup-c-nom-mp"
                                       nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="grup-nom-mp" string="72-7_s-a-mp_grup-nom-mp"
                                       nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="grup-nom-mp" string="72-8_w-mp_s-a-mp"
                                       nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="grup-nom-mp" string="72-9_n-mp_s-a-mp"
                                       nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="grup-nom-mp" string="72-10_w-mp"
                                       nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="grup-nom-mp" string="72-11_n-mp_n-fp"
                                       nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="grup-nom-mp" string="72-12_n-mp_n-mp"
                                       nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="grup-nom-mp" string="72-13_n-mp_n-ms"
                                       nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="grup-nom-mp" string="72-14_n-mp_n-fs"
                                       nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="grup-nom-mp" string="72-15_n-mp"
                                       nonTerminals="[All the grammar non-terminals]"/>
    </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="1" permittingContext="grup-nom-mp" forbiddingContext=""/>
    </FILTERS>
  </NODE>
  <NODE initCond="sadv" id="73">
    <EVOLUTIONARY_RULES>
      <RULE ruleType="leftMostParsing" symbol="sadv" string="73-0_RG_SPS00_sadv"
                                       nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="sadv" string="73-1_SPCMS_RG_sn"
                                       nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="sadv" string="73-2_RG_SPS00_sn"
                                       nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="sadv" string="73-3_cuantif_sadv"
                                       nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="sadv" string="73-4_RG"
                                       nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="sadv" string="73-5_adv-interrog"
                                       nonTerminals="[All the grammar non-terminals]"/>
    </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="1" permittingContext="sadv" forbiddingContext=""/>
    </FILTERS>
  </NODE>
  <NODE initCond="pron-ns" id="74">
    <EVOLUTIONARY_RULES>
      <RULE ruleType="leftMostParsing" symbol="pron-ns" string="74-0_pposs-ns"
                                       nonTerminals="[All the grammar non-terminals]"/>
    </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="1" permittingContext="pron-ns" forbiddingContext=""/>
    </FILTERS>
  </NODE>
  <NODE initCond="pindef-fp" id="75">
    <EVOLUTIONARY_RULES>
      <RULE ruleType="leftMostParsing" symbol="pindef-fp" string="75-0_PI0CP000"
                                       nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="pindef-fp" string="75-1_PI0FP000"
                                       nonTerminals="[All the grammar non-terminals]"/>
    </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="1" permittingContext="pindef-fp" forbiddingContext=""/>
    </FILTERS>
```

```
</NODE>
<NODE initCond="pposs-fp" id="76">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="pposs-fp" string="76-0_DA0FP0_PX3FP0C0"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="pposs-fp" string="76-1_DA0FP0_PX2FP0P0"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="pposs-fp" string="76-2_DA0FP0_PX2FP0S0"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="pposs-fp" string="76-3_DA0FP0_PX1FP0P0"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="pposs-fp" string="76-4_DA0FP0_PX1FP0S0"
                                     nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="pposs-fp" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="grup-nom-ms" id="77">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="grup-nom-ms" string="77-0_espec-ms_parti-ms"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-nom-ms" string="77-1_n-ms_pdem-ms"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-nom-ms" string="77-2_n-ms_pos-ms"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-nom-ms" string="77-3_s-a-ms_grup-nom-ms"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-nom-ms" string="77-4_w-ms_s-a-ms"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-nom-ms" string="77-5_n-ms_s-a-ms"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-nom-ms" string="77-6_n-ms_w-ms"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-nom-ms" string="77-7_w-ms_w-ms"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-nom-ms" string="77-8_w-ms"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-nom-ms" string="77-9_n-ms_n-fp"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-nom-ms" string="77-10_n-ms_n-mp"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-nom-ms" string="77-11_n-ms_n-ms"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-nom-ms" string="77-12_n-ms_n-fs"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-nom-ms" string="77-13_n-ms"
                                     nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="grup-nom-ms" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="pposs-fs" id="78">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="pposs-fs" string="78-0_DA0FS0_PX3FS0C0"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="pposs-fs" string="78-1_DA0FS0_PX2FS0P0"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="pposs-fs" string="78-2_DA0FS0_PX2FS0S0"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="pposs-fs" string="78-3_DA0FS0_PX1FS0P0"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="pposs-fs" string="78-4_DA0FS0_PX1FS0S0"
                                     nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
```

```
    <INPUT type="1" permittingContext="pposs-fs" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="cuyo-fs" id="79">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="cuyo-fs" string="79-0_PR0FS000"
                                     nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="cuyo-fs" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="s-a-fs" id="80">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="s-a-fs" string="80-0_parti-fs"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="s-a-fs" string="80-1_a-fs_s-a-fs"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="s-a-fs" string="80-2_sadv_a-fs"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="s-a-fs" string="80-3_s-a-fs_coord_s-a-fs"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="s-a-fs" string="80-4_s-a-fs_Fc_s-a-fs_Fs"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="s-a-fs" string="80-5_s-a-fs_Fc_s-a-fs"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="s-a-fs" string="80-6_a-fs"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="s-a-fs" string="80-7_Fpa_s-a-fs_Fpt"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="s-a-fs" string="80-8_Fe_s-a-fs_Fe"
                                     nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="s-a-fs" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="cuyo-mp" id="81">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="cuyo-mp" string="81-0_PR0MP000"
                                     nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="cuyo-mp" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="indef-mp" id="82">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="indef-mp" string="82-0_DI0CP0"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="indef-mp" string="82-1_DI0MP0"
                                     nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="indef-mp" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="indef-ms" id="83">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="indef-ms" string="83-0_DI0CS0"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="indef-ms" string="83-1_DI0MS0"
                                     nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="indef-ms" forbiddingContext=""/>
  </FILTERS>
```

```
</NODE>
<NODE initCond="pdem-fs" id="84">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="pdem-fs" string="84-0_PD0CS000"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="pdem-fs" string="84-1_PD0FS000"
                                     nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="pdem-fs" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="s-a-fp" id="85">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="s-a-fp" string="85-0_parti-fp"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="s-a-fp" string="85-1_a-fp_s-a-fp"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="s-a-fp" string="85-2_sadv_a-fp"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="s-a-fp" string="85-3_s-a-fp_coord_s-a-fp"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="s-a-fp" string="85-4_s-a-fp_Fc_s-a-fp_Fs"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="s-a-fp" string="85-5_s-a-fp_Fc_s-a-fp"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="s-a-fp" string="85-6_a-fp"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="s-a-fp" string="85-7_Fpa_s-a-fp_Fpt"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="s-a-fp" string="85-8_Fe_s-a-fp_Fe"
                                     nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="s-a-fp" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="cuyo-ms" id="86">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="cuyo-ms" string="86-0_PR0MS000"
                                     nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="cuyo-ms" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="pindef-fs" id="87">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="pindef-fs" string="87-0_PI0FS000"
                                     nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="pindef-fs" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="pdem-fp" id="88">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="pdem-fp" string="88-0_PD0CP000"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="pdem-fp" string="88-1_PD0FP000"
                                     nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="pdem-fp" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="geraux-ser" id="89">
```

```
    <EVOLUTIONARY_RULES>
      <RULE ruleType="leftMostParsing" symbol="geraux-ser" string="89-0_VSG0000_PP*_PP*"
                                       nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="geraux-ser" string="89-1_VSG0000_PP*"
                                       nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="geraux-ser" string="89-2_VSG0000"
                                       nonTerminals="[All the grammar non-terminals]"/>
    </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="1" permittingContext="geraux-ser" forbiddingContext=""/>
    </FILTERS>
  </NODE>
  <NODE initCond="w-mp" id="90">
    <EVOLUTIONARY_RULES>
      <RULE ruleType="leftMostParsing" symbol="w-mp" string="90-0_NP*"
                                       nonTerminals="[All the grammar non-terminals]"/>
    </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="1" permittingContext="w-mp" forbiddingContext=""/>
    </FILTERS>
  </NODE>
  <NODE initCond="pindef-mp" id="91">
    <EVOLUTIONARY_RULES>
      <RULE ruleType="leftMostParsing" symbol="pindef-mp" string="91-0_PI0CP000"
                                       nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="pindef-mp" string="91-1_PI0MP000"
                                       nonTerminals="[All the grammar non-terminals]"/>
    </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="1" permittingContext="pindef-mp" forbiddingContext=""/>
    </FILTERS>
  </NODE>
  <NODE initCond="n-fp" id="92">
    <EVOLUTIONARY_RULES>
      <RULE ruleType="leftMostParsing" symbol="n-fp" string="92-0_NCCN00*"
                                       nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="n-fp" string="92-1_NCFN00*"
                                       nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="n-fp" string="92-2_NCCP00*"
                                       nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="n-fp" string="92-3_NCFP00*"
                                       nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="n-fp" string="92-4_NC0000*"
                                       nonTerminals="[All the grammar non-terminals]"/>
    </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="1" permittingContext="n-fp" forbiddingContext=""/>
    </FILTERS>
  </NODE>
  <NODE initCond="dem-ms" id="93">
    <EVOLUTIONARY_RULES>
      <RULE ruleType="leftMostParsing" symbol="dem-ms" string="93-0_DD0CS0"
                                       nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="dem-ms" string="93-1_DD0MS0"
                                       nonTerminals="[All the grammar non-terminals]"/>
    </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="1" permittingContext="dem-ms" forbiddingContext=""/>
    </FILTERS>
  </NODE>
  <NODE initCond="dem-mp" id="94">
    <EVOLUTIONARY_RULES>
      <RULE ruleType="leftMostParsing" symbol="dem-mp" string="94-0_DD0CP0"
                                       nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="dem-mp" string="94-1_DD0MP0"
                                       nonTerminals="[All the grammar non-terminals]"/>
    </EVOLUTIONARY_RULES>
```

```
  <FILTERS>
    <INPUT type="1" permittingContext="dem-mp" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="nom-tmp-mp" id="95">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="nom-tmp-mp" string="95-0_NCMP*"
                                     nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="nom-tmp-mp" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="F-no-c" id="96">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="F-no-c" string="96-0_Fz"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="F-no-c" string="96-1_Fx"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="F-no-c" string="96-2_Ft"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="F-no-c" string="96-3_Fs"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="F-no-c" string="96-4_Frc"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="F-no-c" string="96-5_Fra"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="F-no-c" string="96-6_Fpt"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="F-no-c" string="96-7_Fpa"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="F-no-c" string="96-8_Flt"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="F-no-c" string="96-9_Fla"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="F-no-c" string="96-10_Fia"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="F-no-c" string="96-11_Fh"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="F-no-c" string="96-12_Fg"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="F-no-c" string="96-13_Fe"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="F-no-c" string="96-14_Fd"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="F-no-c" string="96-15_Fct"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="F-no-c" string="96-16_Fca"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="F-no-c" string="96-17_Faa"
                                     nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="F-no-c" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="pindef-ms" id="97">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="pindef-ms" string="97-0_PI0CS000"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="pindef-ms" string="97-1_PI0MS000"
                                     nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="pindef-ms" forbiddingContext=""/>
  </FILTERS>
</NODE>
```

```
<NODE initCond="forma-ger" id="98">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="forma-ger" string="98-0_VAG0000"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="forma-ger" string="98-1_VMG0000"
                                     nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="forma-ger" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="nom-tmp-ms" id="99">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="nom-tmp-ms" string="99-0_NCMS*"
                                     nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="nom-tmp-ms" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="w-ms" id="100">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="w-ms" string="100-0_NP*"
                                     nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="w-ms" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="n-fs" id="101">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="n-fs" string="101-0_NCCN00*"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="n-fs" string="101-1_NCFN00*"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="n-fs" string="101-2_NCCS00*"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="n-fs" string="101-3_NCFS00*"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="n-fs" string="101-4_NC0000*"
                                     nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="n-fs" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="pron-fp" id="102">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="pron-fp" string="102-0_pindef-fp"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="pron-fp" string="102-1_pposs-fp"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="pron-fp" string="102-2_pinterrog-fp"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="pron-fp" string="102-3_pdem-fp"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="pron-fp" string="102-4_psubj-fp"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="pron-fp" string="102-5_pinterrog-p"
                                     nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="pron-fp" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="pron-fs" id="103">
  <EVOLUTIONARY_RULES>
```

```
    <RULE ruleType="leftMostParsing" symbol="pron-fs" string="103-0_pindef-fs"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="pron-fs" string="103-1_pposs-fs"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="pron-fs" string="103-2_pinterrog-fs"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="pron-fs" string="103-3_pdem-fs"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="pron-fs" string="103-4_psubj-fs"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="pron-fs" string="103-5_pinterrog"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="pron-fs" string="103-6_pinterrog-s"
                                     nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="pron-fs" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="psubj-s" id="104">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="psubj-s" string="104-0_PP2CSN00"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="psubj-s" string="104-1_PP1CSN00"
                                     nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="psubj-s" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="pos-fs" id="105">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="pos-fs" string="105-0_DP3FP0"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="pos-fs" string="105-1_DP3FS0"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="pos-fs" string="105-2_DP2FSP"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="pos-fs" string="105-3_DP1FSP"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="pos-fs" string="105-4_DP3CS0"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="pos-fs" string="105-5_DP2CSS"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="pos-fs" string="105-6_DP1CSS"
                                     nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="pos-fs" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="cuyo-fp" id="106">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="cuyo-fp" string="106-0_PR0FP000"
                                     nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="cuyo-fp" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="geraux" id="107">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="geraux" string="107-0_VAG0000_PP*_PP*"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="geraux" string="107-1_VAG0000_PP*"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="geraux" string="107-2_VAG0000"
```

```
                                    nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="geraux" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="pposs-ns" id="108">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="pposs-ns" string="108-0_DA0NS0_PX3NS0C0"
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="pposs-ns" string="108-1_DA0NS0_PX2NS0P0"
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="pposs-ns" string="108-2_DA0NS0_PX2NS0S0"
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="pposs-ns" string="108-3_DA0NS0_PX1NS0P0"
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="pposs-ns" string="108-4_DA0NS0_PX1NS0S0"
                                    nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="pposs-ns" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="pos-fp" id="109">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="pos-fp" string="109-0_DP2FPP"
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="pos-fp" string="109-1_DP1FPP"
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="pos-fp" string="109-2_DP3CP0"
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="pos-fp" string="109-3_DP2CPS"
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="pos-fp" string="109-4_DP1CPS"
                                    nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="pos-fp" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="infaux-ser" id="110">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="infaux-ser" string="110-0_VSN0000"
                                    nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="infaux-ser" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="pron-ms" id="111">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="pron-ms" string="111-0_pindef-ms"
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="pron-ms" string="111-1_pposs-ms"
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="pron-ms" string="111-2_pinterrog-ms"
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="pron-ms" string="111-3_pdem-ms"
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="pron-ms" string="111-4_psubj-ms"
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="pron-ms" string="111-5_pinterrog"
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="pron-ms" string="111-6_pinterrog-s"
                                    nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
```

```
    <INPUT type="1" permittingContext="pron-ms" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="sn" id="112">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="sn" string="112-0_CC_sn_CC_sn"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="sn" string="112-1_pdem-fp_s-a-fp"
                                     nonTerminals="[All the grammar non-terminals]"/>
    [And the same for every possible derivation of "sn"]
    [...]
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="sn" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="parti-flex" id="113">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="parti-flex" string="113-0_parti-fp"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="parti-flex" string="113-1_parti-mp"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="parti-flex" string="113-2_parti-fs"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="parti-flex" string="113-3_parti-ms"
                                     nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="parti-flex" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="grup-nom" id="114">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="grup-nom" string="114-0_Zu"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-nom" string="114-1_Zp"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-nom" string="114-2_Zm"
                                     nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="grup-nom" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="ger" id="115">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="ger" string="115-0_ger_PP*_PP*"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="ger" string="115-1_ger_PP*"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="ger" string="115-2_VMG0000_gerundi"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="ger" string="115-3_VAG0000_SPS00_infinitiu"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="ger" string="115-4_VMG0000_CS_infinitiu"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="ger" string="115-5_VMG0000_SPS00_infinitiu"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="ger" string="115-6_VMG0000_infinitiu"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="ger" string="115-7_geraux_parti-ser"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="ger" string="115-8_geraux_parti"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="ger" string="115-9_VAG0000_CS_infinitiu"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="ger" string="115-10_forma-ger_PP*_PP*"
```

```
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="ger" string="115-11_forma-ger_PP*"
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="ger" string="115-12_forma-ger"
                                    nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="ger" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="inf" id="116">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="inf" string="116-0_infaux_VMP00SM_gerundi"
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="inf" string="116-1_VMN0000_gerundi"
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="inf" string="116-2_infaux_VMP00SM_CS_infinitiu"
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="inf" string="116-3_infaux_VMP00SM_SPS00_infinitiu"
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="inf" string="116-4_infaux_VMP00SM_infinitiu"
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="inf" string="116-5_VAN0000_SPS00_infinitiu"
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="inf" string="116-6_VMN0000_CS_infinitiu"
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="inf" string="116-7_VMN0000_SPS00_infinitiu"
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="inf" string="116-8_VMN0000_infinitiu"
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="inf" string="116-9_infaux_parti-ser"
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="inf" string="116-10_infaux_parti"
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="inf" string="116-11_VAN0000_CS_infinitiu"
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="inf" string="116-12_forma-inf_PP*_PP*"
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="inf" string="116-13_forma-inf_PP*"
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="inf" string="116-14_forma-inf"
                                    nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="inf" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="dem-fs" id="117">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="dem-fs" string="117-0_DD0CS0"
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="dem-fs" string="117-1_DD0FS0"
                                    nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="dem-fs" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="parti-aux" id="118">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="parti-aux" string="118-0_VAP00SM"
                                    nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="parti-aux" forbiddingContext=""/>
  </FILTERS>
</NODE>
```

```
<NODE initCond="int-ms" id="119">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="int-ms" string="119-0_DT0CN0"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="int-ms" string="119-1_DT0MS0"
                                     nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="int-ms" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="pinterrog-ms" id="120">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="pinterrog-ms" string="120-0_PT0MS000"
                                     nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="pinterrog-ms" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="vser" id="121">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="vser" string="121-0_VSS*"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="vser" string="121-1_VSM*"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="vser" string="121-2_VSI*"
                                     nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="vser" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="int-mp" id="122">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="int-mp" string="122-0_DT0CN0"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="int-mp" string="122-1_DT0MP0"
                                     nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="int-mp" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="j-fp" id="123">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="j-fp" string="123-0_DA0FP0"
                                     nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="j-fp" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="pinterrog-mp" id="124">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="pinterrog-mp" string="124-0_PT0MP000"
                                     nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="pinterrog-mp" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="dem-fp" id="125">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="dem-fp" string="125-0_DD0CP0"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="dem-fp" string="125-1_DD0FP0"
```

```
                                     nonTerminals="[All the grammar non-terminals]"/>
    </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="1" permittingContext="dem-fp" forbiddingContext=""/>
    </FILTERS>
  </NODE>
  <NODE initCond="quant-mp" id="126">
    <EVOLUTIONARY_RULES>
      <RULE ruleType="leftMostParsing" symbol="quant-mp" string="126-0_indef-mp"
                                     nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="quant-mp" string="126-1_Zd_SPS00"
                                     nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="quant-mp" string="126-2_Z"
                                     nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="quant-mp" string="126-3_num-mp"
                                     nonTerminals="[All the grammar non-terminals]"/>
    </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="1" permittingContext="quant-mp" forbiddingContext=""/>
    </FILTERS>
  </NODE>
  <NODE initCond="j-fs" id="127">
    <EVOLUTIONARY_RULES>
      <RULE ruleType="leftMostParsing" symbol="j-fs" string="127-0_DA0FS0"
                                     nonTerminals="[All the grammar non-terminals]"/>
    </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="1" permittingContext="j-fs" forbiddingContext=""/>
    </FILTERS>
  </NODE>
  <NODE initCond="pnum-fs" id="128">
    <EVOLUTIONARY_RULES>
      <RULE ruleType="leftMostParsing" symbol="pnum-fs" string="128-0_Z"
                                     nonTerminals="[All the grammar non-terminals]"/>
    </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="1" permittingContext="pnum-fs" forbiddingContext=""/>
    </FILTERS>
  </NODE>
  <NODE initCond="infaux" id="129">
    <EVOLUTIONARY_RULES>
      <RULE ruleType="leftMostParsing" symbol="infaux" string="129-0_VAN0000"
                                     nonTerminals="[All the grammar non-terminals]"/>
    </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="1" permittingContext="infaux" forbiddingContext=""/>
    </FILTERS>
  </NODE>
  <NODE initCond="nom-tmp-fp" id="130">
    <EVOLUTIONARY_RULES>
      <RULE ruleType="leftMostParsing" symbol="nom-tmp-fp" string="130-0_NCFP*"
                                     nonTerminals="[All the grammar non-terminals]"/>
    </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="1" permittingContext="nom-tmp-fp" forbiddingContext=""/>
    </FILTERS>
  </NODE>
  <NODE initCond="indef-fs" id="131">
    <EVOLUTIONARY_RULES>
      <RULE ruleType="leftMostParsing" symbol="indef-fs" string="131-0_DI0CS0"
                                     nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="indef-fs" string="131-1_DI0FS0"
                                     nonTerminals="[All the grammar non-terminals]"/>
    </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="1" permittingContext="indef-fs" forbiddingContext=""/>
    </FILTERS>
```

```
</NODE>
<NODE initCond="quant-ms" id="132">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="quant-ms" string="132-0_indef-ms"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="quant-ms" string="132-1_Zd_SPS00"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="quant-ms" string="132-2_Z"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="quant-ms" string="132-3_num-ms"
                                     nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="quant-ms" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="pnum-fp" id="133">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="pnum-fp" string="133-0_Z"
                                     nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="pnum-fp" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="inf-pas" id="134">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="inf-pas" string="134-0_infaux_parti-ser_parti-flex"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="inf-pas" string="134-1_infaux-ser_parti-flex"
                                     nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="inf-pas" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="nom-tmp-fs" id="135">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="nom-tmp-fs" string="135-0_NCFS*"
                                     nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="nom-tmp-fs" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="prel-adv" id="136">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="prel-adv" string="136-0_PR000000"
                                     nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="prel-adv" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="pron-mp" id="137">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="pron-mp" string="137-0_pindef-mp"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="pron-mp" string="137-1_pposs-mp"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="pron-mp" string="137-2_pinterrog-mp"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="pron-mp" string="137-3_pdem-mp"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="pron-mp" string="137-4_psubj-mp"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="pron-mp" string="137-5_pinterrog-p"
```

```
                                     nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="pron-mp" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="indef-fp" id="138">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="indef-fp" string="138-0_DI0CP0"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="indef-fp" string="138-1_DI0FP0"
                                     nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="indef-fp" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="sp-de" id="139">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="sp-de" string="139-0_SPCMS_grup-nom-ms"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="sp-de" string="139-1_SPS00_sn"
                                     nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="sp-de" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="quien-s" id="140">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="quien-s" string="140-0_PR0CS000"
                                     nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="quien-s" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="ger-pas" id="141">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="ger-pas" string="141-0_geraux_parti-aux_parti-flex"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="ger-pas" string="141-1_geraux-ser_parti-flex"
                                     nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="ger-pas" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="paton-s" id="142">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="paton-s" string="142-0_PP3CSA00"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="paton-s" string="142-1_PP3CSD00"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="paton-s" string="142-2_PP2CS000"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="paton-s" string="142-3_PP1CS000"
                                     nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="paton-s" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="quien-p" id="143">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="quien-p" string="143-0_PR0CP000"
                                     nonTerminals="[All the grammar non-terminals]"/>
```

```
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="quien-p" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="espec-fp" id="144">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="espec-fp" string="144-0_j-fp"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="espec-fp" string="144-1_indef-fp"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="espec-fp" string="144-2_exc-fp"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="espec-fp" string="144-3_int-fp"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="espec-fp" string="144-4_pos-fp"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="espec-fp" string="144-5_dem-fp"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="espec-fp" string="144-6_num-fp"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="espec-fp" string="144-7_cuantif"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="espec-fp" string="144-8_grup-complex-spec-fp"
                                     nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="espec-fp" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="cuantif" id="145">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="cuantif" string="145-0_RG"
                                     nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="cuantif" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="prel-fs" id="146">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="prel-fs" string="146-0_PR0FS000"
                                     nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="prel-fs" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="parti-fs" id="147">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="parti-fs" string="147-0_VMP00SF"
                                     nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="parti-fs" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="paton-p" id="148">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="paton-p" string="148-0_PP3CPA00"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="paton-p" string="148-1_PP3CPD00"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="paton-p" string="148-2_PP2CP000"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="paton-p" string="148-3_PP1CP000"
                                     nonTerminals="[All the grammar non-terminals]"/>
```

```
    </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="1" permittingContext="paton-p" forbiddingContext=""/>
    </FILTERS>
  </NODE>
  <NODE initCond="espec-fs" id="149">
    <EVOLUTIONARY_RULES>
      <RULE ruleType="leftMostParsing" symbol="espec-fs" string="149-0_j-fs"
                                       nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="espec-fs" string="149-1_indef-fs"
                                       nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="espec-fs" string="149-2_exc-fs"
                                       nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="espec-fs" string="149-3_int-fs"
                                       nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="espec-fs" string="149-4_pos-fs"
                                       nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="espec-fs" string="149-5_dem-fs"
                                       nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="espec-fs" string="149-6_num-fs"
                                       nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="espec-fs" string="149-7_cuantif"
                                       nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="espec-fs" string="149-8_grup-complex-spec-fs"
                                       nonTerminals="[All the grammar non-terminals]"/>
    </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="1" permittingContext="espec-fs" forbiddingContext=""/>
    </FILTERS>
  </NODE>
  <NODE initCond="parti-fp" id="150">
    <EVOLUTIONARY_RULES>
      <RULE ruleType="leftMostParsing" symbol="parti-fp" string="150-0_VMP00PF"
                                       nonTerminals="[All the grammar non-terminals]"/>
    </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="1" permittingContext="parti-fp" forbiddingContext=""/>
    </FILTERS>
  </NODE>
  <NODE initCond="verb" id="151">
    <EVOLUTIONARY_RULES>
      <RULE ruleType="leftMostParsing" symbol="verb" string="151-0_VAC*_SPS00_infinitiu"
                                       nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="verb" string="151-1_VAS*_SPS00_infinitiu"
                                       nonTerminals="[All the grammar non-terminals]"/>
      [And the same for every possible derivation of "verb"]
      [...]
    </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="1" permittingContext="verb" forbiddingContext=""/>
    </FILTERS>
  </NODE>
  <NODE initCond="pos-ms" id="152">
    <EVOLUTIONARY_RULES>
      <RULE ruleType="leftMostParsing" symbol="pos-ms" string="152-0_DP3MP0"
                                       nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="pos-ms" string="152-1_DP3MS0"
                                       nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="pos-ms" string="152-2_DP2MSP"
                                       nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="pos-ms" string="152-3_DP1MSP"
                                       nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="pos-ms" string="152-4_DP3CS0"
                                       nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="pos-ms" string="152-5_DP2CSS"
                                       nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="pos-ms" string="152-6_DP1CSS"
```

```
                                    nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="pos-ms" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="pos-mp" id="153">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="pos-mp" string="153-0_DP2MPP"
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="pos-mp" string="153-1_DP1MPP"
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="pos-mp" string="153-2_DP3CP0"
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="pos-mp" string="153-3_DP2CPS"
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="pos-mp" string="153-4_DP1CPS"
                                    nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="pos-mp" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="prel-fp" id="154">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="prel-fp" string="154-0_PR0FP000"
                                    nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="prel-fp" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="patons" id="155">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="patons" string="155-0_PP3CN000"
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="patons" string="155-1_paton"
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="patons" string="155-2_paton-fs"
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="patons" string="155-3_paton-ms"
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="patons" string="155-4_paton-fp"
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="patons" string="155-5_paton-mp"
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="patons" string="155-6_paton-p"
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="patons" string="155-7_paton-s"
                                    nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="patons" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="psubj" id="156">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="psubj" string="156-0_psubj-fp"
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="psubj" string="156-1_psubj-mp"
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="psubj" string="156-2_psubj-fs"
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="psubj" string="156-3_psubj-ms"
                                    nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="psubj" string="156-4_psubj-s"
                                    nonTerminals="[All the grammar non-terminals]"/>
```

```
    </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="1" permittingContext="psubj" forbiddingContext=""/>
    </FILTERS>
  </NODE>
  <NODE initCond="prel-mp" id="157">
    <EVOLUTIONARY_RULES>
      <RULE ruleType="leftMostParsing" symbol="prel-mp" string="157-0_PR0MP000"
                                       nonTerminals="[All the grammar non-terminals]"/>
    </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="1" permittingContext="prel-mp" forbiddingContext=""/>
    </FILTERS>
  </NODE>
  <NODE initCond="s-a-ms" id="158">
    <EVOLUTIONARY_RULES>
      <RULE ruleType="leftMostParsing" symbol="s-a-ms" string="158-0_parti-ms"
                                       nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="s-a-ms" string="158-1_a-ms_s-a-ms"
                                       nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="s-a-ms" string="158-2_sadv_a-ms"
                                       nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="s-a-ms" string="158-3_s-a-ms_coord_s-a-ms"
                                       nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="s-a-ms" string="158-4_s-a-ms_Fc_s-a-ms_Fs"
                                       nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="s-a-ms" string="158-5_s-a-ms_Fc_s-a-ms"
                                       nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="s-a-ms" string="158-6_a-ms"
                                       nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="s-a-ms" string="158-7_Fpa_s-a-ms_Fpt"
                                       nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="s-a-ms" string="158-8_Fe_s-a-ms_Fe"
                                       nonTerminals="[All the grammar non-terminals]"/>
    </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="1" permittingContext="s-a-ms" forbiddingContext=""/>
    </FILTERS>
  </NODE>
  <NODE initCond="interjeccio" id="159">
    <EVOLUTIONARY_RULES>
      <RULE ruleType="leftMostParsing" symbol="interjeccio" string="159-0_I"
                                       nonTerminals="[All the grammar non-terminals]"/>
    </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="1" permittingContext="interjeccio" forbiddingContext=""/>
    </FILTERS>
  </NODE>
  <NODE initCond="morfema-verbal" id="160">
    <EVOLUTIONARY_RULES>
      <RULE ruleType="leftMostParsing" symbol="morfema-verbal" string="160-0_P0000000"
                                       nonTerminals="[All the grammar non-terminals]"/>
    </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="1" permittingContext="morfema-verbal" forbiddingContext=""/>
    </FILTERS>
  </NODE>
  <NODE initCond="s-a-mp" id="161">
    <EVOLUTIONARY_RULES>
      <RULE ruleType="leftMostParsing" symbol="s-a-mp" string="161-0_parti-mp"
                                       nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="s-a-mp" string="161-1_a-mp_s-a-mp"
                                       nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="s-a-mp" string="161-2_sadv_a-mp"
                                       nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="s-a-mp" string="161-3_s-a-mp_coord_s-a-mp"
                                       nonTerminals="[All the grammar non-terminals]"/>
```

```
    <RULE ruleType="leftMostParsing" symbol="s-a-mp" string="161-4_s-a-mp_Fc_s-a-mp_Fs"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="s-a-mp" string="161-5_s-a-mp_Fc_s-a-mp"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="s-a-mp" string="161-6_a-mp"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="s-a-mp" string="161-7_Fpa_s-a-mp_Fpt"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="s-a-mp" string="161-8_Fe_s-a-mp_Fe"
                                     nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="s-a-mp" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="grup-nom-fs" id="162">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="grup-nom-fs" string="162-0_espec-fs_parti-fs"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-nom-fs" string="162-1_n-fs_pdem-fs"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-nom-fs" string="162-2_n-fs_pos-fs"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-nom-fs" string="162-3_s-a-fs_grup-nom-fs"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-nom-fs" string="162-4_w-fs_s-a-fs"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-nom-fs" string="162-5_n-fs_s-a-fs"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-nom-fs" string="162-6_n-fs_w-fs"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-nom-fs" string="162-7_w-fs_w-fs"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-nom-fs" string="162-8_w-fs"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-nom-fs" string="162-9_n-fs_n-fp"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-nom-fs" string="162-10_n-fs_n-mp"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-nom-fs" string="162-11_n-fs_n-ms"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-nom-fs" string="162-12_n-fs_n-fs"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-nom-fs" string="162-13_n-fs"
                                     nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="grup-nom-fs" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="v-hacer-3p" id="163">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="v-hacer-3p" string="163-0_VMIS3S0"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="v-hacer-3p" string="163-1_VMSF3S0"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="v-hacer-3p" string="163-2_VMSI3S0"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="v-hacer-3p" string="163-3_VMIC3S0"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="v-hacer-3p" string="163-4_VMIF3S0"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="v-hacer-3p" string="163-5_VMSP3S0"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="v-hacer-3p" string="163-6_VMII3S0"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="v-hacer-3p" string="163-7_VMIP3S0"
```

```
                                  nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="v-hacer-3p" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="prel-ms" id="164">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="prel-ms" string="164-0_PR0MS000"
                                  nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="prel-ms" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="grup-nom-fp" id="165">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="grup-nom-fp" string="165-0_espec-fp_parti-fp"
                                  nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-nom-fp" string="165-1_n-fp_pdem-fp"
                                  nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-nom-fp" string="165-2_n-fp_pos-fp"
                                  nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-nom-fp" string="165-3_pnum-fp_pnum-fp_pnum-fp"
                                  nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-nom-fp" string="165-4_pnum-fp_pnum-fp"
                                  nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-nom-fp" string="165-5_grup-c-nom-fp_s-a-fp"
                                  nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-nom-fp" string="165-6_grup-c-nom-fp"
                                  nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-nom-fp" string="165-7_s-a-fp_grup-nom-fp"
                                  nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-nom-fp" string="165-8_n-fp_s-a-fp"
                                  nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-nom-fp" string="165-9_w-fp_s-a-fp"
                                  nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-nom-fp" string="165-10_w-fp"
                                  nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-nom-fp" string="165-11_n-fp_n-fp"
                                  nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-nom-fp" string="165-12_n-fp_n-mp"
                                  nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-nom-fp" string="165-13_n-fp_n-ms"
                                  nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-nom-fp" string="165-14_n-fp_n-fs"
                                  nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="grup-nom-fp" string="165-15_n-fp"
                                  nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="grup-nom-fp" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="paton" id="166">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="paton" string="166-0_PP3CNA00"
                                  nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="paton" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="conj-subord" id="167">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="conj-subord" string="167-0_CS"
                                  nonTerminals="[All the grammar non-terminals]"/>
```

```
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="conj-subord" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="pnum-mp" id="168">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="pnum-mp" string="168-0_Z"
                                     nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="pnum-mp" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="int-fp" id="169">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="int-fp" string="169-0_DT0CN0"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="int-fp" string="169-1_DT0FP0"
                                     nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="int-fp" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="w-fs" id="170">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="w-fs" string="170-0_NP*"
                                     nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="w-fs" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="grup-verb-part" id="171">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="grup-verb-part" string="171-0_parti-flex"
                                     nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="grup-verb-part" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="pposs-mp" id="172">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="pposs-mp" string="172-0_DA0MP0_PX3MP0C0"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="pposs-mp" string="172-1_DA0MP0_PX2MP0P0"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="pposs-mp" string="172-2_DA0MP0_PX2MP0S0"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="pposs-mp" string="172-3_DA0MP0_PX1MP0P0"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="pposs-mp" string="172-4_DA0MP0_PX1MP0S0"
                                     nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="1" permittingContext="pposs-mp" forbiddingContext=""/>
  </FILTERS>
</NODE>
<NODE initCond="int-fs" id="173">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="leftMostParsing" symbol="int-fs" string="173-0_DT0CN0"
                                     nonTerminals="[All the grammar non-terminals]"/>
    <RULE ruleType="leftMostParsing" symbol="int-fs" string="173-1_DT0FS0"
                                     nonTerminals="[All the grammar non-terminals]"/>
  </EVOLUTIONARY_RULES>
```

```
    <FILTERS>
      <INPUT type="1" permittingContext="int-fs" forbiddingContext=""/>
    </FILTERS>
  </NODE>
  <NODE initCond="w-fp" id="174">
    <EVOLUTIONARY_RULES>
      <RULE ruleType="leftMostParsing" symbol="w-fp" string="174-0_NP*"
                                       nonTerminals="[All the grammar non-terminals]"/>
    </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="1" permittingContext="w-fp" forbiddingContext=""/>
    </FILTERS>
  </NODE>
  <NODE initCond="pposs-ms" id="175">
    <EVOLUTIONARY_RULES>
      <RULE ruleType="leftMostParsing" symbol="pposs-ms" string="175-0_DA0MS0_PX3MS0C0"
                                       nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="pposs-ms" string="175-1_DA0MS0_PX2MS0P0"
                                       nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="pposs-ms" string="175-2_DA0MS0_PX2MS0S0"
                                       nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="pposs-ms" string="175-3_DA0MS0_PX1MS0P0"
                                       nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="pposs-ms" string="175-4_DA0MS0_PX1MS0S0"
                                       nonTerminals="[All the grammar non-terminals]"/>
    </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="1" permittingContext="pposs-ms" forbiddingContext=""/>
    </FILTERS>
  </NODE>
  <NODE initCond="morf-pron" id="176">
    <EVOLUTIONARY_RULES>
      <RULE ruleType="leftMostParsing" symbol="morf-pron" string="176-0_P0300000"
                                       nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="morf-pron" string="176-1_P020P000"
                                       nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="morf-pron" string="176-2_P020S000"
                                       nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="morf-pron" string="176-3_P010P000"
                                       nonTerminals="[All the grammar non-terminals]"/>
      <RULE ruleType="leftMostParsing" symbol="morf-pron" string="176-4_P010S000"
                                       nonTerminals="[All the grammar non-terminals]"/>
    </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="1" permittingContext="morf-pron" forbiddingContext=""/>
    </FILTERS>
  </NODE>
  <NODE initCond="pnum-ms" id="177">
    <EVOLUTIONARY_RULES>
      <RULE ruleType="leftMostParsing" symbol="pnum-ms" string="177-0_Z"
                                       nonTerminals="[All the grammar non-terminals]"/>
    </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="1" permittingContext="pnum-ms" forbiddingContext=""/>
    </FILTERS>
  </NODE>
  <NODE initCond="">
    <EVOLUTIONARY_RULES>
      <RULE ruleType="insertion" actionType="RIGHT" symbol="%"/>
      <RULE ruleType="insertion" actionType="LEFT" symbol="%"/>
    </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="2" permittingContext="PP*_PP3MS000_VSI*_VSIP3S0_NCMS*_NCMS00*_NCMS000_0-0_0-1_0-2_0-3_0-4_
                                         0-5_0-6_0-7_0-8_0-9_0-10_0-11_0-12_0-13_0-14_0-15_0-16_0-17_0-18_
                                         0-19_0-20_0-21_0-22_0-23_0-24_0-25_0-26_0-27_0-28_0-29_0-30_0-31_
                                         [And so on for every production rule id]_
                                         175-4_176-0_176-1_176-2_176-3_176-4_177-0" forbiddingContext=""/>
```

```
      <OUTPUT type="RegularLangMembershipFilter" regularExpression="%%.*|%.*%|.*%%"/>
    </FILTERS>
  </NODE>
  <NODE initCond="">
    <EVOLUTIONARY_RULES>
      <RULE ruleType="deletion" actionType="RIGHT" symbol=""/>
      <RULE ruleType="splicingParsing" wordX="PP3MS000" wordY="%" wordU="%" wordV="VSIP3S0"
                                       ignoredSymbols="[Every production rule id]" />
    </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="RegularLangMembershipFilter" regularExpression="(%[^%]*PP3MS000%)|(%(0-0|0-1|0-2|0-3|0-4|0-5|0-6|
                                                                  [And so on for every production rule id]
                                                                  |176-2|176-3|176-4|177-0)*VSIP3S0[^%]*%)"/>
    </FILTERS>
  </NODE>
  <NODE initCond="">
    <EVOLUTIONARY_RULES>
      <RULE ruleType="deletion" actionType="RIGHT" symbol=""/>
      <RULE ruleType="splicingParsing" wordX="PP3MS000" wordY="%" wordU="%" wordV="VSI*"
                                       ignoredSymbols="[Every production rule id]" />
    </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="RegularLangMembershipFilter" regularExpression="(%[^%]*PP3MS000%)|(%(0-0|0-1|0-2|0-3|0-4|0-5|0-6|
                                                                  [And so on for every production rule id]
                                                                  |176-2|176-3|176-4|177-0)*VSI\*[^%]*%)"/>
    </FILTERS>
  </NODE>
  <NODE initCond="">
    <EVOLUTIONARY_RULES>
      <RULE ruleType="deletion" actionType="RIGHT" symbol=""/>
      <RULE ruleType="splicingParsing" wordX="PP*" wordY="%" wordU="%" wordV="VSIP3S0"
                                       ignoredSymbols="[Every production rule id]" />
    </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="RegularLangMembershipFilter" regularExpression="(%[^%]*PP\*%)|(%(0-0|0-1|0-2|0-3|0-4|0-5|0-6|
                                                                  [And so on for every production rule id]
                                                                  |176-2|176-3|176-4|177-0)*VSIP3S0[^%]*%)"/>
    </FILTERS>
  </NODE>
  <NODE initCond="">
    <EVOLUTIONARY_RULES>
      <RULE ruleType="deletion" actionType="RIGHT" symbol=""/>
      <RULE ruleType="splicingParsing" wordX="PP*" wordY="%" wordU="%" wordV="VSI*"
                                       ignoredSymbols="[Every production rule id]" />
    </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="RegularLangMembershipFilter" regularExpression="(%[^%]*PP\*%)|(%(0-0|0-1|0-2|0-3|0-4|0-5|0-6|
                                                                  [And so on for every production rule id]
                                                                  |176-2|176-3|176-4|177-0)*VSI\*[^%]*%)"/>
    </FILTERS>
  </NODE>
  <NODE initCond="">
    <EVOLUTIONARY_RULES>
      <RULE ruleType="deletion" actionType="RIGHT" symbol=""/>
      <RULE ruleType="splicingParsing" wordX="VSIP3S0" wordY="%" wordU="%" wordV="NCMS000"
                                       ignoredSymbols="[Every production rule id]" />
    </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="RegularLangMembershipFilter" regularExpression="(%[^%]*VSIP3S0%)|(%(0-0|0-1|0-2|0-3|0-4|0-5|0-6|
                                                                  [And so on for every production rule id]
                                                                  |176-2|176-3|176-4|177-0)*NCMS000[^%]*%)"/>
    </FILTERS>
  </NODE>
  <NODE initCond="">
    <EVOLUTIONARY_RULES>
      <RULE ruleType="deletion" actionType="RIGHT" symbol=""/>
      <RULE ruleType="splicingParsing" wordX="VSIP3S0" wordY="%" wordU="%" wordV="NCMS*"
```

```
                                   ignoredSymbols="[Every production rule id]" />
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="RegularLangMembershipFilter" regularExpression="(%[^%]*VSIP3S0%)|(%(0-0|0-1|0-2|0-3|0-4|0-5|0-6|
                                                                 [And so on for every production rule id]
                                                                 |176-2|176-3|176-4|177-0)*NCMS\*[^%]*%)"/>
  </FILTERS>
</NODE>
<NODE initCond="">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="deletion" actionType="RIGHT" symbol=""/>
    <RULE ruleType="splicingParsing" wordX="VSIP3S0" wordY="%" wordU="%" wordV="NCMS00*"
                                   ignoredSymbols="[Every production rule id]" />
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="RegularLangMembershipFilter" regularExpression="(%[^%]*VSIP3S0%)|(%(0-0|0-1|0-2|0-3|0-4|0-5|0-6|
                                                                 [And so on for every production rule id]
                                                                 |176-2|176-3|176-4|177-0)*NCMS00\*[^%]*%)"/>
  </FILTERS>
</NODE>
<NODE initCond="">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="deletion" actionType="RIGHT" symbol=""/>
    <RULE ruleType="splicingParsing" wordX="VSI*" wordY="%" wordU="%" wordV="NCMS000"
                                   ignoredSymbols="[Every production rule id]" />
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="RegularLangMembershipFilter" regularExpression="(%[^%]*VSI\*%)|(%(0-0|0-1|0-2|0-3|0-4|0-5|0-6|
                                                                 [And so on for every production rule id]
                                                                 |176-2|176-3|176-4|177-0)*NCMS000[^%]*%)"/>
  </FILTERS>
</NODE>
<NODE initCond="">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="deletion" actionType="RIGHT" symbol=""/>
    <RULE ruleType="splicingParsing" wordX="VSI*" wordY="%" wordU="%" wordV="NCMS*"
                                   ignoredSymbols="[Every production rule id]" />
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="RegularLangMembershipFilter" regularExpression="(%[^%]*VSI\*%)|(%(0-0|0-1|0-2|0-3|0-4|0-5|0-6|
                                                                 [And so on for every production rule id]
                                                                 |176-2|176-3|176-4|177-0)*NCMS\*[^%]*%)"/>
  </FILTERS>
</NODE>
<NODE initCond="">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="deletion" actionType="RIGHT" symbol=""/>
    <RULE ruleType="splicingParsing" wordX="VSI*" wordY="%" wordU="%" wordV="NCMS00*"
                                   ignoredSymbols="[Every production rule id]" />
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="RegularLangMembershipFilter" regularExpression="(%[^%]*VSI\*%)|(%(0-0|0-1|0-2|0-3|0-4|0-5|0-6|
                                                                 [And so on for every production rule id]
                                                                 |176-2|176-3|176-4|177-0)*NCMS00\*[^%]*%)"/>
  </FILTERS>
</NODE>
<NODE initCond="">
  <EVOLUTIONARY_RULES>
    <RULE ruleType="deletion" actionType="RIGHT" symbol=""/>
  </EVOLUTIONARY_RULES>
  <FILTERS>
    <INPUT type="RegularLangMembershipFilter" regularExpression="%[0-9\-]*(PP3MS000|PP\*)?[0-9\-]*(VSIP3S0|VSI\*)?
                                                                 [0-9\-]*(NCMS000|NCMS\*|NCMS00\*)?%"/>
  </FILTERS>
</NODE>
<NODE initCond="">
  <EVOLUTIONARY_RULES>
```

```
      <RULE ruleType="deletion" actionType="RIGHT" symbol=""/>
    </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="2" permittingContext="PP*_PP3MS000_VSI*_VSIP3S0_NCMS*_NCMS00*_NCMS000_0-0_0-1_0-2_0-3_0-4_0-5_
                                        0-6_0-7_0-8_0-9_0-10_0-11_0-12_0-13_0-14_0-15_0-16_0-17_0-18_0-19_0-20_
                                        0-21_0-22_0-23_0-24_0-25_0-26_0-27_0-28_0-29_0-30_0-31_0-32_0-33_0-34_
                                        [And so on for every production rule id]_
                                        174-0_175-0_175-1_175-2_175-3_175-4_176-0_176-1_176-2_176-3_176-4_177-0_
                                        [All the grammar non-terminals]" forbiddingContext=""/>
    </FILTERS>
  </NODE>
  <NODE initCond="">
    <EVOLUTIONARY_RULES>
      <RULE ruleType="deletion" actionType="RIGHT" symbol=""/>
    </EVOLUTIONARY_RULES>
    <FILTERS>
      <INPUT type="RegularLangMembershipFilter" regularExpression="%[0-9\-]*(PP3MS000|PP\*)[0-9\-]*(VSIP3S0|VSI\*)
                                                                   [0-9\-]*(NCMS000|NCMS\*|NCMS00\*)%"/>
      <OUTPUT type="1" permittingContext="" forbiddingContext="PP*_PP3MS000_VSI*_VSIP3S0_NCMS*_NCMS00*_NCMS000"/>
    </FILTERS>
  </NODE>
</EVOLUTIONARY_PROCESSORS>

<STOPPING_CONDITION>
  <CONDITION type="NonEmptyNodeStoppingCondition" nodeID="191"/>
</STOPPING_CONDITION>
</NEP>
```

Bibliography

A. L. Abu Dalhoum, M. Alfonseca, M. Cebrian, R. Sanchez-Alfonso, and A. Ortega. Computer-generated music using grammatical evolution, 2008.

L. M. Adleman. Molecular computation of solutions to combinatorial problem. *Science*, 266:1021–1024, 1994.

A.V. Aho, R. Sethi, and J.D. Ullman. *Compiladores. Principios, Técnicas y Herramientas.* Addison-Wesley, 1998.

B. J. Alexander and M. J. Gratton. Constructing an optimisation phase using grammatical evolution, 2009.

E. Alfonseca. *An Approach for Automatic Generation of on-line Information Systems based on the Integration of Natural Language Processing and Adaptive Hypermedia techniques.* PhD thesis, Computer Science Department, UAM, 2003.

M. Alfonseca Moreno, M. de la Cruz Echeandia, A. Ortega de la Puente, and E. Pulido Cañabate. *Compiladores e interpretes: teoría y práctica.* Pearson. Prentice Hall, 2006.

L. Alonso, R. Moreno, M. Vázquez, E. del Rosal, and J. Santacreu. Spontaneous recovery of conditioned response in an autonomous agent. *Estudios de Psicología*, 26(3):365–376, 2005a.

L. Alonso, R. Moreno, M. Vázquez, and J. Santacreu. Simulation of the filtering role of habituation to stimuli. *The Spanish Journal of Psychology*, 8(2):134–141, 2005b.

W. Banzhaf. *Genotype-phenotype-mapping and neutral variation - A case study in genetic programming*, pages 322–332. Jerusalem. Springer-Verlag, 1994.

D. Beaumont and S. Stepney. Grammatical evolution of l-systems, 2009.

W. Bechtel and A. Abrahamsen. *Connectionism and the Mind.* Blackwell Publishers, 2002.

G. Bel Enguix, M. D. Jimenez-Lopez, R. Mercaş, and A Perekrestenko. Networks of evolutionary processors as natural language parsers. In *Proceedings ICAART 2009*, 2009.

A. Brabazon and M. O'Neill. Anticipating bankruptcy reorganisation from raw financial data using grammatical evolution. *Applications of Evolutionary Computing*, 2611:368–377, 2003.

A. Brabazon and M. O'Neill. Bond-issuer credit rating with grammatical evolution. *Applications of Evolutionary Computing*, 3005:270–279, 2004.

A. Brabazon, M. O'Neill, R. Matthews, and C. Ryan. Grammatical evolution and coporate failure prediction. In W. B. Langdon, E. Cant-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M. A. Potter, A. C. Schultz, J. F. Miller, E. Burke, and N. Jonoska, editors, *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1011–1018. New York. Morgan Kaufamann Publishers, 2002a.

A. Brabazon, M. O'Neill, C. Ryan, and R. Matthews. Evolving classifiers to model the relationship between strategy and corporate performance using grammatical evolution. In E. Lutton, J. A. Foster, J. Miller, Ryan C., and A. G. B. Tettamanzi, editors, *Proceedings of the 4th European Conference on Genetic Programming, EuroGP 2002*, volume 2278 of LNCS, pages 103–112. Kinsale, Ireland. Springer-Verlag, 2002b.

T. Brabazon and M. O'Neill. Trading foreign exchange markets using evolutionary automatic programming. In A. M. Barry, editor, *Proceedings of the Bird of a Featther Workshops, Genetic and Evolutionary Computation Conference*, pages 133–136. New York. AAAI, 2002.

R. Bradley, A. Brabazon, and M. O'Neill. Objective function design in a grammatical evolutionary trading system, 2010.

S. E. Brandon, E. H. Vogel, and A. R. Wagner. Stimulus representation in sop: I theoretical rationalization and some implications. *Behavioural Processes*, 62(1-3): 5–25, 2003.

T. Brants. Tnt–a statistical part-of-speech tagger. In *Proceedings of the 6th Conference on Applied Natural Language Processing*, 2000.

R. Burbidge, J. H. Walker, and M. S. Wilson. Grammatical evolution of a robot controller, 2009.

J. H. Byrne. Analysis of synaptic depression contributing to habituation of gill-wthdrawal reflex in aplysia californica. *Journal of Neurophysiology*, 48:431–438, 1982.

T. J. Carew, H. M. Pinsker, and E. R. Kandel. Long-term habituation of a defensive withdrawal reflex in aplysia. *Science*, 175:451–454, 1972.

J. Castellanos, C. Martín-Vide, V. Mitrana, and J. M. Sempere. Solving np-complete problems with networks of evolutionary processors. In *Connectionist Models of Neurons, Learning Processes and Artificial Intelligence : 6th International Work-Conference on Artificial and Natural Neural Networks, IWANN 2001 Granada, Spain, June 13-15, 2001, Proceedings, Part I*, pages 621–, 2001.

J. Castellanos, C. Martin-Vide, V. Mitrana, and J. M. Sempere. Networks of evolutionary processors. *Acta Informatica*, 39(6-7):517–529, 2003.

J. Castellanos, P. Leupold, and V. Mitrana. On the size complexity of hybrid networks of evolutionary processors. *Theoretical Computer Science*, 330(2):205–220, 2005.

L. Chen. Macro-grammatical evolution for nonlinear time series modeling-a case study of reservoir inflow forecasting. *Engineering With Computers*, 27(4):393–404, October 2011.

L. Chen and T. S. Wang. Modeling strength of high-performance concrete using an improved grammatical evolution combined with macrogenetic algorithm. *Journal of Computing In Civil Engineering*, 24(3):281–288, May 2010.

A. Choudhary and K. Krithivasan. Network of evolutionary processors with splicing rules. *Mechanisms, Symbols and Models Underlying Cognition, PT 1, PROCEEDINGS*, 3561:290–299, 2005.

H. Christiansen. Syntax, semantics and implementation strategies for programmign languages with powerful abstraction mechanisms. In *Proceedings of the Eighteenth Annual Hawaii International Conference on System Sciences*, volume 2, pages 57–66, 1985.

H. Christiansen. Recognition of generative languages. *Lecture Notes in Computer Science*, 217:63–81, 1986.

H. Christiansen. A survey of adaptable grammars. *Sigplan Notices*, 25(11):35–44, 1990.

R. Cleary and M. O'Neill. An attribute grammar decoder for the 01 multiconstrained knapsack problem. *Evolutionary Computation in Combinatorial Optimization, Proceedings*, 3448:34–45, 2005.

E. Csuhaj-Varjú and V. Mitrana. Evolutionary systems: A language generating device inspired by evolving communities of cells. *Acta Informatica*, 36:913–926, 2000.

E. Csuhaj-Varju and A. Salomaa. *Lecture Notes on Computer Science 1218*, chapter Networks of parallel language processors. 1997.

E. Csuhaj-Varjú, J. Dassow, J. Kelemen, and G. Paun. *Grammar Systems*. London, Gordon and Breach, 1993.

E. Csuhaj-Varju, C. Martin-Vide, and V. Mitrana. Hybrid networks of evolutionary processors are computationally complete. *Acta Informatica*, 41(4-5):257–272, 2005.

W. Cui, A. Brabazon, and M. O'Neill. Evolving dynamic trade execution strategies using grammatical evolution, 2010.

J. Cullen. Evolving digital circuits in an industry standard hardware description language, 2008.

L. N. de Castro. *Fundamentals of Natural Computing. Basic Concepts, Algorithms, and Applications*. Chapman & Hall/CRC, 2006.

M. de la Cruz, A. Jiménez, E. del Rosal, G. Bel-Engix, and A. Ortega. NEPs-lingua: a new textual language to program NEPs. In *Proceedings of the 3rd International Conference on Agents and Artificial Intelligence*, 2011.

E. del Rosal and M. Cuéllar. *Methods and Models in Artificial and Natural Computation. A Homage to Professor Mira's Scientific Legacy*, volume 561 of *LNCS*, chapter jNEPView: a graphical trace viewer for the simulations of NEPs, pages 356–366. Springer, 2009.

E. del Rosal, L. Alonso, R. Moreno, M. Vázquez, and J. Santacreu. Simulation of habituation to simple and multiple stimuli. *Behavioural Processes*, 73:272–277, 2006.

E. del Rosal, A. Ortega, M. Alfonseca, and M. de la Cruz. Christiansen grammar evolution for the modelling of psychological processes. In J. Ottjes and H. Veeke, editors, *Proccedings of the 5th International Industrial Simulation Conference (ISC 2007)*, pages 99–104, Delft, The Netherlands., 2007. EUROSIS-ETI.

E. del Rosal, R. Nuñez, C. Castañeda, and A. Ortega. Simulating NEPs in a cluster with jNEP. *International Journal of Computers, Communications and Control. Supplementary Issue Proccedings of ICCCC 2008*, III:480–485, 2008.

E. del Rosal, R. Nuñez, C. Castañeda, and A. Ortega. *From Natural Language to Soft Computing: New Paradigms in Artificial Intelligence*, chapter Simulating NEPs in a cluster with jNEP. Editing House of Romanian Academy, 2009a.

E. del Rosal, J. M. Rojas, R. Núñez, C. Castañeda, and A. Ortega. On the solution of NP-complete problems by means of jNEP run on computers. In *Proceedings of International Conference on Agents and Artificial Intelligence (ICAART 2009), pages 605-612, Porto, Portugal, 19-21 January 2009. INSTICC Press.*, 2009b.

E. del Rosal, A. Ortega de la Puente, and D. Perez-Marin. PNEPs for shallow parsing - NEPs extended for parsing applied to shallow parsing. In *Proceedings of ICAART 2010 Second International Conference on Agents an Artificial Intelligence*, pages 403–410, 2010.

E. del Rosal, M. de la Cruz, and A. Ortega de la Puente. *Foundations on Natural and Artificial Computation*, volume 6686 of *LNCS*, chapter Towards the automatic programming of NEPs, pages 303–312. Springer-Verlag, Berlin, Heidelberg, 2011. ISBN 978-3-642-21343-4.

E. del Rosal, A. Ortega, and M. de la Cruz. *Advances in Intelligent and Soft Computing*, chapter Towards the automatic programming of NEPs: a first case study, pages 37–44. Springer, 2012.

M. A. Díaz, N. Gómez Blas, E. Santos Menéndez, R. Gonzalo, and F. Gisbert. Networks of evolutionary processors (NEP) as decision support systems. In *Fith International Conference. Information Research and Applications*, volume 1, pages 192–203. ETHIA, 2007.

A. Dickinson and J Burke. *The essentials of conditioning and learning.* Pacific Grove: Brooks/Cole Publishing, 1996.

J. Earley. An efficient context-free parsing algorithm. *Communications of the ACM*, 13(2):94–102, 1970.

M. D. Echeandia, A. O. de la Puente, and M. Alfonseca. Attribute grammar evolution. *Artificial Intelligence and Knowledge Engineering Applications: a Bioinspired Approach, PT 2, Proceedings*, 3562:182–191, 2005.

H. Ehrig, K. Ehrig, U. Prange, and G. Taentzer. *Fundamentals of algebraic graph transformation.* Springer-Verlag, 2006.

A. E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing.* Berlin Heidelberg: Springer-Verlag, 2003.

L. Errico and C. Jesshope. Towards a new architecture for symbolic processing. In I. Plander, editor, *Artificial Intelligence and Information-Control Systems of Robots '94.* Singapore, World Sci. Publ., 1994.

L. J. Fogel, A. J. Owens, and M. J. Walsh. *Artificial Intelligence through Simulated Evolution.* John Wiley, 1966.

S. Forrest, B. Javornik, R. E. Smith, and A. S. Perelson. Using genetic algorithms to explore pattern recognition in the immune system. *Evolutionary Computation*, 1(3):191–211, 1993.

J. J. Freeman. A linear representation for GP using context free grammars. In J. R. Koza, W. Banzhaf, K. Chellapilla, K. Deb, M. Dorigo, D. B. Fogel, M. H. Garzon, D. E. Goldberg, H. Iba, and R. Riolo, editors, *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 72–77, 1998.

E. Galvan-Lopez, J. M. Swafford, M. O'Neill, and A. Brabazon. Evolving a ms. pacman controller using grammatical evolution, 2010.

M. García-Quismondo, R. Gutiérrez-Escudero, M. A. Martínez del Amor, E. Orejuela, and I. Pérez-Hurtado. P-lingua 2.0: A software framework for cell–like p systems. *International Journal of Computers, Communications and Control*, IV (3):234–243, 2009.

M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. WH Freeman, New York, 1979.

C. Gomez, F. Javier, D. Valle Agudo, J. Rivero Espinosa, and D. Cuadra Fernandez. *Procesamiento del lenguaje Natural*, chapter Methodological approach for pragmatic annotation,, pages 209–216. 2008.

L. K. Grover. Quantum computers can search arbitrarily large databases by a single query. *Physical Review Letters*, 79(23):4709–4712, 1997.

P. M. Groves and R. F. Thompson. Habituation: a dual-process theory. *Psychological Review*, 77:419–450, 1970.

G. Hall. *Perceptual and associative learning*. Oxford: Clarendon, 1991.

Z. S. Harris. *String Analysis of Sentence Structure*. Mouton, The Hague, 1962.

S. S. Haykin. *Neural networks and learning machines*. Upper Saddle River, 2009.

M. Hemberg and U. M. O'Reilly. Extending grammatical evolution to evolve digital surfaces with genr8. *Genetic Programming, Proceedings*, 3003:299–308, 2004.

W.D. Hillis. *The Connection Machine*. Cambridge, MIT Press, 1985.

IBM. http://www.haifa.il.ibm.com/projects/systems/cjvm/index.html, 2000. URL `http://www.haifa.il.ibm.com/projects/systems/cjvm/index.html`.

Company Inria Sophia Antipolis. http://www-sop.inria.fr/sloop/javall/, 2008. URL `http://www-sop.inria.fr/sloop/javall/`.

JavaParty. http://svn.ipd.uni-karlsruhe.de/trac/javaparty/wiki/javaparty?redirectedfrom=wikistart, 2008. URL `http://svn.ipd.uni-karlsruhe.de/trac/javaparty/wiki/JavaParty?redirectedfrom=WikiStart`.

JESSICA2. http://i.cs.hku.hk/ clwang/projects/jessica2.html, 2008. URL `http://i.cs.hku.hk/~clwang/projects/JESSICA2.html`.

JGraph. http://www.jgraph.com/jgraph.html, 2009.

JGraphT. http://jgrapht.sourceforge.net/, 2009.

A. Jimenez, E. del Rosal, and J. de Lara. *Trends in Practical Applications of Agents and Multiagent Systems*, volume 71 of *Advances in Intelligent and Soft Computing*, chapter A Visual Language for Modelling and Simulation of Networks of Evolutionary Processors, pages 411–418. Springer, 2010.

D. Jurafsky and J. H. Martin. *Speech and Language Processing. An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition.* Prentice Hall, 2000.

C. M. Kao, L. Chen, C. C. Wei, and Y. R. Fu. Grammatical evolution for total phosphorus in reservoir prediction, 2011.

M. Keijzer, C. Ryan, M. O'Neill, M. Cattolico, and V. Babovic. Ripple crossover in genetic programming. *Genetic Programming, Proceedings*, 2038:74–86, 2001.

S. Kelly and J. Tolvanen. *Domain-Specific Modeling: Enabling Full Code Generation.* Wiley-IEEE Computer Society, 2008.

D. E. Knuth. *Mathematical Systems Theory*, volume 2, chapter Semantics of Context-Free Languages, pages 127–145. 1968.

T. Kohonen. The self-organizing map. *Proceedings of the IEEE*, 78(9):1464–1480, 1990.

J. R. Koza. Hierarchical genetic algorithms operating on populations of computer programs. In N. Sridharan, editor, *Proceedings of the 11th International Conference on Artificial Intelligence*, page 768774. Morgan Kaufmann, 1989.

J. R. Koza. *Genetic Programming: on the Programming of Computers by Means of Natural Selection.* MIT Press, Cambridge, MA, USA., 1992.

J. R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs.* MIT Press, Cambridge Massachusetts, 1994.

J. R. Koza, David, F. H. re, Bennett III, and M. Keane. *Genetic Programming 3: Darwinian Invention and Problem Solving.* Morgan Kaufman, 1999.

R. Lara. A model of the neural mechanisms responsible for stimulus specific habituation of the orienting reflex in vertebrates. *Cognition & Brain Theory*, 6:463–482, 1983.

P. Launay and J. L. Pazat. A framework for parallel programming in java, 1997.

A. Lindenmayer. Mathematical models for cellular interaction in development. *Journal of Theoretical Biology*, 18:280–315, 1968.

N. J. Mackintosh. A theory of attention: Variations in the associability of stimuli with reinforcement. *Psychological Review*, 82:276–298, 1975.

B. Mandelbrot. *The Fractal Geometry of Nature.* W. H. Freemand and Company, 1983.

F. Manea. Using AHNEPs in the recognition of context free languages. In *Proceedings of the Workshop on Symbolic Networks, ECAI 2004.*, 2004a. Unpublished document.

F. Manea. Using AHNEPs in the recognition of context-free languages. In *In Proceedings of the Workshop on Symbolic Networks ECAI*, 2004b.

F. Manea and V. Mitrana. All np-problems can be solved in polynomial time by accepting hybrid networks of evolutionary processors of constant size. *Information Processing Letters*, 103(3):112–118, July 2007.

F. Manea, C. Martín-Vide, and V. Mitrana. All NP-Problems can be solved in polynomial time by accepting networks of splicing processors of constant size. *DNA Computing*, pages 47–57, 2006.

F. Manea, C. Martin-Vide, and V. Mitrana. Accepting networks of splicing processors: Complexity results. *Theoretical Computer Science*, 371(1-2):72–82, February 2007.

M. Margenstern, V. Mitrana, and M. J. Perez-Jimenez. Accepting hybrid networks of evolutionary processors. *DNA Computing*, 3384:235–246, 2005.

G. Martin and J. Pear. *Behavior Modification*. Upper Saddle Rider, NJ: Prentice-Hall, 2002.

J. Martin. *Introduction to Languages and the Theory of Computation*. McGraw-Hill, 2003.

C. Martín-Vide and V. Mitrana. Solving 3cnf-sat and hpp in linear time using www. *Machines, Computations, and Universality*, 3354:269–280, 2005.

C. Martin-Vide, V. Mitrana, M. J. Perez-Jimenez, and F. Sancho-Caparrini. Hybrid networks of evolutionary processors. *Genetic and Evolutionary Computation. GECCO 2003, PT I, Proceedings*, 2723:401–412, 2003.

R. Matousek and J. Bednar. Grammatical evolution: Epsilon tube in symbolic regression task, 2009a.

R. Matousek and J. Bednar. Grammatical evolution: Epsilon tube in symbolic regression task, 2009b.

J. E. Mazur. *Learning and behavior*. Upper Saddle River(New Jersey). Prentice-Hall, 2002.

W. McCulloch and W. H. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical biophysics*, 5:115–133, 1943.

B. A. McKinney, J. E. Crowe, H. U. Voss, P. S. Crooke, N. Barney, and J. H. Moore. Hybrid grammar-based approach to nonlinear dynamical system identification from biological time series. *Physical Review*, 73(2):021912, 2006.

R. Mikheev. Periods, capitalized words, etc. *Comput. Linguist.*, 28(3):289–318, 2002. ISSN 0891-2017.

R. Mitkov. *The Oxford Handbook of Computational Linguistics*. Oxford University Press, 2003.

J. H. Moore and L. W. Hahn. Petri net modeling of high-order genetic systems using grammatical evolution. *Biosystems*, 72(1-2):177–186, 2003.

J. H. Moore and L. W. Hahn. An improved grammatical evolution strategy for hierarchical petri net modeling of complex genetic systems. *Applications of Evolutionary Computing*, 3005:63–72, 2004.

J. H. Moore, E. M. Boczko, and M. L. Summar. Connecting the dots between genes, biochemistry, and disease susceptibility: systems biology modeling in human genetics. *Molecular Genetics and Metabolism*, 84(2):104–111, 2005.

S. Muggleton. *Inductive Logic Programming*. London: Academic, 1992.

C. Navarrete Navarrete, M. de la Cruz Echeandia, E. Anguiano Rey, A. Ortega de la Puente, and J. M. Rojas Silas. Parallel simulation of NEPs on clusters. *Web Intelligence and Intelligent Agent Technology, IEEE/WIC/ACM International Conference on*, 3:171–174, 2011.

M. Nicolau and D. Costelloe. Using grammatical evolution to parameterise interactive 3d image generation, 2011.

W. T. O'Donohue and R. Kitchener. *Handbook of behaviorism.* San Diego: Academic Press, 1999.

M. O'Neill and A. Brabazon. mgga: The meta-grammar genetic algorithm. *Genetic Programming, Proceedings*, 3447:311–320, 2005.

M. O'Neill and C. Ryan. Genetic code degeneracy: Implications for grammatical evolution and beyond. *Advances in Artificial Life, Proceedings*, 1674:149–153, 1999a.

M. O'Neill and C. Ryan. Evolving multi-line compilable c programs. *Genetic Programming*, 1598:83–92, 1999b.

M. O'Neill and C. Ryan. *Evolutionary Algorithms in Engineering and Computer Science*, chapter Automatic generation of caching algorithms, pages 127–134. Jyvskyl, Finland. John Wiley & Sons, 1999c.

M. O'Neill and C. Ryan. Crossover in grammatical evolution: A smooth operator? *Genetic Programming, Proceedings*, 1802:149–162, 2000.

M. O'Neill and C. Ryan. *Grammatical Evolution. Evolutionary Automatic Programming in an Arbitrary Language.* Kluwer Academic Publishers, 2003.

M. O'Neill and C. Ryan. Grammatical evolution by grammatical evolution: The evolution of grammar and genetic code. *Genetic Programming, Proceedings*, 3003: 138–149, 2004.

M. O'Neill, A. Brabazon, C. Ryan, and J. Collins. Developing a market timing system using grammatical evolution. In L. spector, E. D. Goodman, A. Wu, W. B. Langdon, H.-M. Voigt, M. Gen, S. Sen, S. Dorigo, M. an Pezeshk, M. H. Grazon, and E. Burke, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 1375–1381. San Francisco, California, USA. Morgan Kaufmann, 2001a.

M. O'Neill, A. Brabazon, C. Ryan, and J. J. Collins. Evolving market index trading rules using grammatical evolution. *Applications of Evolutionary Computing, Proceedings*, 2037:343–352, 2001b.

M. O'Neill, C. Ryan, M. Keijzer, and M. Cattolico. Crossover in grammatical evolution: The search continues. *Genetic Programming, Proceedings*, 2038:337–347, 2001c.

M. O'Neill, C. Ryan, and M. Nicolau. Grammar defined introns: An investigation into grammar, introns, and bias in grammatical evolution. In L. Spector, E. D. Goodman, A. Wu, W. B. Langdom, H.-M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M. H. Garzon, and E. Burke, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 97–103. San Francisco, California, USA. Morgan Kaufmann, 2001d.

M. O'Neill, A. Brabazon, and C. Ryan. *Genetic Algorithms and Genetic Programming in Economics and Finance*, chapter Forecasting market indices using evolutionary automatic programming. A case study. Kluwer Academic Publishers, 2002.

M. O'Neill, A. Brabazon, M. Nicolau, S. Mcgarraghy, and P. Keenan. pi grammatical evolution. *Genetic and Evolutionary Computation GECCO 2004 , PT 2, Proceedings*, 3103:617–629, 2004.

A. Ortega, A. A. Dalhoum, and M. Alfonseca. Grammatical evolution to design fractal curves with a given dimension. *IBM Journal of Research and Development*, 47(4):483–493, 2003.

A. Ortega, M. de la Cruz, and M. Alfonseca. Christiansen grammar evolution: grammatical evolution with semantics. *IEEE Transactions on Evolutionary Computation*, 11-1:77–90, 2007.

A. Ortega, E. del Rosal, D. Pérez, R. Merca, A. Perekrestenko, and M. Alfonseca. *Bio-Inspired Systems: Computational and Ambient Intelligence*, volume 5517 of *LNCS*, chapter PNEPs, NEPs for Context Free Parsing: Application to Natural Language Processing, pages 472–479. Springer, 2009.

A. Ortega, E. del Rosal, D. R. Pérez, R. Mercas, R. Perekrestenko, and M. Alfonseca. *Bio-Inspired Models for Natural and Formal Languages*, chapter PNEPs, NEPs extension to parse context free languages, pages 305–335. Cambridge Scholar Publishing, 2011.

A. Ortega de la Puente, M. de la Cruz Echeandía, E. del Rosal, C. Navarrete Navarrete, A. Jiménez Martínez, J. de Lara, E. Anguiano Rey, M. Cuéllar, and J. M. Rojas Siles. Developing tools for networks of processors. To be published in the journal "Triangle", 2012.

L. Padró, M. Collado, S. Resse, M. Lloberes, and I. Castellón. Freeling 2.1: five years of open-source language processing tools. In *Proceedings of the Seventh Conference on International Language Resources and Evaluation (LREC-10)*, page 93136, La Valletta, Malta., 2010.

S. Papert. *Mindstorms: Children, Computers, and Powerful Ideas.* Basic Books, 1980.

N. R. Paterson and M. Livesey. Distinguishing gnotype an phenotype in genetic programming. In J. R. Koza, editor, *Late Breaking Papers at the Genetic Programming 1996 Conference Stanford University July 28-31, 1996*, pages 141–150. Stanford University, CA, USA. Stanford Bookstore, 1996.

I. P. Pavlov. *Conditioned reflexes.* London: Oxford University Press., 1927.

G. Pearce, J. M. & Hall. A model for pavlovian learning: Variations in the effectiveness of conditioned but not of unconditioned stimuli. *Psychological Review*, 87:532–552, 1980.

A. M. Peleteiro, J. C. Burguillo, Z. Oplatkova, and I. Zelinka. Epmas: Evolutionary programming multi-agent systems, 2010.

D. Perez, M. Nicolau, M. O'Neil, and A. Brabazon. Evolving behaviour trees for the mario ai competition using grammatical evolution, 2011a.

D. Perez, M. Nicolau, M. O'Neill, and A. Brabazon. Reactiveness and navigation in computer games: Different needs, different approaches, 2011b.

J. Porta, E. del Rosal, and I. Ahumanda. Design and development of iberia: a corpus of scientific spanish. *CORPORA*, 6(2):145–158, 2011.

G. Păun. P systems with active membranes: attacking np-complete problems. *Automata, Languages and Combinatorics*, 6 (1):7590, 2001.

Gh. Păun. Computing with membranes. *Journal of Computer and System Sciences*, 61:108–143, 2000.

Gh. Păun, G. Rozenberg, and A. Salomaa. *DNA Computing. New Computing Paradigms*. Berlin, Springer, 1998.

C. H. Rankin and B. S. Broster. Factors affecting habituation and recovery from habituation in the nematode caenorhabditis-elegans. *Behavioral Neuroscience*, 106(2):239–249, 1992.

C. H. Rankin, C. D. O. Beck, and C. M. Chiba. Caenorhabditis-elegansa new model system for the study of learning and memory. *Behavioral Brain Research*, 37(1): 89–92, 1990.

J. Reddin, J. McDermott, and M. O'Neill. Elevated pitch: Automated grammatical evolution of short compositions, 2009.

R. A. Rescorla and A. R. Wagner. *Classical conditioning II: Current research and theory*, pages 64–99. Nueva York: Appleton-Century-Crofts, 1972.

J. L. Risco-Martin, J. M. Colmenar, D. Atienza, and J. I. Hidalgo. Simulation of high-performance memory allocators. *Microprocessors and Microsystems*, 35(8): 755–765, November 2011.

E. Rodrigues and A. Pozo. Grammar-guided genetic programming and automatically defined functions. *Advances in Artificial Intelligence, Proceedings*, 2507: 324–333, 2002.

C. Ryan, J. Collins, and M. O'Neill. Grammatical evolution: Evolving programs for an arbitrary language. In W. Banzhaf, R. Poli, M. Schoenauer, and T. C. Fogarty, editors, *Proceedings of the First European Workshop on Genetic Programming*, volume 1391 of LNCS, pages 83–95. Paris. Springer-Verlag, 1998.

C. Ryan, M. Keijzer, and M. Nicolau. On the avoidance of fruitless wraps in grammatical evolution. *Genetic and Evolutionary Computation GECCO 2003, PT II, Proceedings*, 2724:1752–1763, 2003.

N. A. Schmajuk, Y. W. Lam, and J. A. Gray. Latent inhibition: A neural network approach. *Journal of Experimental Psychology-Animal Behavior Processes*, 22 (3):321–349, 1996.

S. Seifert and I. Fischer. *Parsing String Generating Hypergraph Grammars*. Springer, 2004.

S. Sen and J. A. Clark. Evolutionary computation techniques for intrusion detection in mobile ad hoc networks. *Computer Networks*, 55(15):3441–3457, October 2011.

J. M. R. Siles, M. D. Echeandia, and A. O. de la Puente. Towards the automatic programming of h systems: jhsys, a java h system simulator, 2010.

B. F. Skinner. *Science and human behabior*. New York: The free press., 1965.

O. Smart, I. G. Tsoulos, D. Gavrilis, and G. Georgoulas. Grammatical evolution for features of epileptic oscillations in clinical intracranial electroencephalograms. *Expert Systems With Applications*, 38(8):9991–9999, August 2011.

J. E. R. Staddon and J. J. Higa. Multiple time scales in simple habituation. *Psychological Review*, 103(4):720–733, 1996.

J. C. Stanley. Computer-simulation of a model of habituation. *Nature*, 261(5556): 146–148, 1976.

R. F. Thompson and W. A. Spencer. Habituation: a model phenomenon for the study of neuronal substrates of behavior. *Psychological Review*, 73:16–43, 1966.

J. Timmis and P. J. Bentley. *Proceedings of the 1st International Conference on Artificial Inmune Systems*. UKC, 2002.

I. Tsoulos, D. Gavrilis, and E. Glavas. Neural network construction and training using grammatical evolution. *Neurocomputing*, 72(1-3):269–277, December 2008.

I. G. Tsoulos and I. E. Lagaris. Genetically controlled random search: a global optimization method for continuous multidimensional functions. *Computer Physics Communications*, 174(2):152–159, 2006.

I. G. Tsoulos, D. Gavrilis, and E. Dermatas. Gdf: A tool for function estimation through grammatical evolution. *Computer Physics Communications*, 174(7):555–559, 2006.

S. D. Turner, S. M. Dudek, and M. D. Ritchie. Grammatical evolution of neural networks for discovering epistasis among quantitative trait loci, 2010.

M. Volk. *Introduction to Natural Language Processing,*. Course CMSC 723 / LING 645 in the Stockholm University, Sweden., 2004.

A. R. Wagner. *Information processing in animals: Memory mechanisms*, pages 5–47. Hillsdale, NJ: Erlbaum., 1981.

D. L. Wang. A neural model of synaptic plasticity underlying short-term and long-term habituation. *Adaptive Behavior*, 2:111–129, 1994.

D. A. Watt and O. L. Madsen. Extended attribute grammars. Technical Report 10, University of Glasgow, July 1977.

W. Weaver. *Translation, Machine Translation of Languages: Fourteen Essays*. 1955.

P. A. Whigham. Grammatically-based genetic programming. In Rosca J. P., editor, *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications*, pages 33–41. Tahoe City, California, USA, 1995.

G. Zhang, B.and Yang and W. Zheng. Jcluster: an efficient java parallel environment on a large-scale heterogeneous cluster: Research articles, October 2006. ISSN 1532-0626.

A. Zollmann and A. Venugopal. Syntax augmented machine translation via chart parsing. In *Proccedings of the Workshop on Statistic Machine Translation*. HLT/NAACL, New York, June. 2006.