CENTRO DE INVESTIGACIÓN CIENTÍFICA Y DE EDUCACIÓN SUPERIOR DE ENSENADA, BAJA CALIFORNIA

PROGRAMA DE POSGRADO EN CIENCIAS EN CIENCIAS DE LA COMPUTACIÓN

Gramáticas de Comportamiento Robótico

Tesis

para cubrir parcialmente los requisitos necesarios para obtener el grado de Maestro en Ciencias

Presenta:

Miguel Alejandro Jiménez García

Ensenada, Baja California, México

2015

Resumen de la tesis que presenta Miguel Alejandro Jiménez García como requisito parcial para la obtención del grado de Maestro en Ciencias en Ciencias de la Computación.

Gramáticas de Comportamiento Robótico

Resumen elaborado por:

Miguel Alejandro Jiménez García

Dentro del marco de los avances de la inteligencia artificial, el reto del diseño e implementación de modelos de computación con los cuales se pueda desarrollar mecanismos que definan entidades “inteligentes” siempre ha constituido un desafió importante en la búsqueda de la emulación del comportamiento humano.

El desarrollo de agentes robóticos autónomos, los cuales sean capaces de percibir su medio ambiente y puedan comportarse, de forma óptima y eficiente, de acuerdo a lo sensado con el fin de alcanzar un objetivo específico; es un problema que hasta la fecha no se ha resuelto en su totalidad. Diversos esfuerzos y trabajos se han realizado para tratar de dar autonomia a agentes robóticos, lo cual en parte se ha logrado, pero el camino para alcanzar resultados satisfactorios sigue siendo largo.

Éste trabajo presentará un sistema fundamentado en la teoría de lenguajes formales, más específicamente en los lenguajes del tipo independientes de contexto, lo cual tiene como propósito el utilizar la notación, conocida como gramáticas independientes de contexto para representar y generar comportamientos basados en eventos y acciones. Las gramáticas independientes de contexto son una notación natural y recursiva, las cuales a diferencia de otros tipos de notaciones presentan un balance de poder descriptivo y tratabilidad, motivo por el cual se seleccionaron como base para el sistema a desarrollar en este trabajo.

En esta tesis se presentará el modelo del sistema diseñado para un agente, el cual tendrá como núcleo lo que llamaremos ’gramática de comportamientos’, la cual es una grámatica independiente de contexto que sigue ciertas reglas para su construcción y ademas definirá un lenguaje de comportamientos, el cual a su vez será la representación del comportamiento del agente.

Palabras Clave: **Gramáticas y lenguajes formales, Comportamiento robótico, Aprendizaje por refuerzo, Algoritmos Genéticos**

Abstract of the thesis presented by Miguel Alejandro Jiménez García as a partial requirement to obtain the Master of Science degree in Master in Sciences in Computer Science.

Robotic Behavior's Grammars

Abstract by:

Miguel Alejandro Jiménez García

In what concerns to the field of artificial intelligence and its advancements, one aspect that is very important in the search for the emulation of human behavior is the challenge of designing and developing computer models, which can be used as building blocks to define intelligent entities.

The development of autonomous robotic agents that are able to percive the surrounding environment and, in accordance to sensed information, act in an optimum and efficient manner is considered a challenging unsolved problem; at least at the time of writing this thesis. Current works with such characteristics have the aim of studying different places for specific scenarios and conditions, but the road to the complete autonomous agent is still far away. In fact, even in the nearby future, a full-fledged robot like the ones that appear in science fiction will not make an appearance.

A system based on formal languages devoted to the representation and execution of robot behaviors will be presented in this thesis. The selected languages are the ones known as context-free languages, and the core of the system lies in a particular notation: "the context-free grammars", which are a natural and recursive notation for this type of languages. Context-free grammars provide a good balance between descriptive power and treatability, which was one of the main reasons for the selection of these kind of grammars for this work.

This thesis will present the model for the above-mentioned system, whose core element will be named as "behavior grammar". A behavior grammar is a context-free grammar that maps actuators and sensors (actions and events) to the grammar structure and for that reason it follows specific rules in their application. The languages defined by the behavior grammars will represent an agent behavior and we will call them behavioral languages.

Keywords: **Formal grammar and languajes, Robotic behavior, Reinforcement learning, Genetic Algorithms**

Dedicatoria

A mi familia que siempre me ha apoyado.

Por mas lejos que nos encontremos seguiremos compartiendo un lazo muy especial.

Agradecimientos

A mis directores de tesis, el M.C. José Luis Briceño Cervantes y el Dr. Gustavo Olague Caballero por su guía, paciencia, y de quienes recibí valiosos conocimientos.

A los miembros de mi comité de tesis, el Dr. Hugo Homero Hidalgo Silva y el Dr. Roberto Conte Galván por sus valiosos comentarios y sugerencias.

A mis amigos y compañeros de CICESE, por su amistad, apoyo y con los que pase agradables y divertidos momentos.

A todos aquellos que de forma directa o indirecta me apoyaron en esta etapa, entre los que se encuentran mis profesores de CICESE y de CETYS Universidad.

Al Centro de Investigación Científica y de Educación Superior de Ensenada.

Al Consejo Nacional de Ciencia y Tecnología (CONACyT) por brindarme el apoyo económico para realizar mis estudios de maestría.

A mi familia por su apoyo y paciencia.

xU rre sphaela nLYAmAlU nafa qraffa /.

Tabla de contenido

Página

Tabla de contenido (continuación)

Tabla de contenido (continuación)

Tabla de contenido (continuación)

Lista de figuras

Figura Página

Tesis defendida por

Miguel Alejandro Jiménez García

y aprobada por el siguiente comité

M.C. José Luis Briseño Cervantes
Codirector del Comité

Dr. Gustavo Olague Caballero
Codirector del Comité

Dr. Hugo Homero Hidalgo Silva
Miembro del Comité

Dr. Roberto Conte Galván
Miembro del Comité

Dra. Ana Isabel Martínez García
Coordinadora del Programa de Posgrado en Ciencias de la Computación

Dr. Jesús Favela Vara
Director de Estudios de Posgrado

Junio, 2015

Capítulo 1. Introducción

1.1. Motivación

El lograr que una máquina actúe de forma independiente es uno de los grandes sueños de la humanidad y hasta el momento sigue presentando un gran desafío. El hacer que una máquina “piense” y actúe de forma autónoma no es una tarea trivial, y menos aun que aprenda por si sola. Aunque en la actualidad existen algoritmos con los cuales se puede dotar a las máquinas con algo de autonomía y pueden ayudar a que estas dejen de ser simples, el sueño que se ve en los trabajos de ciencia ficción, como aquel robot con inteligencia y comportamientos que se asemejan a los de un ser humano, por lo menos en este mundo, sigue siendo eso, un sueño.

1.2. Justificación

El desarrollo de agentes robóticos que posean capacidades para realizar diversas tareas y comportarse de acuerdo a reglas establecidas en su medio es un problema que sigue siendo estudiado y abordado desde distintas perspectivas. Diversas técnicas de inteligencia artificial y de aprendizaje de máquina siguen siendo los principales medios por los cuales se busca que los agentes robóticos obtengan mayor autonomía, aunque también existen aquellos que utilizan otra variedad de herramientas para abordar éste reto.

Los lenguajes independientes de contexto poseen un tipo de notación natural, jerárquica y recursiva que son las gramáticas independientes de contexto. Debido a las propiedades de estas gramáticas son lo suficientemente poderosas como herramientas para definir lenguajes y a la vez son hasta cierto grado tratables. Éstas son algunas de las características que hacen que los lenguajes, y gramáticas, independientes de contexto sean llamativas como herramientas para distintos ámbitos, siendo la descripción de lenguajes de programación y el diseño de compiladores dos ejemplos que no se pueden quedar sin mención.

Al hacer uso de gramáticas independientes de contexto como un medio para la descripción y generación de comportamientos, se pueden lograr patrones de tales compor-

tamientos que incorporen las distintas propiedades de éste tipo de gramáticas. Si una gramática describe un comportamiento, entonces se puede decir que se tiene un lenguaje de dicho comportamiento, es decir un lenguaje de comportamientos definido por una gramática de comportamientos.

1.3. Objetivos

1.3.1. Objetivo general

El objetivo de este trabajo es desarrollar un sistema el cual mediante la utilización de teoría de lenguajes y gramáticas formales, represente y genere comportamientos sencillos para agente artificiales.

1.3.2. Objetivos específicos

- Utilizar lenguajes formales para describir comportamientos basados en eventos y acciones.
- Diseñar el tipo de gramáticas que definirán los lenguajes de comportamientos.
- Implementar la ejecución de comportamientos mediante el proceso de derivación de gramáticas.
- Demostrar mediante la implementación de un agente basado en una gramática de comportamientos:
  - Para el problema de la hormiga artificial, la solución a la instancia conocida como el trayecto de Santa Fe.
  - La funcionalidad del sistema utilizando el motor Unity3D para la implementación.
- Verificar la posibilidad de utilizar gramáticas de comportamientos de robots en:
  - NAO de Aldebaran.
  - Lego Mindstorm NXT 2.0.

1.4. Metodología

A continuación se describe las distintas fases de desarrollo seguidas para y durante el desarrollo de este trabajo:

1. Estudio de lenguajes: Revisión de teoría y fundamentos de lenguajes formales para localizar una representación adecuada para los comportamientos que se buscan representar.
2. Prototipos: El diseño de prototipos que sirvan como base para el trabajo posterior y que ayuden a verificar la validez de la utilización de lenguajes para representar y generar comportamientos.
3. Definición formal de la gramática de comportamientos: Definir de manera formal los lenguajes de comportamientos por medio de lo que llamaremos gramáticas de comportamientos.
4. Estudio de técnicas de aprendizaje: Revisión de técnicas de inteligencia artificial que se puedan utilizar en conjunto con las gramáticas de comportamientos.
5. Experimentos: Implementación de varios agentes basados en gramáticas de comportamientos.
6. Análisis de resultados y conclusiones: Análisis y conclusiones de los resultados obtenidos en los experimentos.

1.5. Estructura del documento

Este texto se divide en varios capítulos, los cuales se describen brevemente a continuación.

Capítulo 2: Se presenta aquí lo relacionado con las dos áreas en donde se sitúa este trabajo, las cuales son estudios de comportamientos en agentes artificiales y teoría de lenguajes aplicada en el área de ciencias de la computación.

Capítulo 3: Se introducen las bases y fundamentos de teoría de lenguajes, los cuales ayudarán a facilitar la comprensión del trabajo descrito en este texto, es decir, las gramáticas de comportamientos.

Capítulo 4: Se presenta el marco teórico de las técnicas de inteligencia artificial que se utilizan para asistir a la solución del trayecto de Santa Fe.

Capítulo 5: Se presentan la descripción y esquemas del modelo seguido para el desarrollo del sistema.

Capítulo 6: Este capítulo describe gramáticas de comportamientos, su definición formal, componentes, estructura, restricciones, así como comentarios para su diseño e implementación.

Capítulo 7: La implementación de distintas gramáticas experimentales son presentadas en este capítulo, es decir, se dan a conocer los experimentos realizados así como sus resultados.

Capítulo 8: Se presentan las conclusiones y el trabajo futuro.

Capítulo 2. Trabajo relacionado

El estudio de cómo generar diversos patrones de comportamiento en agentes artificiales, ya sean robots o agentes simulados, así como el estudio y aplicación de lenguajes formales en el área de computación, son esfuerzos de desarrollo que cuentan con una extensa literatura. En esta sección se presentará un poco de aquellos estudios que están relacionados al presente trabajo. En las secciones siguientes se mostrará por separado un poco del trabajo que se ha hecho en ambas áreas, mientras que en la última sección de este capítulo se revisarán aquellos trabajos, los cuales de forma similar a este trabajo, utilizan una mezcla de ambos.

2.1. Estudios de Comportamientos

Dentro del área de aprendizaje y generación de comportamientos, ya sea para robots u otros agentes artificiales, se han realizado distintos trabajos prometedores que involucran el uso de distintas técnicas y algoritmos, por medio de los cuales se busca superar las limitaciones y restricciones impuestas por el hardware y software utilizados, así como por el problema que se busca resolver. Trabajar con robots no es algo sencillo, es un proceso que consume tiempo y requiere de cuidados especiales, ya que tienden a ser frágiles y caros, por lo que la utilización de simuladores no es algo fuera de lo común, sin embargo, por más bien construidos que estén los simuladores, éstos no son perfectos.

Una gran parte de los estudios en el área están enfocados hacia el desarrollo de agentes que sean capaces de, bajo cierta condiciones, aprender a resolver diversos problemas. Los algoritmos del área de inteligencia artificial son algunos de los más populares en este tipo de trabajos, como aquellos utilizados en las redes neuronales, el aprendizaje por refuerzo, o algoritmos genéticos, por mencionar algunos.

Algunos trabajos buscan el desarrollo de agentes que imiten hasta cierto grado el comportamiento de un humano. Con relación a agentes virtuales, se pueden encontrar trabajos como el de Gamez *et al.* (2013), en el cual implementaron diversos comportamientos básicos en una red neuronal con el fin de obtener un agente que simulara el estar siendo controlado por un ser humano. Con un objetivo similar Geisler (2004) modeló un agente a partir de un muestreo de datos generados por un jugador (humano). Bauckhage

et al. (2007) utilizaron vectores de datos, que obtuvieron a partir de jugadores humanos en ambientes virtuales (vídeo-juegos), combinados con aprendizaje por refuerzo para lograr agentes que aprendieran los comportamientos de los jugadores; también utilizaron un modelado bayesiano y algoritmos de agrupamiento para ayudar a que los agentes tuvieran un mejor entendimiento de las decisiones que siguieron los jugadores de los cuales se aprendió.

Hester *et al.* (2010) y Er y Deng (2005) hacen mención explicita acerca de la dificultad que presenta el tratar de resolver problemas de aprendizaje mediante la utilización de un solo algoritmo, lo que se puede observar no solo en sus trabajos, sino en muchos otros. Hester *et al.* (2010) y Bakker *et al.* (2002) en sus trabajos utilizan y justifican la integración de varias técnicas de aprendizaje en sus agentes; así Hester *et al.* (2010) presentan un trabajo en el cual entrenan a un robot para la ejecución de tiros de penal mediante aprendizaje por refuerzo asistido por árboles de decisión; Bakker *et al.* (2002) utiliza de forma similar aprendizaje por refuerzo pero afirma que para algunos casos es necesario que el agente sea capaz de recordar, por lo cual asiste a su algoritmo de refuerzo con una red neuronal para darle al agente memoria de largo y corto plazo, así como de un mecanismo de extracción de eventos, el cual transforma las lecturas de datos continuos de los sensores del agente en cadenas compactas de eventos discretos.

También en la literatura se pueden encontrar otros trabajos que difieren un poco de los enfoques tradiciones como el de Gu y Su (2006) donde se introduce el concepto de aprendizaje por clasificación, en el cual se considera que el aprendizaje de comportamientos en un robot como un proceso de clasificación, y utilizan lo que llaman red neuronal ART (Adaptative Resonace Theory) para el aprendizaje de comportamientos, el cual esta basado en la descomposición y combinación de imágenes de movimientos de articulaciones. Ogura *et al.* (2003) trabaja con lo que llaman redes de comportamientos (behavior network), las cuales están compuestas por nodos, que representan estados o valores de sensores, y por arcos que representan movimientos. En dicho trabajo se busca la automatización de estas redes mediante la integración de nodos por medio de algoritmos de agrupamiento y generación de aristas por medio de algoritmos genéticos.

Por otro lado, están aquellos trabajos que de similar manera trabajan con algunas de

las técnicas ya mencionadas de inteligencia artificial, pero que al mismo tiempo poseen un enfoque interactivo, ya que, el proceso de diseño, aprendizaje y desarrollo de comportamientos involucra de forma más activa a los seres humanos. Atkeson *et al.* (2000) propone una metodología en la cual se busca que, por medio de aprendizaje por demostración y aprendizaje por refuerzo, los robots aprendan habilidades motoras, donde los robots aprendan por medio de demostración y observación, y refinen lo aprendido por medio de aprendizaje por refuerzo.

Trabajos como los de Stiefelhagen *et al.* (2004), Csapo *et al.* (2012) y Vircikova *et al.* (2011) están más enfocados en la integración de diversos módulos, cada uno de los cuales realiza funciones diferentes, con el fin de desarrollar agentes que sean capaces de interactuar de forma autónoma con los seres humanos; enfocándose Stiefelhagen *et al.* (2004) en la integración de módulos y la demostración y medida del uso de dichas tecnologías para la interacción humano-robot; mientras que Csapo *et al.* (2012) tienen un enfoque dirigido a la conversación y uso de módulos de ayuda para ello. Otros trabajos de agentes interactivos incluyen los de Nikolaidis y Shah (2013) y Vircikova *et al.* (2011). Nikolaidis y Shah (2013) por ejemplo, realizan un entrenamiento cruzado, es decir se intercambian roles de manera iterativa para un agente artificial y para un humano involucrado, los cuales aprenderán a desempeñar una tarea en conjunto, algo similar al entrenamiento de un equipo de seres humanos. En el caso de Vircikova *et al.* (2011), existe un maestro humano que por medio de ejemplos enseña un comportamiento (en el estudio presentado la tarea fue el baile) y por medio de un proceso de computación evolutiva, se evalúa y optimiza (dicho comportamiento) de manera iterativa de acuerdo a los gustos del evaluador.

Farchy *et al.* (2013) por otra parte propone algo bastante interesante que busca mejorar la precision de los simuladores. El método propuesto involucra a ambos, robot y simulador, y mediante un proceso iterativo se busca conforme se van obteniendo resultados en el robot y en el simulador, ir ajustando este último para aumentar su precisión.

Otras áreas del conocimiento también se ven involucradas en ocasiones en estos temas de agentes artificiales, en especial aquellas que como Anderson *et al.* (2004) buscan de alguna forma u otra ayudar a resolver los enigmas de los proceso de la mente median-

te el desarrollo y diseño de arquitecturas basadas en teorías de la mente. Así como se pueden encontrar arquitecturas basadas en neurociencias, también están aquellas que están diseñadas para ayudar a facilitar el diseño e implementación de comportamientos para fines específicos, como lo es BICA de Martın *et al.* (2010), cuya arquitectura fue diseñada principalmente para desarrollar comportamientos para agentes que participan en la competencia RoboCup. También se pueden encontrar trabajos como el de Dongshu *et al.* (2011) en el que se utilizan controladores para la ejecución de ciertos comportamientos junto con un controlador extra para la integración de los demás.

Entre otros trabajos que utilizan programación genética para encontrar comportamientos se puede mencionar la solución que se presenta en Koza (1992) para el problema de la hormiga artificial, en el cual se evoluciona un programa que guía un agente (hormiga artificial) por un trayecto de comida. En la sección de experimentos se revisará como resolver este problema por el método que se propone en este trabajo: la gramática de comportamientos.

2.2. Estudios de gramáticas y lenguajes

Entre los diversos usos que se le pueden dar a las gramáticas se encuentra la descripción de lenguajes de programación, así como el desarrollo de analizadores léxicos.

Algunas de las aplicaciones más comunes de los lenguajes formales así como de sus equivalentes, autómatas, gramáticas y máquinas de turing, suelen ser muy utilizadas en la vida cotidiana y para aquellos que desconocen las bases de esta área la forma en que operan puede pasar totalmente desapercibida. La búsqueda de palabras de texto es un ámbito en el que se suelen utilizar autómatas aunque se suelen combinar con otras técnicas para mejorar la eficacia; las expresiones regulares se suelen utilizar para la búsqueda de patrones en textos, así como para el desarrollo de analizadores léxicos para lenguajes de programación, los cuales exploran una cadena de entrada y reconocen agrupaciones lógicas y palabras clave.

Las gramáticas juegan un papel muy importante en el área de computación, ya que son utilizadas para describir lenguajes de programación así como para desarrollar analizadores sintácticos, los cuales validan que los programas escritos sean válidos. Los

analizadores léxicos y sintácticos son dos de los componentes que forman parte de un compilador, es decir, el traductor que convierte un programa fuente a lenguaje de máquina (0s y 1s) para que pueda ser ejecutado por un computador (Hopcroft *et al.*, 2002).

Por otra parte, se pueden encontrar trabajos relacionados con el uso de gramáticas para representar no solo lenguajes conformados por caracteres y números. Así, un ejemplo de ello es la gramática de formas (shape grammar) que fue introducida por Stiny (1980) para representar las relaciones de formas (figuras) en el espacio. A su vez la gramática de formas ha sido utilizada en trabajos como el de Riemenschneider *et al.* (2012) para el estudio de representación y reconstrucción de fachadas de edificios.

2.3. Trabajos que relacionan comportamientos y lenguajes

En cuanto a trabajos que relacionan lenguajes formales y el como utilizar estos para generar distintos tipos de comportamientos en agentes se pueden encontrar algunos muy interesantes. Uno de estos trabajos es el de Manikonda *et al.* (1995) cuyo lenguaje de descripción de movimiento MDLe (motion description language) es un lenguaje independiente de contexto, que fue demostrado por Hristu-Varsakelis *et al.* (2003) donde se provee una base para la programación de robots mediante el uso de comportamientos, los cuales están basados en máquinas de estado cinemático (kinematic state machines) y la interacción de estas con la información recibida por parte de los sensores del agente. Las máquinas de estado utilizadas en la generación de los comportamientos están modeladas a base de ecuaciones diferenciales. Los varios factores de este enfoque permiten una base matemática para su estudio. El MDLe tiene comportamientos atómicos y compuestos. El MDLe de Manikonda *et al.* (1995) es una extension del lenguaje de descripción de movimiento (MDL) de Brockett (1988).

Otro trabajo bastante interesante es la gramática de movimiento (motion grammar) que presenta Dantam *et al.* (2011) la cual es una extension de una gramática libre de contexto, ya que se añaden elementos adicionales que ayudan a tratar con diversos elementos como estados y elementos dinámicos del ambiente. Esta gramática de movimiento representa una política para controlar sistemas robóticos. Un sistema basado en esta gramática realiza un análisis sintáctico en tiempo real (online) de muestras las cuales

están definidas por los distintos estados del sistema y por la discretización de las lecturas de los sensores del agente. La función de análisis sintáctico se realiza por medio de lo que llaman analizador sintáctico de movimiento, que recibe, lee las instancias y ejecuta las acciones correspondientes. Una gramática de movimientos representa el lenguaje de las lecturas de los sensores. En Dantam y Stilman (2012) y Dantam y Stilman (2013) se pueden observar el seguimiento que se le da a la gramática de movimiento.

Capítulo 3. Teoría de lenguajes

En esta sección se presentara la teoría concerniente a lenguajes y gramáticas formales, de tal modo que ayuden al lector a tener un mejor entendimiento del modelo que se propone en el presente trabajo. Si se desea una consulta más a profundidad de los temas abarcados en este capítulo favor de consultar Hopcroft *et al.* (2002).

3.1. Alfabetos

Un alfabeto, comúnmente representado por Σ, es un conjunto de símbolos, es finito y no vacío.

Algunos de los alfabetos más comunes:

- $\Sigma = \{0, 1\}$ alfabeto binario.
- $\Sigma = \{A, B, C, \ldots, Z\}$ Alfabeto de letras mayúsculas.

3.2. Cadenas

Símbolos pertenecientes a un alfabeto ordenados en secuencia.

3.2.1. Cadena vacía

Cadena que está formada por cero símbolos. Se puede formar a partir de cualquier alfabeto y se representa por medio de ϵ.

3.2.2. Longitud de cadena

Total de posiciones en una cadena que están siendo ocupadas por símbolos. Un ejemplo puede ser la cadena 1001001001 que tiene una longitud de 10. Suele decirse que el número de símbolos en la cadena es la longitud pero esto no es del todo correcto. La cadena 1001001001 contiene 2 símbolos (0 y 1) los que ocupan en total diez posiciones por lo que la longitud es 10.

La notación que se utiliza para indicar la longitud de una cadena w es $|w|$. Continuando con el ejemplo anterior, $|1001001001| = 10$. Así mismo para una cadena vacía $|\epsilon| = 0$.

3.2.3. Potencias de un alfabeto

Para un alfabeto Σ es posible expresar el conjunto de todas las cadenas de una determinada longitud mediante notación exponencial. Se define Σ^k como el conjunto de todas las cadenas cuya longitud es k y que están formadas solo por símbolos que pertenecen a Σ.

El conjunto de todas las posibles cadenas que se pueden formar a partir de un alfabeto Σ se define como Σ^*. Cuando no se desea incluir la cadena vacía dentro del conjunto de cadenas (no vacías) del alfabeto Σ se escribe como Σ^+, tal que

- $\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup ...$

  y por ende

- $\Sigma^* = \Sigma^+ \cup \{\epsilon\}$.

3.2.4. Concatenación de cadenas

Suponga que se tienen dos cadenas y y z entonces la concatenación de estas se denota como yz, esto es una cadena de longitud $|y|$ + $|z|$ donde las primeras $|y|$ posiciones están ocupadas por los símbolos que ocupan dichas posiciones en la cadena y, ademas las posiciones que se encuentran a continuación contienen de manera similar los símbolos que ocupan las posiciones de la cadena z (ordenados de la misma forma).

Otra forma de expresarlo:

Sea y una cadena compuesta por i posiciones ocupadas por símbolos

$y = a_1 a_2 a_3 \dots a_i$

Sea z una cadena compuesta por j posiciones ocupadas por símbolos

$z = b_1 b_2 b_3 \dots b_j$

entonces, al concatenar y y z

$yz = a_1a_2a_3 \ldots a_ib_1b_2b_3 \ldots b_j$

donde la longitud de la cadena yz esta dada por

$|yz| = i + j$

3.3. Convenios y nomenclaturas

Usualmente se utilizan las primeras letras del alfabeto (minúsculas) así como dígitos para representar símbolos. Para representar cadenas se suelen utilizar las últimas letras del alfabeto (minúsculas), comúnmente w, x, y, z.

3.4. Lenguajes

Un lenguaje es un conjunto de cadenas, válidas bajo ciertos criterios, formadas por los símbolos de algún alfabeto Σ. Si Σ es el alfabeto utilizado y L está dentro de Σ^*, se dice que L es un lenguaje de Σ. Los lenguajes que usamos para el habla y comunicación se pueden también describir de forma similar, como conjuntos de palabras válidas formadas por cadenas de un alfabeto al que pertenecen las letras utilizadas. Para un lenguaje de programación, un programa es una cadena construida por los símbolos de dicho lenguaje. Por otra parte, al tratar con autómatas o gramáticas es común encontrar otra variedad de lenguajes.

- $\{\epsilon\}$ Lenguaje que solo contiene la cadena vacía.
- El lenguaje de palíndromos. Un palíndromo es aquella cadena que se lee de igual forma en cualquier sentido.
- $\{0^n1^n | n \geq 1\}$ El conjunto de cadenas tal que estén formadas por uno o más ceros seguidos de la misma cantidad de unos.
- $\{0^n1^n2^n | n \geq 1\}$ Conjunto de todas las cadenas de los dígitos 012 tal que aparecen el mismo número de veces, por ejemplo, 012, 001122, 000011112222.

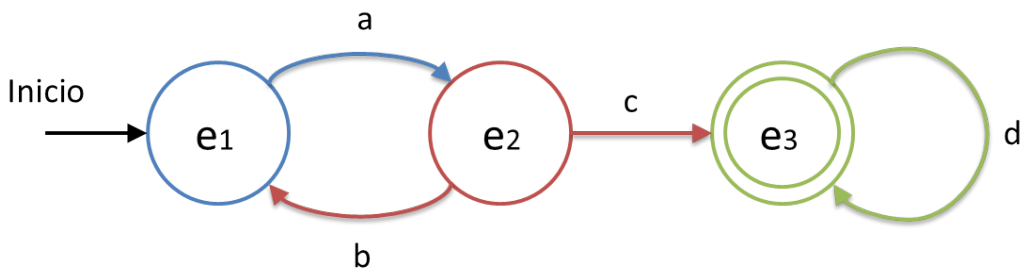


Figura 1: Autómata genérico de 3 estados.

3.5. Lenguajes regulares

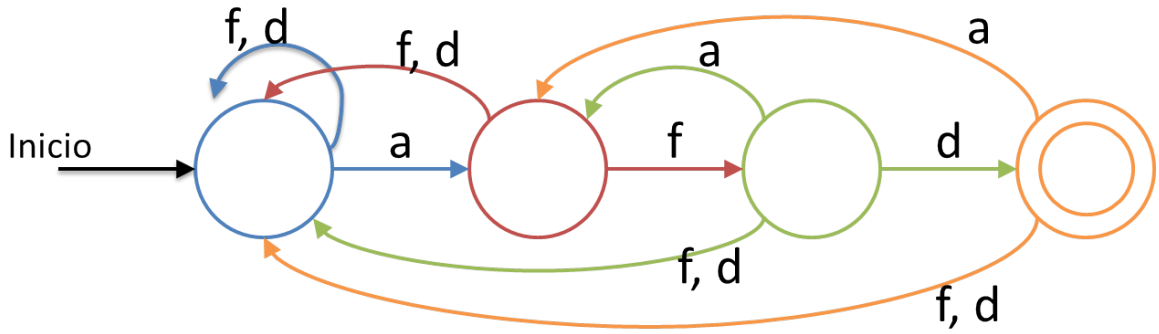
Son aquellos lenguajes que son definidos por autómatas finitos y expresiones regulares.

3.5.1. Autómatas finitos (AF)

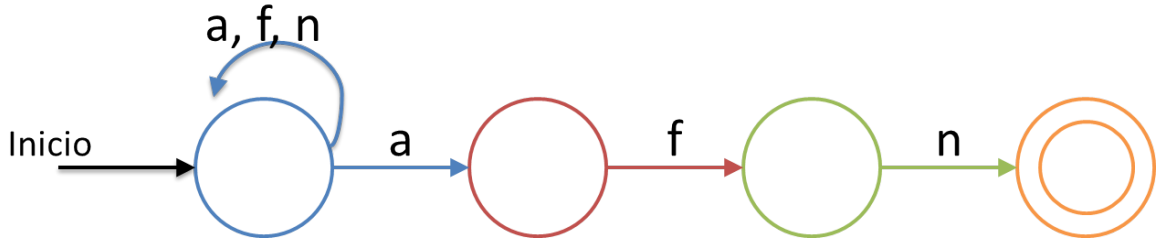
Son una de las formas para la representación de lenguajes regulares. Formados por un conjunto de estados y transiciones para los cuales las entradas externas definen el flujo que se sigue. En la figura 1 se puede observar un autómata genérico de tres estados, donde los círculos representan los estados, los arcos las transiciones entre estados y las etiquetas (a,b,c,d) en dichos arcos son las entradas; el estado más a la izquierda señalado por la flecha etiquetado con la palabra inicio se denomina “estado inicial” y el estado con doble circulo es un “estado final” o “de aceptación”.

Los autómatas aceptan cadenas. Si comenzando por un estado inicial del autómata y tomando las posiciones ocupadas por símbolos de la cadena como entradas, de manera ordenada y una por una, las transiciones de estados llevan a un estado de aceptación, entonces se dice que la cadena es aceptada por el autómata.

Los autómatas finitos pueden dividirse en deterministas (AFD) y no deterministas (AFN). Un autómata es determinista si puede estar en solo un estado en un determinado momento y es no determinista si puede estar en varios estados a la vez. Los autómatas finitos no deterministas a su vez se dividen en autómatas finitos no deterministas con y sin transiciones ϵ (AFN-ϵ). En la figura 2 se pueden observar dos autómatas, uno determinista y uno no determinista, ambos definen lenguajes similares; aceptan todas la



(a) AFD



(b) AFN

Figura 2: Ejemplos de un autómata finito determinista (a) y de un autómata finito no determinista (b). Las entradas de ambos autómatas son los símbolos 'a', 'f' y X. Ambos autómatas aceptan cadenas conformadas por los símbolos mencionados que terminen con la subcadena 'afX'. X = 'd' para el autómata determinista y X = 'n' para el no determinista.

cadenas conformadas por los símbolos 'a', 'f' y X ('d' para el autómata determinista y 'n' para el no determinista), pero que terminen con la subcadena 'afX'.

3.5.2. Expresiones regulares (ER)

Expresiones algebraicas capaces de describir los lenguajes. Los lenguajes descritos por las expresiones regulares son los mismos que aquellos descritos por los autómatas finitos. Los lenguajes que describen son los lenguajes regulares.

3.5.3. Equivalencias entre AF y ER

Aunque la forma de representación sea diferente, las expresiones regulares representan la misma clase de lenguajes que los autómatas finitos. En la figura 3 se puede apreciar el diagrama de las equivalencias entre las distintas notaciones para los lenguajes regulares.

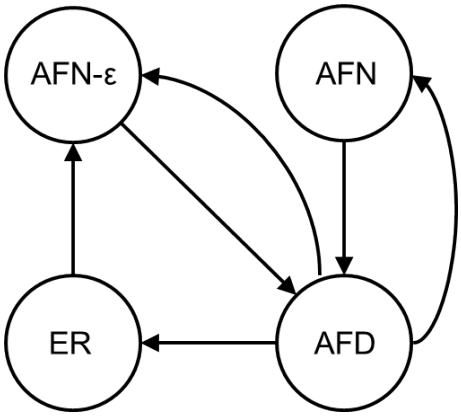


Figura 3: Equivalencias entre autómatas finitos no deterministas (AFN), autómatas finitos deterministas (AFD) y expresiones regulares (ER).

3.5.4. Aplicaciones

Los autómatas finitos suelen ser usados en aplicaciones de búsqueda de texto. Las expresiones regulares son usadas con mucha frecuencia como especificación del componente de los compiladores conocido como analizador léxico, para encontrar unidades sintácticas, es decir las agrupaciones de sub-cadenas que conforman palabras clave, identificadores, etc. Otro uso que tienen las expresiones regulares es la búsqueda de patrones en texto.

3.6. Lenguajes independientes de contexto

Lenguajes que tienen una notación conocida como gramática independiente de contexto. También pueden ser definidos por una notación similar a los autómatas finitos para los lenguajes regulares la cual es el autómata de pila.

3.6.1. Gramáticas independientes de contexto

Son una notación natural y recursiva para los lenguajes independientes de contexto. Llamadas en inglés CFG (Context-Free Grammar).

3.6.1.1. Definición formal

Se representa una gramática independiente de contexto G de la forma

$G = (V, T, P, S)$

- V – Es un conjunto finito de variables, cada una representa un conjunto de cadenas o lenguajes. Comúnmente llamadas símbolos no terminales.
- T – Alfabeto de los símbolos que conformarán las cadenas del lenguaje que definirá la gramática. Comúnmente llamados símbolos terminales. También se suele utilizar la letra Σ para denotar el conjunto de símbolos terminales tal que $G = (V, \Sigma, P, S)$.
- P – Producciones o reglas que denotan la propiedad recursiva del lenguaje que define la gramática. Cada producción está formada por:
  - a) Cabeza de la producción. Es una variable que define la regla.
  - b) Símbolo de producción $\rightarrow$.
  - c) Cuerpo de la producción. Denota una posible manera de formar cadenas para el lenguaje que define la cabeza de la producción, está formada por cero o más símbolos terminales y variables.
- S – Símbolo inicial. Variable que representará el lenguaje que define la gramática.

Ejemplo:

A continuación se muestra una gramática independiente de contexto cuyo lenguaje contiene solamente la palabra ”grammar”:

Sea $\mathbf{G} = (V, \Sigma, P, S)$

$\mathbf{V} = \{S, R, A, M\}$ $\mathbf{\Sigma} = \{g, r, a, m\}$ $\mathbf{S} = \{S\}$

$\mathbf{P} = \{\ G \rightarrow\ gR,$

$R \rightarrow\ rAr,$

$A \rightarrow\ aMa,$

$M \rightarrow\ mm\}$

3.6.1.2. Notación compacta para producciones

Cuando una variable se encuentra en la cabeza de varias producciones se suelen compactar las producciones escribiendo una sola vez la cabeza de la producción y el símbolo de producción, seguidos por todos los cuerpos de dichas reglas separados por barras verticales.

Ejemplo:

Considérese las siguientes reglas de producción para la variable X

$X \rightarrow A$

$X \rightarrow B$

$X \rightarrow C$

$X \rightarrow \epsilon$

las cuales se pueden reescribir en notación compacta de la siguiente forma:

$X \rightarrow A|B|C|\epsilon$

3.6.1.3. Derivación

Proceso mediante el cual se verifica si una cadena pertenece al lenguaje que define la gramática.

Primeramente al realizar una derivación se expande la variable correspondiente al símbolo inicial, es decir, esta se sustituye por el cuerpo de alguna producción cuya cabeza sea la variable que define el símbolo inicial. Se continúa expandiendo la cadena resultante de manera tal que se van reemplazando las variables restantes hasta que se obtiene una cadena que contenga solo símbolos terminales.

Todas las cadenas que pueden ser obtenidas mediante el proceso de derivación conformarán el lenguaje que define la gramática.

Para representar el proceso de derivación se utilizará un símbolo de relación $\Rightarrow$.

Suponga una gramática independiente de contexto $G = (V, \Sigma, P, S)$. $\alpha A \beta$ es una cadena de variables y terminales, A es una variable del conjunto V, α y β son cadenas de variables (del conjunto V) y símbolos terminales (del conjunto Σ). $A \rightarrow \ \gamma$ es una producción del conjunto P. Entonces $\alpha A \beta \Rightarrow \alpha \gamma \beta$.

Ejemplo:

Para la gramática del ejemplo anterior, que define el lenguaje de la palabra ”grammar”:

Considerando las reglas de producción:

$$\mathbf{P} = \{\ G \rightarrow\ gR, \tag{1}$$

$$R \rightarrow\ rAr, \tag{2}$$

$$A \rightarrow\ aMa, \tag{3}$$

$$M \rightarrow\ mm\} \tag{4}$$

Comenzando por el símbolo inicial:

$\mathbf{S}$ - Símbolo inicial.

$g\mathbf{R}$ - Expandiendo el símbolo inicial por medio de la producción (1).

$gr\mathbf{A}r$ - Resultado de aplicar la producción (2).

$gra\mathbf{M}ar$ - Por la producción (3).

$grammar$ - Se termina expandiendo la variable M por medio de la regla (4).

3.6.1.4. Formas de sentencia

Son cadenas producto de un proceso de derivación que comenzó por el símbolo inicial, es decir las cadenas formadas por variables y/o terminales a lo largo del proceso de derivación.

Ejemplo:

Las distintas formas de sentencia del proceso de derivación del ejemplo anterior:

$S \Rightarrow\ gR \Rightarrow\ grAr \Rightarrow\ graMar \Rightarrow\ grammar$

En (Hopcroft *et al.*, 2002) se les llama formas sentenciales, pero debido a que “sentencial” no es una palabra válida en el lenguaje español (http://lema.rae.es/drae, 2012), en éste texto se denotarán como “formas de sentencia”.

3.6.1.5. Notación para gramáticas

En este documento se usará la siguiente notación para el manejo de las gramáticas.

1. Las primeras letras del alfabeto en minúsculas representarán símbolos terminales, de la misma forma serán tratados dígitos y algunos otros caracteres como lo son algunos signos de puntuación y paréntesis.
2. Las últimas letras de alfabeto en minúsculas representarán cadenas de terminales.
3. Las primaras letras del alfabeto en mayúsculas representarán variables.
4. Las últimas letras mayúsculas del alfabeto representaran variables o terminales.
5. Las letras griegas en minúsculas representaran cadenas de símbolos terminales, variables o combinación de ambos.

3.6.1.6. Derivación izquierda y derecha

En ocasiones es necesario restringir o dar forma a los pasos con los que se realiza una derivación, por lo que se suele utilizar un método en el cual se exige expandir la variable que se encuentre en la posición más hacia algún extremo de la cadena.

Cuando se decide ir expandiendo la variable que se encuentre más hacia la izquierda de la cadena se denominará el proceso como derivación más a la izquierda, cuando se exige que sea por la derecha de la cadena, se denominará como una derivación más a la derecha.

Ejemplo:

Considérese la siguiente gramática para operaciones simples y buscando derivar la palabra (cadena de símbolos terminales) "$n + n - n$":

Sea $\mathbf{G} = (V, \Sigma, P, S)$

$\mathbf{V} = \{S\} \qquad \mathbf{\Sigma} = \{n, +, -\} \qquad \mathbf{S} = \{S\}$

$\mathbf{P} = \{\ S \rightarrow\ S + S, \quad S \rightarrow\ S - S, \quad S \rightarrow\ n\}$

Derivación mas a la izquierda :

$$S \Rightarrow\ S + S \Rightarrow\ n + S \Rightarrow\ n + S - S \Rightarrow\ n + n - S \Rightarrow\ n + n - n$$

Derivación mas a la derecha :

$$S \Rightarrow\ S + S \Rightarrow\ S + S - S \Rightarrow\ S + S - n \Rightarrow\ S + n - n \Rightarrow\ n + n - n$$

Derivación sin un orden especifico :

$$S \Rightarrow\ S - S \Rightarrow\ S + S - S \Rightarrow\ S + n - S \Rightarrow\ n + n - S \Rightarrow\ n + n - n$$

3.6.1.7. Lenguaje de una gramática

Para una gramática independiente de contexto $G = (V, \Sigma, P, S)$, el conjunto de cadenas que pueden ser derivadas a partir del símbolo inicial (S), es conocido como el lenguaje de la gramática, también denotado como $L(G)$. Lenguaje independiente de contexto es como se llama al lenguaje de una gramática independiente de contexto.

3.6.1.8. Árboles de derivación

Los árboles de derivación son una forma en la que se puede representar la derivación de las cadenas. En un árbol de derivación se agrupan los símbolos que forman la cadena de manera jerárquica y mostrando la relación entre variables y la expansión de estas en sub-cadenas de terminales y/o variables. Se conoce como resultado del árbol a la concatenación (desde la izquierda) de los símbolos que aparecen en las hojas de este.

Se puede decir que los resultados de un árbol (cadenas) forman el lenguaje de la gramática cuando su resultado está formado solo por símbolos terminales y la raíz esta etiquetada con el símbolo inicial.

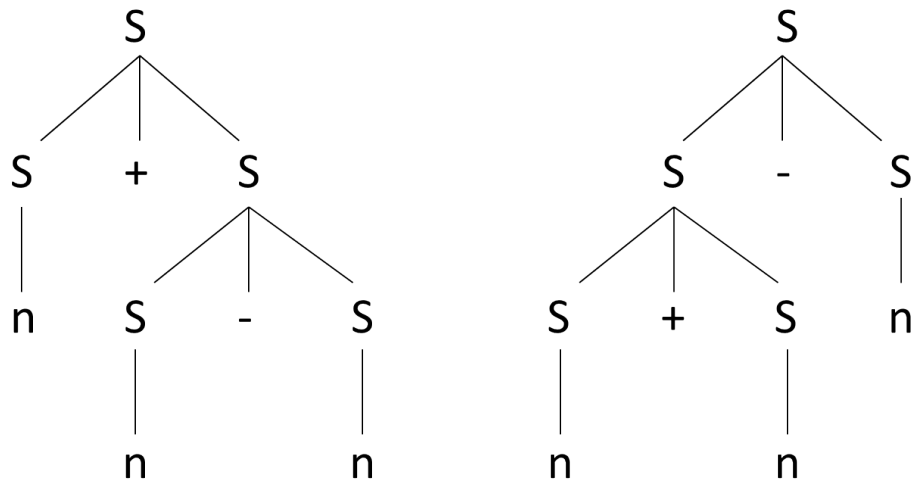


Figura 4: Árboles para los procesos de derivación del ejemplo de la sección 3.6.1.6. (a) Árbol de derivación construido por medio de las derivaciones más a la izquierda y más a la derecha. (b) Árbol para la derivación sin un orden específico.

3.6.1.9. Construcción de árboles de derivación

Los árboles de derivación para una gramática $G = (V, \Sigma, P, S)$ tienen las siguientes características:

- Las etiquetas de los nodos internos son alguna variable del conjunto V.
- Las etiquetas de las hojas pueden ser una variable, un símbolo terminal o el símbolo de la cadena vacía.
- Para algún nodo interno que este etiquetado con una variable A, los nodos hijos de éste deberán estar etiquetados con los símbolos del cuerpo de alguna producción cuya cabeza sea la variable A y estos deberán estar ordenados de izquierda a derecha en el mismo orden en que aparecen los símbolos en el cuerpo de la producción.

En la figura 4 se pueden observar dos árboles de derivación para las derivaciones del ejemplo anterior (Secc. 3.6.1.6), un árbol representa las derivaciones más a la izquierda y más a la derecha, mientras que el otro representa la derivación sin orden un específico.

3.6.1.10. Aplicaciones

La descripción de lenguajes de programación y analizadores léxicos, son algunas de las aplicaciones más comunes. En este trabajo se estudiará el uso de gramáticas para describir y generar comportamientos basados en eventos y acciones.

3.6.1.11. Ambigüedad

Cuando para una cadena de un lenguaje descrito por una gramática existe más de una forma en que ella pueda ser generada, ya sea por derivación o por inferencia recursiva (no abordada en este texto), se dice que dicha gramática posee ambigüedad. En la figura 4 se pueden observar dos arboles distintos los cuales generan la misma palabra: "$n+n-n$"; ésto indica que existe ambigüedad en la gramática utilizada en el ejemplo de la sección anterior.

3.6.2. Autómatas de pila

Los autómatas que se mencionaron anteriormente sirven para representar lenguajes regulares y verificar/comprobar la pertenencia de cadenas para dichos lenguajes. De igual forma existen autómatas que desempeñan funciones similares para lenguajes independientes de contexto, los cuales se conocen como autómatas de pila de datos. Los autómatas de pila son autómatas finitos no deterministas con transiciones ϵ a los cuales se les agrega una pila la cual puede ser leída y a la cual se pueden agregar y retirar elementos; estos elementos siguen un patrón: el último que entra es el primero en salir (o "last in first out" en inglés).

Los autómatas de pila pueden definir el lenguaje aceptado por medio de dos métodos. El primero, similar a los autómatas finitos, es aceptación por estado final; el segundo método es aceptación por pila vacía.

3.6.3. Formas normales

3.6.3.1. Forma normal de Chomsky

Se dice que una gramática $G = (V, \Sigma, P, S)$ está en forma normal de Chomsky si las producciones del conjunto P siguen uno de los patrones:

$A \to BC$

ó

$A \to a$,

donde A, B y C son variables y a es un terminal.

Todo lenguaje independiente de contexto que no sea vacío y sin ϵ puede describirse por una gramática en forma normal de Chomsky. Noam Chomsky demostró que toda gramática puede ser puesta en forma normal de Chomsky.

3.6.3.2. Forma normal de Greibach

Propuesta por Sheila Greibach, esta es otra forma normal para gramáticas independientes de contexto. En esta forma las producciones de la gramática tienen la forma:

$A \to \ a\alpha$

donde A es una variable, a es un terminal, y α es una cadena de cero o más variables.

Convertir una gramática a forma normal de Greibach no es una tarea trivial. (Hopcroft *et al.*, 2002)

Capítulo 4. Teoría de Inteligencia Artificial

A continuación se presentan conceptos, técnicas y algoritmos de inteligencia artificial y aprendizaje de máquina, de tal modo que ayuden al lector a lograr un mejor entendimiento de los siguientes capítulos que conciernen al modelo presentado así como a los experimentos realizados, los cuales se asisten por las técnicas presentadas en esta sección. En (Mitchell, 1997) y (Russell y Norvig, 2009) se puede encontrar información más extensa y detallada sobre los temas que se tratarán en esta sección.

4.1. Aprendizaje por refuerzo

4.1.1. Introducción

Aprendizaje por refuerzo (“Reinforcement Learning” en inglés, de aquí en adelante se abreviará en ocasiones como RL) es una técnica de aprendizaje de máquina que se enfoca en el aprendizaje de un agente y se busca que este interactúe de una forma óptima en el ambiente en el que se encuentra. Muchas de las técnicas de aprendizaje utilizan conjuntos de ejemplos como base para el aprendizaje pero esto no es factible en todos los casos.

El aprendizaje por refuerzo se basa en la idea de un agente autónomo que se encuentra en un ambiente con el cual interactúa. El agente tiene la capacidad de realizar acciones, las cuales pueden cambiar el estado actual en que se encuentra. Para un problema determinado lo que se busca con RL es el encontrar la secuencia de acciones que lleven a un agente a un estado satisfactorio, es decir, a cumplir un objetivo específico en el ambiente antes mencionado.

Al carecer de ejemplos de cómo llevar a cabo su objetivo, con el RL se busca de manera indirecta llevar al agente de un estado inicial a un estado objetivo y esto se realiza por medio de lo que se denomina función de refuerzo. La idea es recompensar al agente por acciones que lo ayuden a llegar de una manera satisfactoria a su objetivo y así aprender una política de control, es decir que obtenga la capacidad de ejecutar, de entre las posibles acciones, aquella que lo ayude a acercarse a su objetivo.

Generalmente se modelan los problemas a tratar con RL mediante lo que se conoce como un proceso de decisión de Markov. (Mitchell, 1997)

4.1.2. Función de refuerzo

Función de refuerzo es aquella que asigna un valor numérico para las distintas acciones o secuencias de acciones que el agente tome en los diferentes estados.

4.1.3. Política

También conocida como política o estrategia de control, es la función que indica o recomienda que acción realizar para los diferentes estados. Lo que se busca que el agente aprenda es lo que se conoce como una política óptima que es aquella que a partir de un estado inicial recomienda una secuencia de acciones que llevan a maximizar la recompensa acumulada, es decir, la suma de los refuerzos recibidos.

4.1.4. Proceso de decisión de Markov (PDM)

Se conoce como proceso de decisión de Markov (Markov decisión process (MDP) en inglés) a un problema de decisión secuencial con un modelo de transición y recompensas acumuladas. Los elementos de un proceso de decisión de Markov son: un conjunto de estados S; un conjunto de acciones $A(s)$ para cada estado s; una función de transición; y una función de recompensa. (Russell y Norvig, 2009)

Para una instancia de tiempo t, el agente estará en un estado s_t en el que dicho agente puede ejecutar una acción a_t, del conjunto $A(s_t)$, con lo cual se recibirá una recompensa $r(s_t, a_t)$ y mediante la función de transición $\delta(s_t, a_t)$ se produce el siguiente estado. Las funciones $r(s_t, a_t)$ y $\delta(s_t, a_t)$ dependen solo del estado y acción actuales.

La solución para un PDM se conoce como política, la cual indica lo que tiene que hacer el agente para cada estado en el que se pueda encontrar. Comúnmente se usa π para denotar una política y $\pi(s)$ denotará la acción recomendada para el estado s. Se dice que una política es completa si para cualquier estado existe una acción recomendada por la política. Se conoce como política optima a aquella que maximiza la recompensa acumulada de las acciones tomadas, y se denota por π^*. (Mitchell, 1997)

4.1.4.1. Utilidades / Recompensas

Existen distintas formas para medir el desempeño/utilidad de un agente basado en un PDM. Una de las más utilizadas es la suma de las recompensas deducidas, donde la utilidad acumulada $V^{\pi}(s_t)$ para un estado s_t es: (Mitchell, 1997)

$$V^{\pi}(s_t) \equiv \ r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ... \tag{1}$$

donde r_{t+i} indica la secuencia de recompensas generadas al comenzar de un estado s_t y como consecuencia de utilizar una política π para la selección de acciones.

4.1.5. Proceso de Aprendizaje

El objetivo del agente es aprender una política $\pi \ : S \rightarrow \ A$, tal que esta dicte la siguiente acción a_t a ejecutarse siendo que el agente se encuentra en un estado s_t, es decir, $\pi(s_t) = a_t$.

Una forma para especificar qué política usar se basa en utilizar aquella política que genere la máxima utilidad. Se define el valor acumulado (utilidad) $V^{\pi}(s_t)$ siguiendo una política π de un estado arbitrario según la ecuación:

$$V^{\pi}(s_t) \equiv \ \sum_{i=0}^{\infty} \gamma^i r_{t+i} \tag{2}$$

que es equivalente a la ecuación 1.

La fórmula 2 nos indica que al comenzar en un estado s_t y al ejecutar las acciones recomendadas por la política π se obtendrá la secuencia de recompensa (Mitchell, 1997). A γ se le conoce como factor de descuento e indica la preferencia entre las recompensas inmediatas sobre las recompensas futuras. El valor de γ se define entre 0 y 1. Mientras más pequeño sea el valor de γ las recompensas futuras tendrán menos importancia para el agente. (Russell y Norvig, 2009)

Se busca que el agente aprenda una política que maximice $V^{\pi}(s)$ para todos los esta-

dos, a lo que se conoce como política óptima, la cual se denota por π^*.

$$\pi^* \equiv \ argmax V^{\pi}(s), (\forall s) \tag{3}$$

4.1.6. Q Learning

Para aquellos casos en los que la única información de la que dispone el agente es solo la secuencia de recompensas inmediatas, es difícil aprender una política de la forma $\pi \ : S \rightarrow \ A$. Es más sencillo implementar una política óptima a base de aprender una función de evaluación numérica que esté definida por estados y acciones.

Aprender la función V^* es una buena opción. Considerando las recompensas futuras acumuladas y si se tiene $V^*(s_1) > V^*(s_2)$ entonces el agente debe dar prioridad al estado s_1 sobre el estado s_2. Aunque en realidad la política debe decidir entre acciones y no estados. La acción que maximice la suma de las recompensas inmediatas $r(s, a)$ más el valor de V^* del estado sucesor, multiplicado por el factor γ , en el estado s, será la acción óptima.

$$\pi^*(s) = argmax[r(s, a) + \gamma V^*(\delta(s, a))] \tag{4}$$

4.1.6.1. La función Q

A la máxima recompensa acumulada, multiplicándole el factor γ , que puede ser recibida comenzando de un estado s y aplicando la acción a, se le conoce como función $Q(s, a)$. En otras palabras, el valor de la política optima sumado a la recompensa inmediata de ejecutar la acción a al encontrarse en el estado s.

$$Q(s, a) \equiv \ r(s, a) + \gamma V^*(\delta(s, a)) \tag{5}$$

Al ser el lado derecho de la ecuación $Q(s, a)$ la misma ecuación que se máxima en la función 4, podemos reescribir dicha función de la siguiente forma:

Algoritmo Algoritmo para Q Learning.

Para cada s, a inicializar las entradas de la tabla $\hat{Q}(s,a)$ a cero
Observar el estado actual s
Repetir por siempre
  Seleccionar una acción a y ejecutarla
  Recibir la recompensa inmediata r
  Observar el nuevo estado s'
    Actualizar la entrada de la tabla para $\hat{Q}(s,a)$ de la siguiente manera:
      $\hat{Q}(s,a) \leftarrow r + \gamma \max_{a'} \hat{Q}(s',a')$
  $s \leftarrow s'$

Figura 5: Un algoritmo para Q learning. Para éste algoritmo se asume que las recompensas y acciones son determinísticas.

$$\pi^*(s) = argmax Q(s,a) \tag{6}$$

Al aprender la función Q en lugar de V^* el agente será capaz de seleccionar acciones optimas aun cuando no tenga información de las funciones de transición o recompensa.

4.1.6.2. Un algoritmo para Q Learning

El algoritmo mostrado en la figura 5 realiza continuas iteraciones para actualizar los valores de la función Q al contar solo con las recompensas inmediatas. Esto se consigue reescribiendo la ecuación 5 de tal forma que se obtiene una ecuación recursiva:

$$Q(s,a) = r(s,a) + \gamma max_{a'} Q(\delta(s,a), a') \tag{7}$$

El algoritmo utiliza $\hat{Q}$ que es una estimación de Q, esta estimación/hipótesis, se define en el algoritmo como una tabla con una entrada para cada par $< s, a >$ (estado-acción). Estando en un estado s, el agente ejecuta una acción a, observa la recompensa inmediata, así como el estado sucesor s' y actualiza las entradas de la tabla siguiendo la regla:

$$\hat{Q}(s,a) \leftarrow r + \gamma \max_{a'} \hat{Q}(s',a') \tag{8}$$

(Mitchell, 1997)

4.2. Algoritmos genéticos

4.2.1. Introducción

Propuestos en los 70's por Holland, este tipo de algoritmos esta basado en en los principios de genética y evolución natural. La base recae sobre el proceso natural de supervivencia del más fuerte (apto). Los algoritmos genéticos mediante un proceso de selección generan varias poblaciones y las evolucionan por medio de procesos de cruce y mutación (Srinivas y Patnaik, 1994).

4.2.2. Proceso natural de evolución

En la naturaleza los individuos más aptos son los que sobreviven. La habilidad de adaptarse a los cambios en el ambiente es otro factor que ayuda a la supervivencia. El código genético de un individuo determina las características que lo definen. La unidad básica del código genético es conocida como gen. Los genes pueden ser considerados como la llave para la supervivencia de un individuo que se encuentra en un ambiente competitivo. La esencia del proceso de evolución son los cambios que se dan a nivel del código genético de los individuos.

En la naturaleza el individuo que sobrevive es aquel que sobrepasa a los demás en la competencia por recursos como comida o pareja para la reproducción, y puede ser considerado como el individuo más apto. Si un individuo sobrevive, también lo hacen sus genes, que se puede también decir que son los más aptos. Cuando dos individuos se reproducen ocurre recombinación de sus genes, el código genético cambia y es lo que se conoce como proceso de evolución. Este proceso de recombinación puede llevar a que aparezcan individuos que posean mejores genes que sus predecesores, y por lo tanto incrementar las posibilidades de supervivencia de los nuevos miembros de la especie.

4.2.3. Componentes de un algoritmo genético

Los componentes del algoritmo genético son los siguientes:

4.2.3.1. Población

Hipótesis, solución, individuo: Posibles soluciones para el problema que se busque resolver. Su representación suele variar dependiendo del problema aunque usualmente se utilizan cadenas de bits.

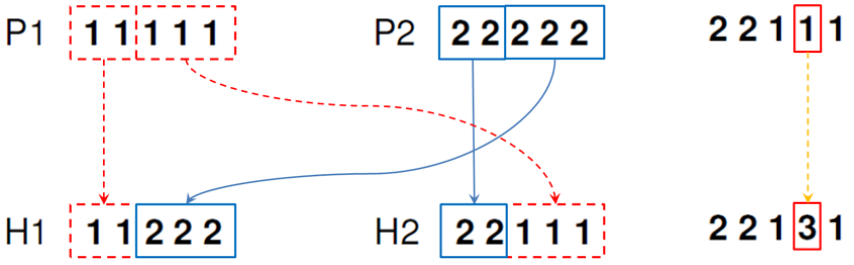
Representación: Utilizada para representar las posibles soluciones de forma a análoga a los cromosomas. La representación depende del problema a tratar. Usualmente se suelen usar cadenas de bits o de números enteros. (Marsland, 2011)

4.2.3.2. Parámetros de control

Controlan la dinámica de la población y son: tamaño de la población, probabilidad de mutación y probabilidad de cruce.

4.2.3.3. Función objetivo

Para determinar qué tan buena es una posible solución es necesario un criterio de evaluación, este criterio se conoce como función objetivo. La función objetivo también ayuda a determinar cuáles de las posibles soluciones generadas pasaran a la siguiente generación (Mitchell, 1997). La función objetivo provee un mecanismo para evaluar a los individuos (posibles soluciones). Para ciertos problemas es necesario mantener la uniformidad, por lo cual se suele normalizar a valores dentro de un rango conveniente de 0 a 1. El valor normalizado de la función objetivo representa la aptitud del individuo (Srinivas y Patnaik, 1994).



(a) Cruce de un punto. (b) Mutación.

Figura 6: Operadores geneticos comunes. a) El cruce de las cadenas padre P1 y P2, generan las cadenas hijas H1 y H2. Las flechas grandes indican el punto de cruce. b) En la parte de arriba la cadena original con un recuadro que indica el alelo a mutar.

4.2.3.4. Operadores Genéticos

Ayudan a determinar las siguientes generaciones por medio de recombinaciones y mutaciones aplicadas a los individuos. Estas operaciones hacen alusión a los procesos que se llevan a cabo en los sistemas biológicos. Los operadores que se utilizan con más frecuencia son mutación y cruce. En la figura 6 se pueden apreciar ejemplos de los operadores de cruce y mutación.

Cruce: El paso siguiente al proceso de selección es la aplicación del operador de cruce. Pares de cadenas son seleccionados y sujetos al cruce. Una de los intentos más sencillos de cruce es el cruce de un solo punto, proceso que selecciona un punto de cruce al azar de un determinado rango. Las porciones de las cadenas que se encuentren antes del punto de cruce se intercambian para generar dos nuevas cadenas. La operación de cruce no siempre se realiza, esta depende de lo que se le llama probabilidad de cruce, que es un número en el rango de 0 a 1.

Mutación: Después del cruce viene el proceso de mutación. Este proceso involucra introducir variaciones a los genes del individuo. De forma similar al operador de cruce, existe un parámetro, probabilidad de mutación, que indica si se realizará o no dicha operación. La mutación puede ser tratada como un operador que es capaz de restaurar el material genético que se perdió, pero de la misma forma puede introducir nuevas variantes en el material genético.

Los dos operadores genéticos que se acaban de describir están relacionados de forma estrecha con la teoría de construcción de bloques (“building blocks” en ingles), en especial el operador de cruce. Esta teoría asume que la yuxtaposición de bloques de construcción de calidad dará como resultado descendencia de buena calidad. Cabe mencionar que esto está también directamente relacionado con la naturaleza de la función objetivo, ya que con los mismos bloques de origen, y si la naturaleza de la función objetivo es desfavorable pueden llegar a generarse bloques de construcción de muy mala calidad. El operador de cruce esta relacionado con la conservación de información, mientras que el operador de mutación con la generación de nuevos bloque de construcción (Srinivas y Patnaik, 1994).

4.2.3.5. Mecanismo de selección

Se utiliza para modelar el mecanismo natural de supervivencia del individuo más apto. En general, aquellas soluciones con mejor aptitud sobrevivirán, mientras que aquellas menos aptas desaparecerán.

A continuación se describen algunos de los mecanismos de selección que se suelen utilizar:

Ruleta: Mecanismo utilizado para generar una selección proporcional. Este mecanismo simula una ruleta, donde a cada individuo se le asigna una sección (Srinivas y Patnaik, 1994).

Torneo: Para un caso en el que dos individuos padres generan 2 nuevos individuos hijos, el mecanismo de selección de torneo evaluará a los cuatro individuos y seleccionará a los dos más aptos para que formen parte de la nueva población.

Elitismo: El elitismo es una técnica que es utilizada para que las mejores soluciones que se encontraron en una generación no desaparezcan al generarse una nueva. La idea es sencilla, las mejores soluciones se copian tal cual a la nueva generación.

Nichos: Mecanismo que consiste en evolucionar subpoblaciones durante ciertos periodos de tiempo, y ocasionalmente copiar algunos individuos a diferentes subpoblaciones.

Cuando se utilizan los mecanismos de torneo y elitismo se tiende a favorecer a aquellos individuos que muestran una mejor aptitud, pero estos individuos no siempre llevarán a encontrar la mejor solución, sino que existe la posibilidad de que se queden estancados ya que se puede dar pie a una población con escasos cambios, esto es por lo que se permite que los mismos individuos perduren por varias generaciones y se reduzca la diversidad, lo que eventualmente puede llevar a una población donde los individuos tienden a ser iguales. Una forma de solucionar la decaída de exploración generada por torneo y elitismo es mediante el uso de mecanismos como el de nichos los cuales tienden a mantener la diversidad (Marsland, 2011).

4.2.4. Proceso de aprendizaje

Un algoritmo genético emula teorías biológicas de evolución para resolver problemas. Un algoritmo genético esta compuesto por un conjunto de elementos individuales, al cual se le llama población, y por un conjunto de operadores biológicamente inspirados. Solamente los elementos más aptos dentro de la población tendrán la oportunidad de sobrevivir y reproducirse, y así transmitir su información a las nuevas generaciones.

En términos de computación, en un algoritmo genético se realiza un mapeo de un problema hacia un conjunto de cadenas, donde cada cadena representa una posible solución para el problema. El algoritmo genético después irá manipulando las cadenas más prometedoras en su búsqueda de mejores soluciones.

Un algoritmo genético opera mediante un sencillo ciclo de varios estados:

- Creación de la población de cadenas.
- Evaluación de cada cadena.
- Selección de la mejor cadena.
- Manipulación genética para crear una nueva población de cadenas.

Algoritmo Un algoritmo genético genérico.

```
1: Inicialización de la población
2: Evaluar población
3: Repetir mientras no se cumpla el criterio de terminación
4:     Seleccionar las soluciones para la siguiente población
5:     Realizar operaciones de cruce y mutación
6:     Evaluar población
```

Figura 7: Un algoritmo generativo genérico, en el cual se muestra de forma simple y resumida el flujo de un algoritmo genético (Mitchell, 1997).

La figura 7 presenta un algoritmo genético genérico donde cada ciclo produce una nueva generación de posibles soluciones para un determinado problema. En el primer estado se crea una población inicial de posibles soluciones, esta población será el punto de comienzo de la búsqueda. En el siguiente estado, las soluciones codificadas como cadenas son evaluadas. Basándose en la evaluación de la aptitud de las cadenas el mecanismo de selección escoge las cadenas a utilizar para la manipulación genética. El proceso de selección determinará cual es el individuo más apto para sobrevivir. Se utilizan los operadores genéticos para generar nuevos miembros que se incluirán a la población. El operador de cruce recombina el material genético, mientras que el operador de mutación produce variantes en la estructura genética. Los descendientes producidos mediante la manipulación genética conformaran la siguiente población a ser evaluada (Srinivas y Patnaik, 1994).

Capítulo 5. Modelo del sistema

En esta sección se presentará de una manera breve el modelo que se siguió para desarrollar el sistema: “agente basado en gramáticas”. A grandes rasgos el modelo esta conformado por dos componentes:

Agente :

Un agente robótico o virtual el cual funcionará a base de una gramática de comportamientos para desenvolverse e interactuar en/con el ambiente.

Ambiente :

Medio en el que se desenvolverá el agente.

El modelo general del agente que se utilizará consta de cinco bloques o módulos principales los cuales, al interactuar entre si y con el ambiente, darán como resultado la generación de diversos comportamientos. Un diagrama que representa el modelo general del sistema puede ser observado en la figura 8. A continuación se describen de forma breve los diversos bloques que conforman al agente:

Sensores :

Los sensores son el medio por el cual el agente percibe el ambiente.

Módulo de eventos :

Utilizando las lecturas de los sensores determina si se generó o no, un evento. Dependiendo de los eventos ocurridos, cuando sea necesario, éste módulo indicará al módulo de acciones como proseguir.

Módulo de acciones :

Este módulo se encargará de generar acciones complejas. Este módulo podrá ser directamente influenciado por el módulo de eventos e influenciará directamente a los actuadores.

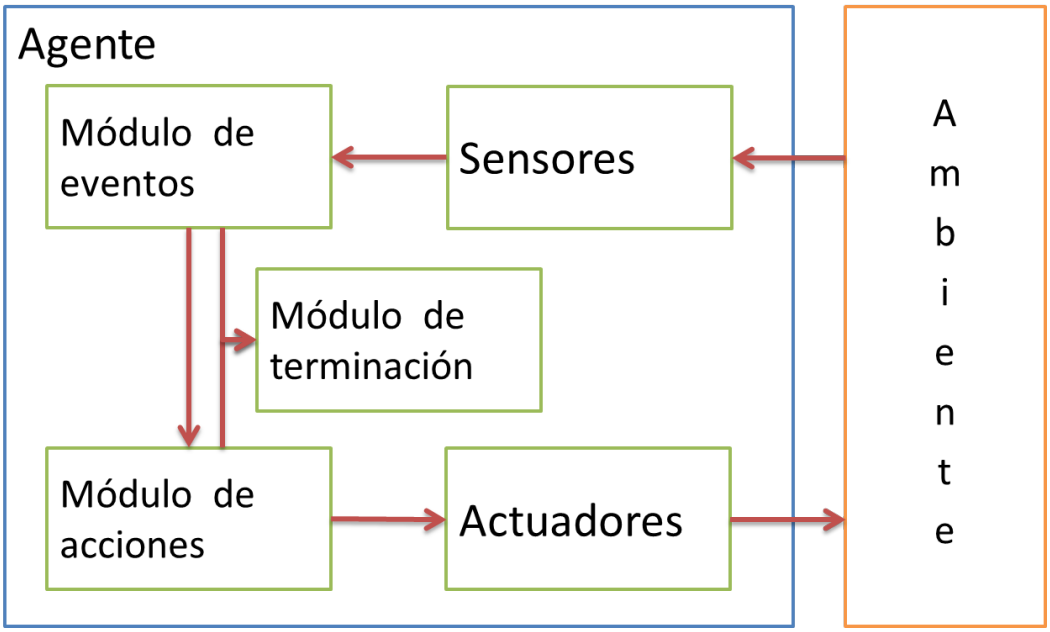


Figura 8: Modelo general del sistema. El agente interactua directamente con el ambiente por medio de los sensores y actuadores.

Actuadores :

Son el medio por el cual el agente interactua con el ambiente, ya sea cambiándolo o cambiándose a si mismo. Este bloque actúa directamente bajo el módulo de acciones.

Módulo de terminación :

Termina o para, el comportamiento que se este ejecutando. Este módulo se puede activar por medio de el módulo de acciones o el de eventos.

Los módulos de eventos y acciones están directamente relacionados con los sensores y actuadores respectivamente; pudiendo decir que estos últimos son el medio por el cual los módulo de eventos y acciones interactuan con el ambiente.

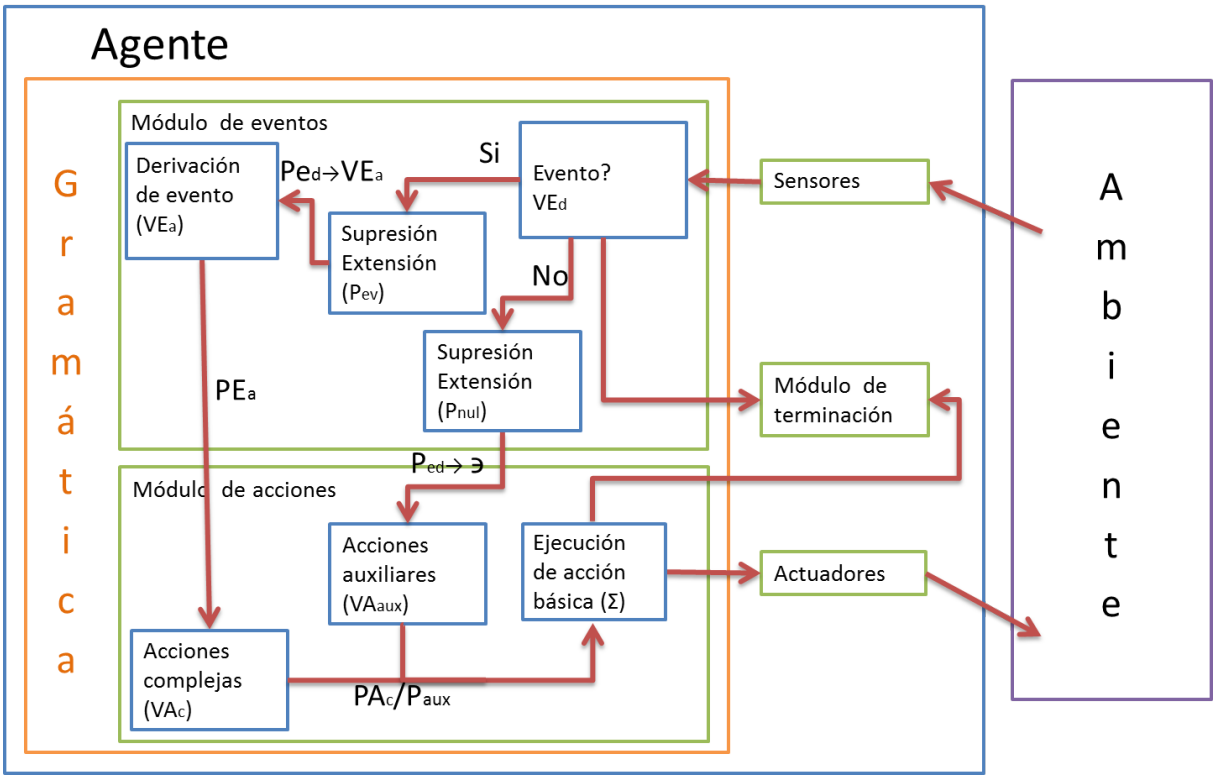


Figura 9: Modelo específico del sistema. Los módulos de acciones y eventos forman parte de lo que es la gramática de comportamientos del agente. Dentro de los bloques de los distintos sub-módulos se puede observar el correspondiente subconjunto de la gramática de comportamientos.

Dentro del agente (ver figura 9) los módulos de acciones y eventos formarán el sistema de una gramática de comportamientos, y a la vez dentro de ellos habrá algunos sub-módulos los cuales tendrán correspondencia con los distintos subconjuntos de variables y producciones de la gramática. La definicion formal, estructura, construcción y flujo de una gramática de comportamientos se presentará en el siguiente capítulo.

Capítulo 6. Gramáticas de comportamientos (GC)

Una gramática de comportamientos es un método propuesto para la representación y generación de comportamientos basados en eventos y acciones. Para un agente los eventos equivalen al sensado del ambiente por medio de sensores y las acciones a la interacción en el ambiente por medio de actuadores. Se les llamará gramáticas de comportamientos robóticos (GCR) para los casos de su aplicación con robots.

6.1. Definición formal de una gramática de comportamientos (GC)

Una gramática de comportamientos es una gramática independiente de contexto, la cual sigue ciertas reglas para su construcción.

Para una gramática de comportamientos $\mathbf{GC} = (V, \Sigma, P, S)$, sus componentes se definen de la siguiente forma:

- $\mathbf{V}$ - Las variables de la gramática. Se dividirán en varios subconjuntos:
  - $\mathbf{VA_c}$ - Variables que representan acciones complejas.
  - $\mathbf{VE_a}$ - Las variables de este subconjunto representan eventos/estados.
  - $\mathbf{VE_d}$ - Una sola variable, que llamaremos la variable de derivación de eventos.
  - $\mathbf{V_{aux}}$ - Las variables de este subconjunto son variables auxiliares.
- $\mathbf{S}$ - Símbolo y evento inicial, que pertenece al conjunto V, subconjunto VE_a.
- $\mathbf{\Sigma}$ - Los símbolos terminales, que representaran las acciones básicas del agente.
- $\mathbf{P}$ - Las producciones se dividirán en varios subconjuntos los cuales están directamente relacionados con los subconjuntos de variables de V.
  - $\mathbf{PA_c}$ - Producciones de generación de acciones complejas (c).
  - $\mathbf{PE_a}$ - Producciones de selección de acciones (a) para eventos.
  - $\mathbf{PE_d}$ - Producciones para derivación (d) de eventos.
  - $\mathbf{P_{aux}}$ - Producciones auxiliares (aux) para acciones.
  - $\mathbf{PSE_{ev_n}}$ y $\mathbf{PSE_{nul_n}}$ - Producciones de supresión y extensión.

6.1.1. Construcción de los subconjuntos de producciones

Los distintos subconjuntos de producciones siguen ciertas reglas para la formación de sus producciones, las cuales se describen a continuación.

- $\mathbf{PA_c}$ - Reglas de producción que representaran acciones complejas y cuyas cabezas de producción son variables que pertenecen al subconjunto VA_c. Estas producciones tendrán una forma similar a la forma normal de Greibach, el cuerpo estará conformado por un símbolo terminal seguido de una cadena de variables, pero solo se restringirá a solo dos variables. A las producciones de este tipo se les llamara producciones de acción y tendrán la siguiente forma:

  $A \rightarrow\ aEB$

  donde

  A es una variable del subconjunto VA_c, a es un símbolo terminal, E es la variable para derivación de eventos del subconjunto VE_d y B es una variable que pertenece al subconjunto VA_c o V_{aux}.

  Ejemplo:

  $\Sigma\ = \{a, b, c, d, e, f, g\}$

  $VA_c = \{A, B, C, D\}$

  $VE_d = \{E\}$

  $PA_c = \{A \rightarrow\ aEA, \quad B \rightarrow\ bEC, \quad C \rightarrow\ cED, \quad D \rightarrow\ dED\}$

  Para este subconjunto, no se permite el caso de variables que puedan derivarse de varias formas, es decir, las cabezas de las reglas de este conjunto deben de ser diferentes, por ejemplo tener las siguientes reglas seria incorrecto, lo ideal es tener solo una regla.

  Ejemplo:

  Incorrecto: Mas de una regla de producción cuya cabeza sea la misma variable, en este caso la variable D.

  $PA_c = \{A \rightarrow\ aEA,\ \ B \rightarrow\ bEC,\ \ D \rightarrow\ dED,\ \ D \rightarrow\ aEB\ \ , D \rightarrow\ bEA\}$

  Correcto: Todas las cabezas de producción de las reglas son diferentes.

  $PA_c = \{A \rightarrow\ aEA, \quad B \rightarrow\ bEC, \quad D \rightarrow\ dE\}$

- $\mathbf{PE_a}$ - Las producciones de este subconjunto representan eventos y las posibles acciones para dicho evento. La cabeza de la producción pertenecerá al subconjunto VE_a, mientras que los cuerpos de producción pertenecerán al subconjunto VA_c. Estas producciones tienen la siguiente forma:

  $E_i \rightarrow \ A_j$

  donde

  E_i es una variable que pertenece al conjunto VE_a y A_j es una sola variable que pertenece al conjunto VA_c. Dichas variables representan un estado y una acción respectivamente, por lo que se puede decir que A_j es una posible acción a realizar al estar en el estado E_i. Por lo menos tiene que definirse una regla de producción de esta forma para cada una de las variables del subconjunto VE_a.

  Ejemplo:

  $VE_a = \{E_1, E_2, E_3\}$

  $VA_c = \{A_1, B, C\}$

  $PE_a = \{E_1 \rightarrow \ A \mid B \mid C, \quad E_2 \rightarrow \ A \mid C, E_3 \rightarrow \ A \mid B \mid C\}$

  Para las reglas cuya cabeza es la variable E_1, podemos decir que, al estar en el estado de decisión E_1 el agente puede realizar las acciona A, B o C.

  Si una variable E_x pertenece al subconjunto VE_a, entonces debe de por lo menos haber una regla en este subconjunto (PE_a) cuya cabeza de producción sea E_x.

- $\mathbf{PE_d}$ - Las producciones en este subconjunto se utilizan para realizar la derivación de eventos. Todas las producciones de este subconjunto tienen la misma variable como cabeza de producción, la variable de derivación de eventos, la cual pertenece al subconjunto VE_d. Se incluirá una regla por cada variable E_x del subconjunto VE_a de la forma:

  $E \rightarrow \ E_x$

  y de manera opcional una para la cadena vacía:

  $E \rightarrow \ \epsilon$

  donde

E es la variable de derivación de eventos perteneciente al subconjunto VE_d, E_x es una variable del subconjunto VE_a la cual representa un evento y ϵ representa la cadena vacía.

Ejemplo:

$VE_a = \{E_1,\ E_2,\ E_3,\ \ ...\ E_n\}$

$VE_d = \{E\}$

$PE_d = \{E \rightarrow\ E_1,\ \ E \rightarrow\ E_2,\ \ ...,\ \ E \rightarrow\ E_n,\ \ E \rightarrow\ \epsilon\ \}$

o en notación compacta:

$PE_d = \{E \rightarrow\ E_1 \mid E_2 \mid ... \mid E_n \mid \epsilon\ \}$

- $\mathbf{PA_{aux}}$ - subconjunto de producciones auxiliares, las cuales pueden ser parte de algún comportamiento más complejo. Las cabezas de producciones pertenecen al subconjunto de variables V_{aux}. Estas reglas pueden tener dos formas, una similar a las producciones de PA_c, bajo las mismas restricciones (producciones de acción) o una forma que solo contenga variables, las que pueden pertenecer a los subconjuntos VA_c y/o V_{aux} y las cuales se llamaran producciones complemento o auxiliares.

  Ejemplo:

  $\Sigma\ = \{a, b, c, x, y, z\}$

  $VA_c = \{A, B, C\}$

  $V_{aux} = \{F, G, H\}$

  $VE_d = \{E\}$

  $PA_c = \{A \rightarrow\ aEF,\ \ B \rightarrow\ bEG,\ \ C \rightarrow\ cEH\}$

  $P_{aux} = \{F \rightarrow\ xEG,\ \ G \rightarrow\ yEA,\ \ H \rightarrow\ CAC\}$

  De forma similar al subconjunto PA_c una misma variable no puede aparecer en más de una cabeza de producción.

- $\mathbf{PSE_{ev_n}}$ y $\mathbf{PSE_{nul_n}}$ - Producciones opcionales de supresión y extensión. Son familias de subconjuntos donde el subindice n representará el nivel del subconjunto. Se utilizará la familia de subconjuntos PSE_{ev_n} cuando se genera un evento (ev), mientras que los subconjuntos PSE_{nul_n} para los casos en que se utiliza el modo de omisión por evento repetido o nulo (nul).

Las producciones de estos subconjuntos podrán presentar una de las siguientes formas:

$A \rightarrow \ \epsilon$

$A \rightarrow \ B$

donde

A es una variable que pertenece a alguno de los subconjuntos de variables VA_c o V_{aux}, ϵ representa la cadena vacía y B es una sola variable que pertenece al subconjunto V_{aux}. En un subconjunto no debe haber más de una regla de producción con la misma variable como cabeza.

6.2. Generación de comportamientos por medio de una GC

La forma por medio de la cual se generan los comportamientos en una GC es por medio de los que conoce como proceso de derivación de gramáticas. En esta sección se describirá la forma por medio de la cual se realiza la derivación de una gramática de comportamientos.

La forma en que se realiza la derivación de la gramática de comportamientos depende de los distintos subconjuntos de variables y producciones. El proceso de derivación de una gramática de comportamiento conlleva a la ejecución de ciertas rutinas las cuales se ejecutarán dependiendo de la expansión de reglas de producción, por ejemplo, una variable que, según la definición de la gramática, pueda ser expandida de varias formas, a menos de que ciertas condiciones se cumplan será expandida siempre por medio de la misma regla de producción.

Cada vez que se expanda una variable y aparezca un nuevo símbolo terminal se ejecutará una acción básica correspondiente a este. El tipo de derivación por defecto sera una derivación más a la izquierda, lo que garantiza un cierto orden en el proceso en que se genera el comportamiento, y por definición de las reglas de producción del subconjunto PA_c se garantiza que siempre que aparezca un símbolo terminal, a su izquierda se encontrará una cadena de solo terminales y a su derecha una de solo variables. Bajo ciertas circunstancias se realizarán excepciones al orden de derivación, pero estas no generarán símbolos terminales.

En las siguientes secciones se describirán las condiciones por medio de las cuales se realizará la derivación de la GC.

6.2.1. Selección de acciones

Al encontrarse en el estado inicial o en algún estado E_a, es decir una forma de sentencia:

wE_x

donde

w es una cadena de símbolos terminales o una cadena vacía, E_x es una variable que representa un evento y pertenece al subconjunto VE_a, este tipo de variables también serán llamadas variables de evento o de estado.

Diremos que nos encontramos en un estado de decisión E_x. Se le llamará estado de decisión porque para continuar el comportamiento es necesario expandir (derivar) E_x, la cual de acuerdo a la definición, debe de tener en PE_a por lo menos una regla de producción para su derivación. Si solo se cuenta con una regla de producción en PE_a cuya cabeza sea E_x entonces se utilizará dicha regla para expandir E_x. En el caso de que se cuente con más de una regla de producción para expandir E_x entonces sera necesario algún método externo para seleccionarla.

Ejemplo:

Si se encuentra en forma de sentencia, es decir en un estado de decisión E_a

wE_x

$E_x \rightarrow \ A_1 \mid A_2 \mid A_3 \mid ... \mid A_n$

y se decide expandir la tercera regla, es decir la correspondiente a la acción A_3, entonces

$wE_x \Rightarrow \ wA_3$

proceso que se puede leer como: Al ocurrir el evento E_x el agente decidió ejecutar la acción A_3.

Más adelante en esta tesis se presentarán varios métodos para asistir al proceso de selección de reglas para los casos en los que alguna variable de VE_a cuente con más de una regla. También se verá un ejemplo en el cual se llevará al agente a seleccionar la acción optima, es decir la acción que al ocurrir un evento E_x (al encontrarse en un estado de decisión E_x) la acción seleccionada ayude al agente a cumplir con su propósito de la manera más eficiente. Este tipo de derivación es una derivación más a la izquierda.

6.2.2. Generación de eventos

En una gramática de comportamientos el subconjunto PE_d describe las reglas de producción por medio de las cuales se puede expandir la variable especial (y única) del subconjunto VE_d de variables.

De acuerdo a la definición de los tipos de reglas de producción, la variable de derivación de eventos (variable única del subconjunto VE_d) siempre aparece a la derecha de un símbolo terminal, el cual al aparecer por primera vez ejecuta la acción básica que le corresponde.

Inmediatamente después de que un símbolo terminal ejecute la acción básica correspondiente y si se sigue el orden de derivación más a la izquierda, entonces es el turno de la variable de derivación de eventos para ser expandida. La expansión de la variable de derivación depende de los siguientes casos principales:

Caso 1 - Derivación por evento válido: Si en reacción a la acción básica ejecutada se disparó un evento, entonces del subconjunto PE_d seleccionará la regla de producción que represente dicho evento (para el subconjunto PE_d los cuerpos de las producciones están conformados por solo una variable que pertenece al subconjunto VE_a, cuyas variables representan los distintos eventos).

Caso 2 - Omisión por evento repetido/nulo: Una GC contempla dos casos en los cuales es posible continuar la expansión de variables sin afectar la derivación actual, estos casos son:

- No se generó/disparó algún evento.
- El evento generado es el mismo que el anterior, es decir el agente sigue en el mismo estado (estado=evento) por lo cual no se ve la necesidad de interferir en la derivación.

Ejemplo:

Para una gramática de comportamientos $\mathbf{GC} = (V, \Sigma, P, S)$

$\mathbf{V} = \{VA_c, V_{aux}, VE_a, VE_d\}$ $\mathbf{P} = \{PA_c, P_{aux}, PE_a, PE_d\}$ $\mathbf{S} = \{S\}$

$\mathbf{\Sigma} = \{a, b, c\}$ $\mathbf{VA_c} = \{A, B, C\}$ $\mathbf{V_{aux}} = \{F, G, H\}$ $\mathbf{VE_a} = \{S, E_a, E_b, E_c\}$ $\mathbf{VE_d} = \{E\}$

$\mathbf{PA_c} = \{A \rightarrow\ aEF, B \rightarrow\ bEG, C \rightarrow\ cEH\}$ $\mathbf{P_{aux}} = \{F \rightarrow\ xEG, G \rightarrow\ H, H \rightarrow\ CAC\}$

$\mathbf{PE_a} = \{E_1 \rightarrow\ A, E_2 \rightarrow\ B, E_3 \rightarrow\ C\}$

si se tiene una forma de sentencia

wA

donde w es una cadena de símbolos terminales y A es una variable de acción que se expandirá

$wA \Rightarrow\ waEF$

a es un símbolo terminal y como acaba de aparecer se ejecuta la acción básica que le corresponde, E es la variable de derivación de eventos, F es una variable auxiliar

Entonces para los dos casos principales de derivación de eventos:

Caso 1: Derivación por evento válido.

$\mathbf{PE_d} = \{E \rightarrow\ E_a, E \rightarrow\ E_b, E \rightarrow\ E_c\}$

Se genera el evento # 2 que corresponde a la variable E_b

$waEF \Rightarrow\ waE_bF$

se continua la derivación

$waE_bF \Rightarrow\ waBF \Rightarrow\ wabEGF$

se ejecuta b y se genera de nuevo el evento # 2

$wabEGF \Rightarrow\ wabE_bGF \Rightarrow\ wabBGF \Rightarrow\ wabbEGGF \Rightarrow\ ...$

Caso 2: Omisión por evento repetido.

$\mathbf{PE_d} = \{E \rightarrow\ E_a, E \rightarrow\ E_b, E \rightarrow\ E_c\, E \rightarrow\ \epsilon\ \}$

Se genera el evento # 2 que corresponde a la variable E_b

$waEF \Rightarrow\ waE_bF$

se continua la derivación

$waE_bF \Rightarrow\ waBF \Rightarrow\ wabEGF$

se ejecuta b y se genera de nuevo el evento # 2 y por ser el modo omisión se selecciona $E \rightarrow\ \epsilon$

$wabEGF \Rightarrow\ wabGF \Rightarrow\ wabHF \Rightarrow\ wabCACF \Rightarrow\ ...$

6.2.3. Ejecución de acciones

La forma en que un agente basado en una gramática de comportamientos interactua con el ambiente y cambia de estado (genera eventos) es por medio de sus acciones. Las acciones en una gramática de comportamientos se definen mediante los diversos componentes de la gramática y se agrupan en dos grupos:

Acciones básicas: Símbolos terminales. Están directamente relacionadas con los actuadores del agente o con secuencias de movimiento ya definidas. Son atómicas, es decir no se interrumpen. Ejemplos pueden ser el moverse una unidad hacia el frente en un “grid” (rejilla, espacio cuadriculado), dar un paso, etc.

Acciones complejas: Variables. Una acción compleja es una combinación de acciones básicas. En una gramática de comportamientos esto se consigue por medio del uso de variables y de las reglas de producción.

La ejecución de acciones corresponde al proceso por medio del cual una acción compleja seleccionada al generarse un evento es ejecutada. Una acción compleja es ejecuta por medio de los símbolos terminales que aparezcan mediante el proceso de derivación, y estos terminales solo aparecen al ser expandidas la producciones que se llamarán producciones de acción.

Producción de acción: $A \rightarrow aEA$

Este tipo de producciones son las que introducen los símbolos terminales a la cadena, así como la variable de derivación de eventos, por lo cual juegan un papel activo en la ejecución de acciones así como en la generación y derivación de eventos. La cabeza de estas producciones puede ser una variable que pertenezca al subconjunto VA_c o al V_{aux}. El cuerpo de estas producciones tiene la forma:

aEA

donde

a es un símbolo terminal, E es la variable de derivación de eventos y A es una variable que pertenece a uno de los subconjuntos VA_c o V_{aux}.

Al expandirse una variable por medio de una producción de acción se ejecutará inmediatamente la acción básica correspondiente al símbolo terminal, se revisará si se generó evento o no, y si las condiciones se cumplen se realizará el proceso de supresión y expansión de variables.

En el ejemplo de la sección anterior (generación de eventos) se puede apreciar también este proceso.

6.2.3.1. Supresión y extensión de acciones

En ocasiones puede llegar a ser necesario el suprimir, cancelar o modificar un comportamiento que se encuentre por comenzar o ya en proceso de ejecución. Estas situaciones se incluyen en la gramática de comportamientos como dos procesos secundarios a la expansión de variables de acciones. Estos procesos reciben los nombres de supresión de variables y extensión de variables.

La supresión y extensión de variables se lleva a cabo de la misma forma y el resultado tiene que ver con la forma de las reglas de producción de los subconjuntos PSE_{ev_n} y PSE_{nul_n}. De los procesos de derivación de una GC, supresión y extensión son los únicos procesos de derivación que no siguen una derivación más a la izquierda.

Es una supresión de variables si el cuerpo de la regla de producción es el símbolo de la cadena vacía, es decir la regla de producción tiene la forma:

$A \rightarrow \epsilon$

Por otro lado, se considera extensión de variables si el cuerpo de la regla de producción es otra variable, es decir la regla de producción tiene la forma:

$A \rightarrow B$

La característica que resalta en las producciones de los subconjuntos de supresión y extensión radica en el cuerpo de las reglas de producción, el cual o es la cadena vacía o una sola variable. Se le llama supresión ya que el cuerpo de producción es la cadena vacía y al expandir la variable de cabeza en su lugar quedará la cadena vacía, es decir nada, con lo cual se puede suprimir un comportamiento en curso. Por otro lado, si el cuerpo de la producción es una variable, y no la cadena vacía, entonces el comportamiento en curso puede cambiar un poco, por lo que se dice que se extendió la variable.

Los procesos de supresión y extensión se realizan inmediatamente después de que se seleccione la regla de producción con la cual se expandirá la variable de derivación de eventos pero antes de que ésta sea expandida. Esto es por que las variables afectadas serán las que se encuentren a su derecha.

Los subconjuntos PSE_{ev_n} y PSE_{nul_n} son los casos generales de supresión y extensión, donde el subindice n indica el nivel de supresión y/o extensión. El nivel indicará cuantas variables a la derecha de la variable de derivación de eventos verán afectadas por supresión y/o extensión. Un nivel n (o N) indica que todas las variables a la derecha serán afectadas.

Se definirán dos situaciones bajo las cuales se realizarán estos procesos, las cuales

corresponden a los casos de generación de eventos:

Caso 1 - Derivación por evento (ev) **valido -** Subconjunto PSE_{ev_n}: Si se genera un evento, entonces se utilizarán las reglas de los subconjuntos PSE_{ev_n} de los diferentes niveles que se hayan definido.

Caso 2 - Omisión por evento repetido/nulo (nul) **-** Subconjunto PSE_{nul_n}: Para aquellos casos en los que se incluye la regla de producción cuyo cuerpo es ϵ (la cadena vacía) y para cuando se selecciones esta regla para expandir la variable de derivación de eventos, antes de dicha expansión se realizará el proceso de supresión/extensión utilizando las reglas de producción de los subconjuntos PSE_{nul_n} de los diferentes niveles que se hayan definido.

Este proceso solo afectará, es decir que solo se expandirán, aquellas variables que cumplan con las siguientes condiciones:

1. Solo aquellas variables que se encuentren en la cadena al momento de que se llevo a cabo la expansión de la producción de acción.
2. Solo aquellas variables que cuenten con una regla de producción, en alguno de los subconjuntos de supresión/extensión, por medio de la cual puedan ser expandidas.

Este proceso se realiza cada vez que se expande una producción de acción, inmediatamente después de que se expandió la variable y antes de ejecutar la acción básica y se realizará de la siguiente forma y tomando en consideración los siguientes puntos:

1. Se determinan las variables a expandir (suprimir/extender).
2. Se seleccionarán las producciones a utilizar para expandir las variables.
    - El nivel del subconjunto determina el rango de alcance de una producción, es decir, una variable que se encuentre 4 posiciones a la derecha de la variable de derivación de eventos, no puede ser expandida por una producción que pertenezca a un subconjunto de nivel menor a 4, es decir, si posición $>$ nivel no se puede aplicar dicha regla.

- Si una variable entra en el rango de más de una producción, se secciona la producción del subconjunto de menor nivel.

3. Una vez determinadas las variables afectadas y las reglas de producción correspondientes, se expandirán todas las variables a la vez, esto con el fin de alterar las posiciones o introducir nuevas variables antes de que termine el proceso.

Ejemplo:

Para una gramática de comportamientos $\mathbf{GC} = (V, \Sigma, P, S)$

$\mathbf{V} = \{VA_c, V_{aux}, VE_a, VE_d\} \quad \mathbf{S} = \{S\}$

$\mathbf{P} = \{PA_c, P_{aux}, PE_a, PE_d, PSE_{ev_1}, PSE_{ev_3}, PSE_{ev_n}, PSE_{nul_1}, PSE_{nul_3}\}$

$\mathbf{\Sigma} = \{a, b, c\}$ $\mathbf{VA_c} = \{A, B\}$ $\mathbf{V_{aux}} = \{F, G, H, C\}$ $\mathbf{VE_a} = \{S, E_a, E_b, E_c\}$ $\mathbf{VE_d} = \{E\}$

$\mathbf{PA_c} = \{A \rightarrow\ aEA, B \rightarrow\ bEB\}$ $\mathbf{P_{aux}} = \{F \rightarrow\ xEF, G \rightarrow\ aEB, H \rightarrow\ CAC, C \rightarrow\ A\}$

$\mathbf{PE_d} = \{E \rightarrow\ E_a, E \rightarrow\ E_b, E \rightarrow\ \epsilon\ \}$ $\mathbf{PE_a} = \{E_1 \rightarrow\ A, E_2 \rightarrow\ B\}$

$\mathbf{PSE_{ev_1}} = \{B \rightarrow\ G, F \rightarrow\ A, H \rightarrow\ B\}$ $\mathbf{PSE_{ev_4}} = \{A \rightarrow\ H, F \rightarrow\ A, H \rightarrow\ B\}$

$\mathbf{PSE_{ev_n}} = \{A \rightarrow\ \epsilon, B \rightarrow\ \epsilon, C \rightarrow\ \epsilon\ \}$

$\mathbf{PSE_{nul_1}} = \{A \rightarrow\ G\}$ $\mathbf{PSE_{nul_3}} = \{A \rightarrow\ F, B \rightarrow\ G, C \rightarrow\ H\}$

Se supone que se está en la siguiente forma de sentencia

$wABCABCABC$

donde w es una cadena de símbolos terminales

1. Se expande la variable A más a la izquierda utilizando la regla de producción $A \rightarrow aEA$

   $w\mathbf{A} \Rightarrow\ w\mathbf{aEA}BCABCABC$

2. Se ejecuta la acción del terminal a para lo cual no se genera un evento por lo cual se selecciona la regla de producción $E \rightarrow \epsilon$

3. Se inicia el proceso de supresión/extensión y ya que no se genero evento se utilizarán las producciones PSE_{nul_1} y PSE_{nul_3}.

   a) El mayor rango es 3 por lo cual las variables candidatas son las 3 variables que se encuentran a la derecha de la variable de derivación de eventos:

   $\alpha\ aE\mathbf{ABC}ABCABC$

   b) Para cada variable las posibles reglas de producción a utilizar son:

   A :

   Nivel 1: $A \rightarrow G$

   Nivel 3: $A \rightarrow F$

   B :

   Nivel 3: $B \rightarrow G$

   C :

   Nivel 3: $C \rightarrow H$

   c) Como para la variable A hay dos reglas de producción se selecciona la de menor nivel, por lo que las reglas a utilizar serán:

   $A : A \rightarrow G$

   $B : B \rightarrow G$

   $C : C \rightarrow H$

   d) Se expanden las variables dentro del rango todas a la vez y se obtiene:

   $waE\mathbf{ABC}ABCABC \Rightarrow waE\mathbf{GGH}ABCABC$

4. Se expande la variable de derivación de eventos utilizando la regla $E \rightarrow \epsilon$ ya que se había mencionado que no se genero evento.

   $wa\mathbf{E}GGHABCABC \Rightarrow waGGHABCABC$

5. Se expande la variable más a la izquierda G mediante la regla de producción $G \rightarrow aEB$

   $wa\mathbf{G}GHABCABC \Rightarrow wa\mathbf{aEB}GHABCABC$

6. Se ejecuta la acción del nuevo terminal a y se genera un evento que corresponde a la variable E_b, se selecciona la regla de producción $E \rightarrow E_b$ (aun no se expandirá) y se revisa por supresiones/expansiones.

7. Se inicia el proceso de supresión/extensión y debido a que en el paso anterior se genero un evento, se utilizarán los subconjuntos PSE_{ev_1}, PSE_{ev_4} y PSE_{ev_n} para seleccionar las reglas de producción para el proceso de supresión/extensión de variables.

   a) Debido a que el mayor nivel es n, todas las variables a la derecha de la variable de derivación de eventos, E, son candidatas a supresión o extensión:

   $waE\mathbf{BGHABCABC}$

   b) Para cada variable las posibles reglas de producción a utilizar son:

   B :

   Nivel 1: $B \rightarrow G$

   Nivel N: $B \rightarrow \epsilon$

   G : No hay producciones de supresión/extensión que aplicar.

   H :

   Nivel 1: $H \rightarrow B$ *(posición $>$ nivel = no aplica)

   Nivel 4: $H \rightarrow B$

   A :

   Nivel 4: $A \rightarrow H$

   Nivel N: $A \rightarrow \epsilon$

   B :

   Nivel 1: $B \rightarrow G$ *(posición $>$ nivel = no aplica)

   Nivel N: $B \rightarrow \epsilon$

   C :

   Nivel N: $C \rightarrow \epsilon$

   A :

   Nivel 4: $A \rightarrow H$ *(posición $>$ nivel = no aplica)

Nivel N: $A \rightarrow \epsilon$

B :

Nivel 1: $B \rightarrow G$ *(posición $>$ nivel = no aplica)

Nivel N: $B \rightarrow \epsilon$

C :

Nivel N: $C \rightarrow \epsilon$

c) Para aquellas variables que estén dentro del rango de su producción de más bajo nivel, esa será la producción a expandir:

B :$B \rightarrow G$

G : No se suprime ni extiende, permanece sin cambio.

H : $H \rightarrow B$

A : $A \rightarrow H$

B : $B \rightarrow \epsilon$

C : $C \rightarrow \epsilon$

A : $A \rightarrow \epsilon$

B : $B \rightarrow \epsilon$

C : $C \rightarrow \epsilon$

d) Se expanden (suprimen o extienden) las variables para las cuales se encontraron reglas de producción y que son validas:

$waE\mathbf{BGHABCABC} \Rightarrow waE\mathbf{GGBH}$

8. Se expande la variable de derivación de eventos y se continua con el comportamiento.

$wa\mathbf{E}GGBH \Rightarrow wa\mathbf{E_b}GGBH$

Para una variable especifica no se puede realizar supresión y extensión a la vez, ya que por definición en un subconjunto no debe haber más de una regla cuya cabeza de producción se la misma que otra regla que pertenezca al mismo subconjunto y de entre los diversos subconjuntos de supresión/extensión se da prioridad a las reglas del subconjunto cuya nivel (rango) sea menor, siempre y cuando la variable a expandir este dentro del rango.

Al terminar el proceso de supresión/extensión de variables se regresará a realizar una derivación más a la izquierda, la cual corresponderá a extender la variable de derivación de eventos que mediante la regla que le corresponda dependiendo si se genero evento o no se genero evento.

6.3. Flujo/Algoritmo de la gramática de comportamientos

A continuación se describe el flujo general de una gramática de comportamientos, en la figura 10 se puede observar el algoritmo que describe este flujo.

Para una gramática de comportamientos $GC = (V, \Sigma, P, S)$

1. Sea el estado de decisión actual el símbolo de inicio de la gramática.
2. Se selecciona una acción para el estado de decisión actual y se expande la variable que representa el estado/ evento actual utilizando dicha regla cuya variable en el cuerpo representa una acción compleja.
3. Se expande la variable de acción mediante su producción correspondiente, la cual insertará un nuevo símbolo terminal en la cadena.
4. Se ejecuta la acción básica correspondiente al nuevo símbolo terminal que se agregó a la cadena por medio de la derivación/expansión de la variable de acción.
5. Se revisa si se generó un evento y se selecciona la producción con la cual se expandirá la variable de derivación de eventos, pero no se expande aún.
    - Si se generó un evento, el cuerpo de la producción seleccionada será una variable que corresponde a un evento.

Algoritmo Flujo de la gramática de comportamientos.

1. Inicio
2. Selección de acciones
3. Ejecución de acción compleja
4. Ejecución de acción básica
5. Se 'revisa' si se generó un evento
6. Supresión y/o extensión de variables
7. Derivación de eventos. El evento es válido?
   Si. ↑ 2
   No. ↓ 8
8. Derivación más a la izquierda. Producción de acción?
   Si. ↑ 4
   No. ↑ 8

Figura 10: Flujo general de una gramática de comportamientos.

- Si no se generó un evento o si se usa la modalidad de omitir y si se repite un evento, el cuerpo de la regla de producción corresponderá a la cadena vacía.

6. Si se cumplen las condiciones se realiza el proceso de supresión/expansión de variables.

7. Se expande la variable de derivación de eventos.

   a) El evento generado fue valido, es decir, si el cuerpo de la regla de producción utilizada corresponde a una variable que representa un evento, éste sera el nuevo estado actual. ↑ 2

   b) El cuerpo de la regla de producción utilizada corresponde la cadena vacía. ↓ 8

8. Se continua con un proceso de derivación más a la izquierda.

   a) Se expande una producción de acción, es decir aparece un nuevo símbolo terminal. ↑ 4

   b) Se expande algún otro tipo de producción. ↑ 8

La terminación no se incluye en el flujo general ya ésta puede realizarse de distintas formas y en distintas fases del flujo. El proceso de terminación se describe en la siguiente sección.

6.4. Formas de terminación de una GC

Al ejecutarse un comportamiento mediante una gramática de comportamientos se tiene que considerar la terminación de este, de entrada se consideran varios casos por medio de los cuales se puede terminar un comportamiento.

Dos formas en que se puede terminar un comportamiento y que son independientes de una GC son:

Terminación por objetivo :

Una forma lógica de terminar un comportamiento es simplemente hacer que el agente detenga sus funciones una vez que alcance su objetivo.

Terminación por tiempo :

Se especifica un limite de tiempo y se detiene al agente cuando se supere dicho limite.

Puede haber más maneras de terminar el comportamiento, en este texto se contemplarán las dos formas ya mencionadas, con las cuales puede trabajar una GC.

Estos tipos de terminación se pueden utilizar de forma más natural en la gramática si se manejan como eventos, que se generen cuando, se encuentre el objetivo o se acabe el tiempo permitido de ejecución. Para dichos eventos se puede incluir una acción compleja que derive a un terminal cuya acción básica consista en inhibir o desactivar al agente o, por que no, empezar un nuevo comportamiento.

Se incluirá una forma nativa de una GC con la cual terminar un comportamiento:

Terminación por derivación :

Mediante los tipos de terminación mencionados anteriormente existe la posibilidad de finalizar el comportamiento mediante la detención de las funciones del agente, pero, dependiendo del diseño de la GC, existe la posibilidad de que el proceso de derivación quede inconcluso.

Por otro lado, existe la posibilidad de que el proceso de derivación termine, es decir que ya no queden variables que expandir y que la cadena conste solamente de símbolos terminales. A esto se le llamará terminación por derivación completa.

La utilización de acciones que estén compuestas de reglas de producción con propiedades recursivas puede ayudar a garantizar, hasta cierto punto, que el proceso de derivación no termine antes de que el agente cumpla su objetivo.

En el caso de que el agente termine y la derivación quede inconclusa o en el caso especial donde se terminen las variables y no se haya alcanzado el objetivo, son puntos que se dejarán a criterio del diseñador y los cuales pueden ser influenciados por diversos factores como la naturaleza del problema, la arquitectura del agente, eficiencia en tiempo, etc.

6.5. Aprendizaje y Diseño

En lo que resta de este capítulo se revisarán varias formas por medio de las cuales se puede llevar a cabo la selección de acciones. Los dos métodos que se presentarán harán énfasis en la selección de una sola acción por cada estado (evento) que se defina en la GC.

Para un estado (evento) E_x para el cual se puedan realizar n número de acciones,

$$E_x \rightarrow \ A_1|A_2|A_3|...|A_n$$

se buscará que el agente aprenda a seleccionar, para dicho estado, siempre la misma acción A_i, lo cual si se logra, da la posibilidad de remover de la gramática aquellas reglas de producción que representen las demás posibles acciones para dicho estado, es decir las reglas de producción cuya cabeza es E_x y cuyo cuerpo no es A_i, dejando solo una producción para E_x que tenga como cuerpo A_i.

$$E_x \rightarrow \ A_i$$

Y si esto se hace para cada estado de decisión, entonces se podrá realizar la selección de acciones de forma automática.

En las siguientes secciones se revisarán métodos de aprendizaje los cuales se utilizaron para realizar este procedimiento, haciendo referencia a lo que en el marco teórico se explicó como una política de control, la cual al estar en un estado recomienda una sola acción, la acción óptima, es decir la mejor acción a tomar.

Cabe mencionar que no es la única forma de lograr esto, sino que es simplemente la que se utilizó en este trabajo; por ejemplo en lugar de utilizar una política que indique solo una acción, se puede utilizar algún otro método que para distintas instancias de tiempo recomiende acciones distintas para un mismo estado, o inclusive que selecciones acciones de forma aleatoria.

6.5.1. Aprendizaje por refuerzo para una gramática de comportamiento

El aprendizaje por refuerzo, como ya se reviso en secciones previas, busca que un agente aprenda una política óptima, la cual para algún estado s nos recomiende una acción a la cual le ayude al agente cumplir con su objetivo.

Para realizar este proceso se realizará un mapeo entre los elementos de la gramática de comportamientos y los de la técnica de aprendizaje por refuerzo. Una vez que se logren identificar los estados, acciones y la política de control para una gramática de comportamientos, el proceso de aprendizaje por refuerzo se puede realizar siguiendo los algoritmos estándar.

Se tiene que tomar en cuenta que las transiciones, utilidades, así como las funciones de aprendizaje, tienden a variar dependiendo del problema.

6.5.1.1. Estados y acciones

Para una gramática de comportamientos $\mathbf{GC} = (V, \Sigma, P, S)$

y recordando que

$V = \{VE_a, VE_d, VA_c, V_{aux}\}$

$P = \{PE_a, PE_d, PA_c, P_{aux}, P_{extra_1}, P_{extra_n}\}$

Para PE_a las producciones tienen la forma

$E_x \rightarrow \ A_i$

donde

E_x es una variable que representa un evento, y pertenece al subconjunto VE_a y A_i es una variable que representa una acción, la cual pertenece al subconjunto VA_c.

y al estar en la forma de sentencia:

$wE_x\beta$

siendo w una cadena de símbolos terminales y β una cadena de variables. w y β pueden ser también cadenas vacías.

Se puede decir que se esta en un estado de decisión E_x, desde el cual se puede ejecutar n número de acciones, donde n es el número de reglas de producción en el subconjunto PE_a para las cuales E_x es la variable de la cabeza de la regla de producción.

- $VE_a = \{E_1, E_2, E_3, E_a, ..., E_m\}$ representa el subconjunto de los m número de **estados/eventos** posibles en los que se puede encontrar el agente de la gramática de comportamientos.
- $VA = \{A_1, A_2, A_3, ..., A_n\}$ representa el subconjunto de las n posibles **acciones** que puede realizar el agente de la gramática de comportamientos.
- $PE_a = \{E_x \rightarrow \ A_x, \ \ E_x \rightarrow \ A_y, \ \ E_x \rightarrow \ A_z, \ \ ...\}$ representa las posibles acciones a realizar en los diferentes estados, siendo la cabeza de las producciones el estado a representar y el cuerpo de las producciones las posibles acciones a realizar para el correspondiente estado.

Entonces ya se tienen los componentes básicos para llevar acabo el proceso de aprendizaje, los cuales son los subconjuntos de estados VE_a, el subconjunto de acciones VA_c y el subconjunto de reglas de producción PE_a que describe la relación de estados y acciones.

6.5.1.2. Política de control

Lo que buscará aprender el agente es una política π la cual indique que acción A_i realizar al encontrarse en un estado de decisión E_j, es decir, al encontrarse en forma de sentencia:

$wE_j\beta$

a la cual se le llamará estado E_j, en el cual es posible realizar las x acciones siguientes:

$E_j \rightarrow\ A_1 \mid A_2 \mid A_3 \ldots A_x$ (notación compacta para producciones)

entonces buscaremos una política tal que $i_j = \pi(E_j), (\forall E)$

Para las reglas de producción donde la cabeza de la producción es E_j, i_j señala cual de dichas reglas es la que se usará para la derivación.

$E_j \rightarrow\ A_{i_j}$

con la cual se expande E_j

$wE_j \Rightarrow\ wA_{i_j}\beta$

6.5.2. Algoritmo genético para una gramática de comportamiento

Extendiendo un poco lo que se reviso en la sección pasada, entonces es posible construir una sencilla estructura de datos que represente a la política de control. Similar a lo mencionado en la sección pasada, dependiendo del problema, los parámetros y operadores pueden variar.

6.5.2.1. Individuo

En esta sección se presentará una forma para representar un individuo para un algoritmo genético. El individuo será una política de control para un agente de una gramática de comportamientos. La representación que se utilizará para el individuo será una cadena de números enteros.

Entonces lo que se buscará evolucionar es una política de control π.

Para una gramática de comportamientos $\mathbf{GC} = (V, \Sigma, P, S)$.

Si se tienen m números de estados

$VE_a = E_1, E_2, E_3, ...E_m$

el individuo tendrá m números de genes, donde cada gen representará uno de los posibles estados en los que se puede encontrar el agente

y para un estado E_j, el cual tiene n posibles acciones a realizar

$E_j \rightarrow \ A_1 \mid A_2 \mid A_3 \ ... \ A_n$

Por lo cual, los posibles valores que puede tener el gen en la posición j están en el rango [$1, x$].

entonces se buscará aquel individuo tal que

$i_j = \pi(E_j), (\forall E)$

es decir, aquel individuo que para cada estado indique la mejor acción a tomar.

Capítulo 7. Gramaticas experimentales

En este capítulo se presentarán los experimentos diseñados e implementados, así como los resultados obtenidos. El orden en que se presentará este capítulo es el siguiente:

Prototipos: Se introducirán los prototipos diseñados, los cuales ayudarán a facilitar las posibilidades de la utilización de lenguajes y gramáticas para la generación de comportamientos sencillos.

Problema de la hormiga artificial: Se definirá el problema de la hormiga artificial y la instancia a resolver, el trayecto de Santa Fe.

Notación: Se revisará la notación utilizada en los experimentos, cuya representación difiere un poco a lo visto en las secciones anteriores, pero sustancialmente expresa lo mismo.

Gramáticas de comportamientos para el trayecto de Santa Fe: Se presentarán tres gramáticas de comportamientos para resolver el trayecto de Santa Fe. Para la gramática de comportamientos #1 (GC # 1) se presentará:

- Diseño general de la gramática
- Aprendizaje de selección de producciones
- Resultados

Las GC # 2 y # 3 son extensiones de la GC # 1, modificadas de tal forma que ayuden a demostrar propiedades no presentes en la GC # 1.

7.1. Gramática prototipo #1

En los comienzos de este trabajo y antes de que se planteara de manera formal la estructura de una gramática de comportamientos, se diseño una pequeña gramática independiente de contexto con el fin de plantear una hipótesis que expusiera la factibilidad de la utilización de gramáticas como medio para describir y generar comportamientos en agentes artificiales.

El objetivo del agente basado en esta primera gramática prototipo consistía simplemente en avanzar hasta encontrar una pared y al encontrarla girar 90 grados hacia la derecha.

La gramática diseñada para este primer prototipo fue la siguiente:

$G = (V, \Sigma, P, \{S\})$

- $V = \{S, M, W, R\}$
- $\Sigma = \{m, w, r\}$
- $P = \{$
  1. $S \rightarrow M$
  2. $M \rightarrow mM$
  3. $M \rightarrow W$
  4. $W \rightarrow wR$
  5. $R \rightarrow r$

  $\}$

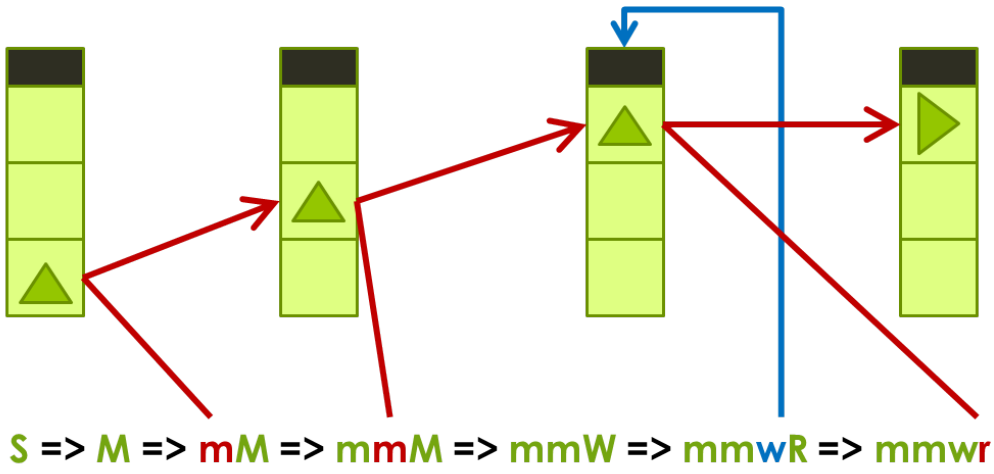


Figura 11: Proceso de derivación para la gramática prototipo # 1.

En esta gramática (ver figura 11) la variable M tiene dos posibles formas de ser expandida, las reglas # 2 y # 3, donde la regla # 3 se utiliza solamente cuando el agente este en una casilla adyacente a la pared y mirando hacia ella, en los demás casos se utiliza la regla # 2.

Cada vez que se expande una variable y aparece un nuevo símbolo terminal el agente realizará una acción que corresponde a dicho símbolo terminal, las cuales son:

m El agente se moverá una casilla hacia el frente.

r El agente girará 90 grados hacia la derecha.

w Indica que se genero el evento de estar mirando hacia la pared.

Este primer prototipo se realizo en papel y se utilizó como base para un segundo prototipo.

7.2. Gramática prototipo #2

Después de tener un primer prototipo se buscó verificar la factibilidad ya no simplemente sobre papel, sino que se buscó realizar un pequeño sistema para probarlo y debido a la relativa sencillez del primer prototipo se decidió hacerlo un poco más complejo, solo un poco, para ello se diseño una segunda gramática prototipo. Este prototipo se implemento en la plataforma Java.

El objetivo del agente basado en esta segunda gramática prototipo consistió simplemente en recorrer todas las casillas de un “grid” (rejilla, cuadrícula, zona cuadriculada). El agente (ver figura 12) comenzaba en alguna posición arbitraria mirando hacia la parte de abajo del tablero. Y cada vez que se encuentra un nuevo terminal, siendo que no hay más variables a la izquierda, se ejecuta una acción básica dependiendo del nuevo terminal.

El agente puede realizar 3 acciones básicas:

- Movimiento una unidad hacia el frente.
- Rotación 90 grados hacia la derecha.
- Rotación 90 grados hacia la izquierda.

de forma similar el agente puede detectar un evento:

- Pared, es decir cuando el agente pasa a una casilla de la periferia y se encuentra mirando hacia afuera del “grid”, este evento (pared) se dispara.

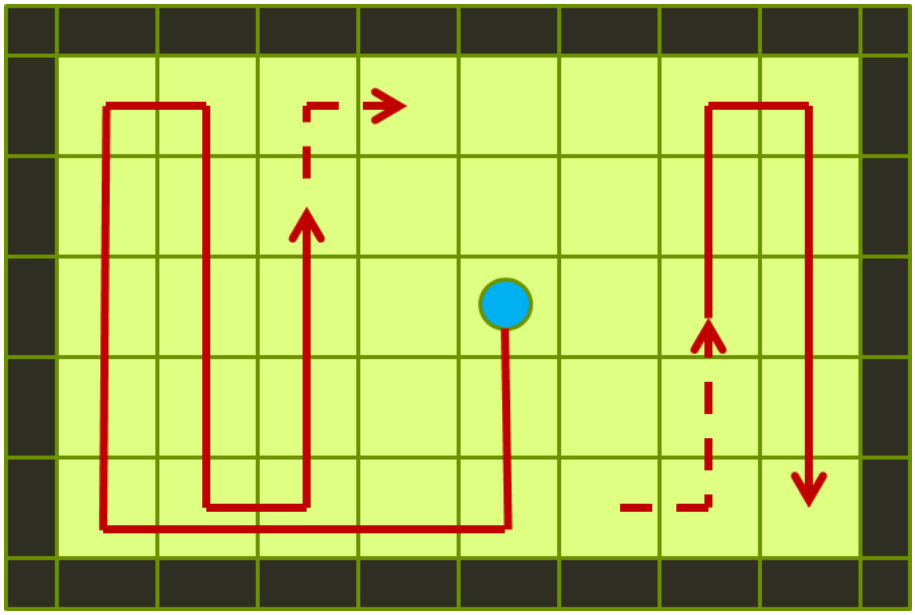


Figura 12: Comportamiento prototipo # 2.

El comportamiento que se buscaba era que el agente se dirigiera a la esquina inferior izquierda del tablero y desde esa casilla comenzara a moverse en zig-zag, recorriendo el tablero de tal forma que pasara por lo menos una vez por cada una de las casillas de la cuadrícula, y una vez que llegara a la ultima casilla se detuviera.

Para lograr esto se diseño una gramática tal que con excepción de dos variables todas las demás variables eran la cabeza de una sola regla de producción. Las dos variables especiales tenían 2 producciones las cuales podían usarse para expandirse dependiendo de si se acababa de disparar el evento de pared o no.

La derivación de esta gramática es una derivación más a la izquierda, la cual se realiza de forma automática ya que la mayoría de las variables tienen solo una posible forma de ser expandidas, mientras que para las demás la regla a expandirse depende de si se generó o no el evento de pared (ver figura 13).

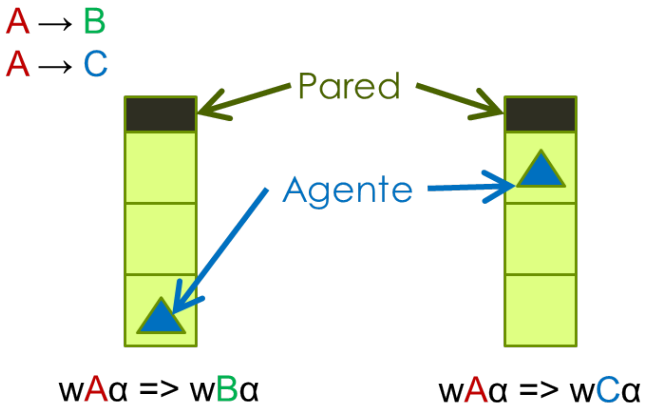


Figura 13: Ejemplo de expansión de reglas de producción para la gramática prototipo # 2.

Ejemplo:

Si se encuentra en una forma de sentencia

$wA\alpha$

y se tienen las siguientes reglas de producción

1. $A \rightarrow B$
2. $A \rightarrow C$

donde w es la cadena de terminales que se han derivado, A es la siguiente variable a expandir, α es una cadena de variables y terminales que aun no se han expandido (variables) o ejecutado (terminales). La regla 1 es la regla por defecto, mientras que la regla 2 esta reservada para cuando se genere el evento de pared.

entonces, la derivación correspondiente será:

- $wA\alpha => wB\alpha$ (no hay pared)
- $wA\alpha => wC\alpha$ (si hay pared)

7.3. Problema de la hormiga artificial

Una vez que se planteó la estructura formal para la gramática de comportamientos se buscó un problema que fuera sencillo, pero que a la vez pudiera demostrar de una forma clara las propiedades de una gramática de comportamientos. El problema que se seleccionó fue el problema de la hormiga artificial.

7.3.1. Definición

El problema consiste en la navegación de una hormiga artificial de tal forma que encuentre toda la comida disponible en un trayecto irregular.

El objetivo de la hormiga artificial es comer toda la comida, recorriendo todo el trayecto, en el menor tiempo posible.

El escenario en el cual actuará la hormiga artificial es una cuadrícula de 32x32 casillas (celdas), el cual se puede observar en la figura 14. La hormiga comienza en la casilla de la esquina superior izquierda (0,0) mirando hacia la derecha (Koza, 1992).

La hormiga artificial es un agente que posee las siguientes limitaciones:

- Solo puede observar la casilla
- Las acciones de la hormiga son limitadas, contará con tres acciones básicas:
  - Movimiento una unidad hacia el frente.
  - Rotación 90 grados hacia la derecha.
  - Rotación 90 grados hacia la izquierda.

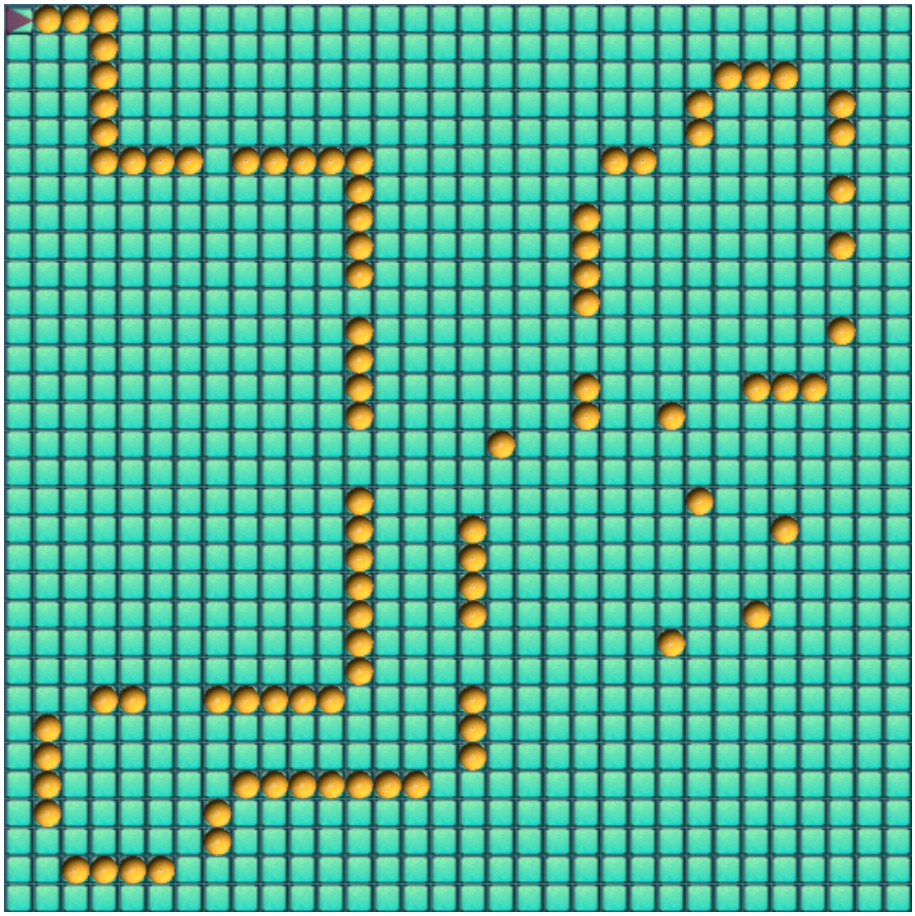
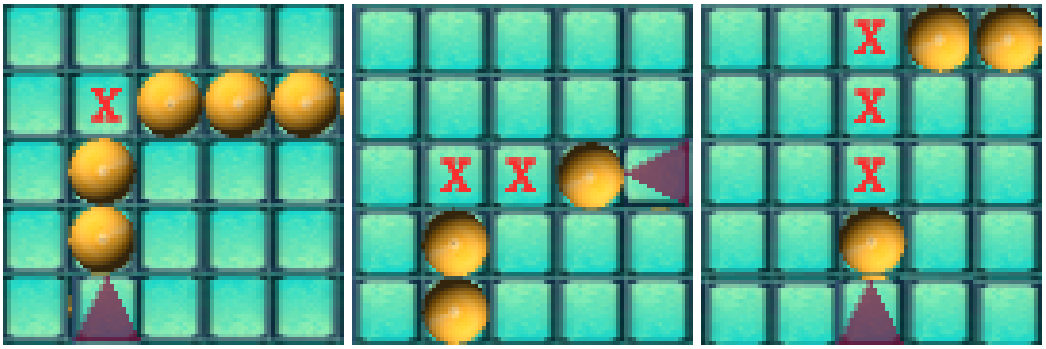


Figura 14: Trayecto de Santa Fe para el problema de la hormiga artificial.

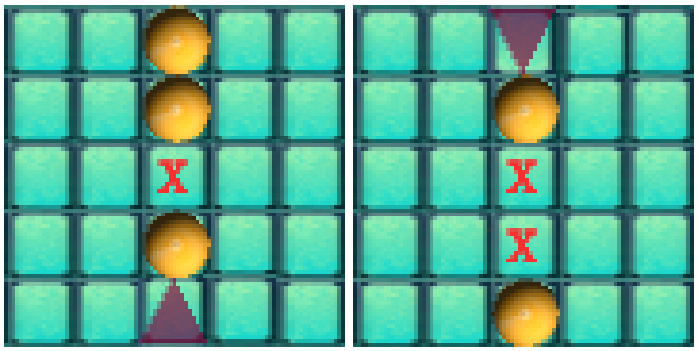
7.3.2. Trayecto de Santa Fe (TSF)

El Trayecto de Santa Fe (figura 14) consta de 89 piezas de comidas, este trayecto no es un trayecto regular, es decir, no es recto ni continuo, y las piezas de comida pueden estar una tras otra o separadas por saltos. Los saltos pueden variar pero siguen ciertos patrones (figura 15), que son:

- Saltos sencillos y dobles.
- Saltos en esquinas. Pueden ser sencillos, dobles (se asemejan a un movimiento de caballo en ajedrez) y triples.



(a) Salto sencillo en esquina (b) Salto doble en esquina (c) Salto triple en esquina



(d) Salto sencillo (e) Salto doble

Figura 15: Tipos de saltos presentes en el trayecto de Santa Fe.

7.4. Notación para los experimentos

Para los experimentos y para dar un poco más de claridad a la lectura de los elementos de la gramática, los símbolos terminales aparecerán entre paréntesis () y las variables entre corchetes []. En la literatura también pueden encontrarse casos en los que se usan otros símbolos para denotar y/o diferenciar entre terminales y variables, por ejemplo $<>$.

En lugar de escribir

$E_x \rightarrow \ aA$

donde

E_x y A son dos variables, y a es un símbolo terminal

se escribirá de la forma

$[E_x] \rightarrow \ (a)[A]$

ya que en ocasiones se utilizará más de un carácter para hacer referencia a variables y/o terminales, lo cual ayudará un poco en la lectura y entendimiento de lo que representa cada símbolo, por ejemplo:

$[EVENTO_x] \rightarrow\ (a)[ACCION_i]$

siendo

$[EVENTO_x]$ y $[ACCION_i]$ dos variables, y

(a) es un símbolo terminal

7.4.1. Producciones, orden y notación compacta

Al hacer referencia al orden de las producciones de alguna variable, dígase '$[E_x]$',el orden se dará de arriba hacia abajo y para notación compacta el orden comenzara de izquierda a derecha comenzando por el cuerpo de producción que se encuentre más a la izquierda y terminando con el que se encuentre más a la derecha.

Ejemplo:

Sean las siguientes producciones escritas en notación compacta:

$[E_x] \rightarrow\ [A_1] \quad | \quad [A_2] \quad | \quad [A_3]$

entonces el orden de las producciones será:

1. $[E_x] \rightarrow\ [A_1]$
2. $[E_x] \rightarrow\ [A_2]$
3. $[E_x] \rightarrow\ [A_3]$

7.5. GC # 1 para el trayecto de Santa Fe

En esta sección se explicará la forma en que utilizando una gramática de comportamientos se puede lograr que la hormiga artificial tenga las herramientas para alcanzar su objetivo, el cual es encontrar todas las piezas de comida del trayecto de Santa Fe.

La forma en que actuará esta hormiga será realizando un proceso de derivación para la GC siguiendo el algoritmo de flujo descrito en el capítulo anterior. En la gramática de comportamientos para la hormiga artificial se definirán los eventos a los cuales reaccionará la hormiga. Las acciones que puede usar la hormiga se definirán como reglas de producción, las cuales se expandirán de acuerdo a una política de control.

La política de control indicará que acción utilizar al generarse un evento.

7.5.1. Acciones

La hormiga artificial es capaz de realizar tres acciones básicas, las cuales se representaran en la gramática por medio de símbolos terminales.

Al realizarse el proceso de derivación de la gramática se irán expandiendo las variables. Cuando por medio del proceso de derivación se agregue un nuevo símbolo terminal a la cadena, se ejecutará la acción básica que le corresponda a dicho símbolo terminal.

Acciones básicas: Son las acciones básicas que puede realizar el agente (la hormiga artificial), representadas por los símbolos terminales de la gramática.

- (f) Movimiento de una unidad hacia el frente.
- (r) Rotación 90 grados hacia la derecha.
- (l) Rotación 90 grados hacia la izquierda.

Acciones complejas: Acciones formadas a partir de acciones básicas. Forman parte del subconjunto PA. Cuando se genere un evento se seleccionará alguna de estas reglas. Para esta gramática tiene una relación directa con las acciones básicas, por lo cual se realizará una supresión de nivel 1. En secciones posteriores se modificarán un poco las reglas para transformar el comportamiento de la hormiga artificial.

- $[ROT\_RIGHT] \rightarrow\ (r)[EVENTO][ROT\_RIGHT]$

  Rotación 90 grados hacia la derecha.

- $[ROT\_LEFT] \rightarrow\ (l)[EVENTO][ROT\_LEFT]$

  Rotación 90 grados hacia la izquierda.

- $[FORWARD] \rightarrow\ (f)[EVENTO][FORWARD]$

  Movimiento de una unidad hacia el frente.

7.5.1.1. Simplificación de acciones por supresión

Para esta gramática se incluirá un subconjunto de supresión de variables que actuará al generarse algún evento, el nivel de supresión sera 1. El subconjunto de supresión PSE_{ev_1} se definirá de la siguiente forma:

$$PSE_{ev_1} = \{[ROT\_RIGHT] \rightarrow\ \epsilon, [ROT\_LEFT] \rightarrow\ \epsilon, [FORWARD] \rightarrow\ \epsilon\ \}$$

Por medio de este proceso se suprimirá (eliminará) la variable que se encuentra más a la derecha de los cuerpos de las acciones complejas. El resultado de esto será un mapeo casi directo de un acción compleja a una acción básica, es decir, al ejecutar una acción compleja esta se expandirá y ejecutará solo una acción básica sin hacer nada más complejo.

En estos experimentos no se utilizan subconjuntos de supresión/extensión del tipo PSE_{nul_n}.

Otra forma de realizar esto es por medio de una variable auxiliar que se expanda a $epsilon$ (la cadena vacía), pero para fines didácticos se utilizará supresión de variables.

7.5.2. Eventos

A continuación se muestran los distintos eventos usados para el problema de la hormiga artificial, así como las posibles acciones que se pueden usar al estar en dicho evento. Estas son las reglas que se encuentran en el subconjunto de producciones PE.

1. $[START] \rightarrow\ [ROT\_RIGHT] \quad | \quad [ROT\_LEFT] \quad | \quad [FORWARD]$

El primer evento, se genera al iniciarse el comportamiento.

2. $[ROT\_360] \rightarrow [ROT\_RIGHT] \quad | \quad [ROT\_LEFT] \quad | \quad [FORWARD]$

   El agente ha rotado 360 grados en la misma dirección sin haber tomado alguna otra acción.

3. $[WALL\_INFRONT] \rightarrow [ROT\_RIGHT] \quad | \quad [ROT\_LEFT]$

   El agente se encuentra en una celda de la orilla y mirando hacia afuera del tablero.

4. $[FOOD] \rightarrow [ROT\_RIGHT] \quad | \quad [ROT\_LEFT] \quad | \quad [FORWARD]$

   En la celda que se encuentra justo enfrente del agente hay comida.

5. $[NO\_FOOD] \rightarrow [ROT\_RIGHT] \quad | \quad [ROT\_LEFT] \quad | \quad [FORWARD]$

   En la celda que se encuentra justo enfrente del agente **no** hay comida.

7.5.3. Derivación de eventos

Para la derivación de eventos se define la variable de derivación de eventos de la siguiente forma:

$$VE_d = \{[EVENTO]\}$$

Para esta gramática de comportamientos, cada vez que se ejecute un evento básico se generará un evento, no se incluirá el caso en que la variable de derivación de eventos se expanda a cadena vacía, por lo cual las reglas de producción para expandir la variable de derivación de eventos incluirán solo aquellas que deriven variables de eventos. Dichas reglas de producción serán definidas en el subconjunto PE_d como se muestra a continuación:

$$PE_d = \{[EVENTO] \rightarrow [START]|[ROT\_360]|[WALL\_INFRONT]|[FOOD]|[NO\_FOOD]\}$$

7.5.4. Prioridad de eventos

En el caso de que se llegasen a generar dos o más eventos a la vez, solo uno de ellos se derivará, esto por la definición la variable de derivación de eventos, para esta GC se le dará a los eventos el siguiente orden de prioridad:

1. $[START]$
2. $[WALL\_INFRONT]$
3. $[FOOD]$
4. $[ROT\_360]$
5. $[NO\_FOOD]$

7.5.5. Gramática completa

La gramática de comportamientos completa para la hormiga artificial se puede ver en el apéndice(A.2).

En las siguientes secciones se revisarán dos métodos por medio de los cuales la hormiga artificial aprenderá una política de control óptima, la cual la guiará hacia todas las piezas de comida que se encuentren en el trayecto de Santa Fe.

7.6. GC con aprendizaje por refuerzo para resolver el trayecto de Santa Fe

7.6.1. Transiciones y tabla de frecuencias

Para este problema se tratará con una función de transición no determinística, esto es, al aplicar una acción desde un estado, el resultado no siempre sera el mismo, en otras palabras, se tiene una función de transiciones probabilista similar a:

$Pr = (s, a, s')$

donde

Pr es la probabilidad de pasar al estado s' al utilizar la acción a desde el estado s.

Esta función sera desconocida para el agente, y para tratar de aproximar esta función se utilizará una tabla de frecuencias (TF), la cual se actualizará de la siguiente forma:

Al encontrarse el agente en un estado s y aplicar la acción a, se observará el nuevo estado s'. Entonces:

$TF[s, a, s'] = TF[s, a, s'] + 1$

donde

s es un estado en el cual se selecciona una acción, a es la acción seleccionada en s y s' es el estado resultante de seleccionar la acción a en el estado s; es decir, cada vez que se pase del estado s al s' utilizando la acción a se sumará 1 a la entrada de la tabla que corresponde a dicha transición.

7.6.2. Utilidades

Así como las transiciones entre estados, la función de utilidad también se presenta de una forma no determinista, y de forma similar, se utiliza una tabla de utilidades (TU), que se actualiza de manera similar a la tabla de transiciones de estados, la diferencia radica en que se actualiza si y solo si en el nuevo estado s' el agente encuentra comida.

$TU[s_t, a_t, s'_{t+1}] \leftarrow TU[s_t, a_t, s'_{t+1}] + C_{t+1}$, si y solo si s'_{t+1} contiene comida

de lo contrario

$TU[s_t, a_t, s'_{t+1}] \leftarrow TU[s_t, a_t, s'_{t+1}] - 0.5$

donde

C_{t+1} es el número de casillas con comida que el agente ha encontrado hasta el momento en el tiempo $t + 1$.

7.6.3. Función de Aprendizaje Q

Como lo que buscamos es una política que nos indique la mejor acción para cada estado, usaremos una función que al utilizar las tablas de frecuencia y utilidades nos recomiende la acción óptima para cada estado. Para ello se utilizará una tabla, la cual se actualiza de la siguiente forma :

$Q[s, a] \leftarrow updateQValues(s, a), \quad \forall$ (par estado-acción (s, a))

donde

Algoritmo $updateQValues(s, a)$

FUNCION - $updateQValues(s, a)$
ENTRADAS: par estado-acción(s,a)
var $n \leftarrow 0$
var $total \leftarrow$ (total de veces que se ha repetido el par(s, a))
var $count \leftarrow 0$
Repetir para todos los eventos s'
  var $c \leftarrow TF[s, a, s']$
  si $c = 0$ entonces
    $n \leftarrow n + (UT[s, a, s'] * (c/total))$
    $count \leftarrow count + 1$
si $count > 0$ entonces
  SALIDA: $(n/count)$
si $count \leqslant 0$ entonces
  SALIDA: 0

Figura 16: Función auxiliar para la actualización de la tabla Q.

$updateQValues(s, a)$ es la llamada a una función la cual se describe en el algoritmo que se muestra en la figura 16.

Al actualizar la tabla Q es posible obtener de una forma sencilla la política optima, la cual para un estado s, se obtiene mediante la siguiente ecuación:

$$\pi^*(s) \leftarrow \max_a Q[s, a] \quad \forall\ (a) \text{ de } \quad \forall\ (s)$$

7.6.4. Parámetros

A continuación se describen los parámetros del algoritmo, así como los valores que se utilizaron para el experimento.

Épocas: 10 El número de veces que se repetirá el recorrido. Se puede decir que es el número de oportunidades que tiene el agente para aprender de forma correcta la política.

Tiempo limite: 400 El tiempo que tiene el agente para tratar de encontrar toda la comida. En realidad no es tiempo, son el número de acciones básicas que se le permite realizar al agente antes de concluir la época.

7.7. GC con algoritmo genético para resolver el trayecto de Santa Fe

La configuración utilizada para los componentes del algoritmo genético (simple) para el problema de la hormiga artificial, con gramática de comportamiento, se describe a continuación:

7.7.1. Población de hipótesis (soluciones)

Para este problema un individuo del algoritmo genético representará una política de control para el agente (la hormiga artificial). La política de control indicará acciones a seguir en respuesta a los diferentes eventos que perciba la hormiga.

La primera población de individuos será generada de forma aleatoria.

7.7.1.1. Representación

La forma que se utilizará para representar la política de control será una cadena de números enteros. La cadena tiene una equivalencia con un gen, y tendrá una longitud equivalente al numero de eventos, donde cada posición de la cadena equivaldrá a un alelo. Cada alelo de la cadena indicará un evento, mientras que el valor numérico indicará la acción a realizar al generarse dicho evento.

Ejemplo:

Sea una cadena de números enteros

1 3 2 3 2

Entonces el primer alelo, que es la primera posición de la cadena, representa el primer evento para el cual se tiene las reglas de producción:

1. $[START] \rightarrow [ROT\_RIGHT]$
2. $[START] \rightarrow [ROT\_LEFT]$
3. $[START] \rightarrow [FORWARD]$

que se puede escribir de la siguiente forma en notación compacta:

$$[START] \rightarrow [ROT\_RIGHT] \quad | \quad [ROT\_LEFT] \quad | \quad [FORWARD]$$

El nombre del evento es $'[START]'$ que es el evento que se genera al comenzar el comportamiento, para este evento existen tres posibles acciones a realizar que son indicadas por las reglas de producción cuya cabeza es la variable $'[START]'$ y que de acuerdo con la cadena que representa la política de control, cuando se genere este evento se ejecutara la acción número **1**, es decir, la regla de producción que se expandirá es:

1. $[START] \rightarrow [ROT\_RIGHT]$

Esto se puede interpretar como:

'Al iniciar el comportamiento la hormiga debe girar 90 grados hacia la derecha!'

7.7.1.2. Parámetros

Los valores de los parámetros del algoritmo genético se muestran a continuación:

Total de generaciones: 15 Una vez evaluadas 15 generaciones y si no se encontró al individuo óptimo el algoritmo se detendrá. La primera población será generada al azar, y al ser evaluada se comenzará con el proceso de evolución.

Tamaño de la población: 12 Serán 12 individuos los que formen la población. En la primera generación estos individuos serán generados al azar, al terminar la primera generación y en adelante se generarán 10 hijos como máximo (dependiendo de la probabilidad de cruce), mientras que los últimos dos serán los mejores individuos de la generación que acaba de ser evaluada.

Probabilidad de Cruce: 0.9 Al aplicar el operador de cruce existe 90 % de probabilidad de que se generen dos nuevos individuos (hijos).

Probabilidad de mutación: 0.1 10 % de probabilidad de mutación de un alelo, donde la probabilidad de que un alelo mute es independiente de los demás alelos.

7.7.2. Función objetivo

El criterio de evaluación que se utilizará para evaluar los individuos será el total de piezas de comida que el agente encuentre en el limite de tiempo. Los posibles valores que se pueden obtener están en el rango [0,89]. Si se encuentra una política que obtenga un valor de 89 (el valor máximo) entonces dicha política sera la política óptima.

7.7.3. Operadores genéticos

A continuación se describen los operadores de cruce y mutación utilizados.

Cruce: Para el operador genético de cruce se realizará un cruce de un punto, para ello se tomaran dos padres y se generaran dos hijos, cada uno de los cuales tendrá como estructura genética la concatenación de dos subcadenas las cuales provienen una de cada padre. Dependiendo de la probabilidad de cruce puede haber o no nuevas cadenas hijas, en caso de que no se aplique para un determinado par de padres, estos pasarán a la siguiente generación.

Mutación: El operador de mutación causará cambios a las posiciones (números) de la cadena de acuerdo a la probabilidad de mutación. Cada posición de la cadena estará sujeta a dicho proceso, independientemente de cualquier otra posición.

7.7.4. Mecanismo de selección

El mecanismo en que se seleccionarán los individuos candidatos a reproducirse y/o a pasar a la siguiente generación es el siguiente:

- Se seleccionarán los candidatos mediante un proceso de ruleta, donde aquellos más aptos (fueron capaces de encontrar más comida) tendrán más probabilidades de ser seleccionados.
- Los dos mejores individuos tienen su pase garantizado a la siguiente generación.

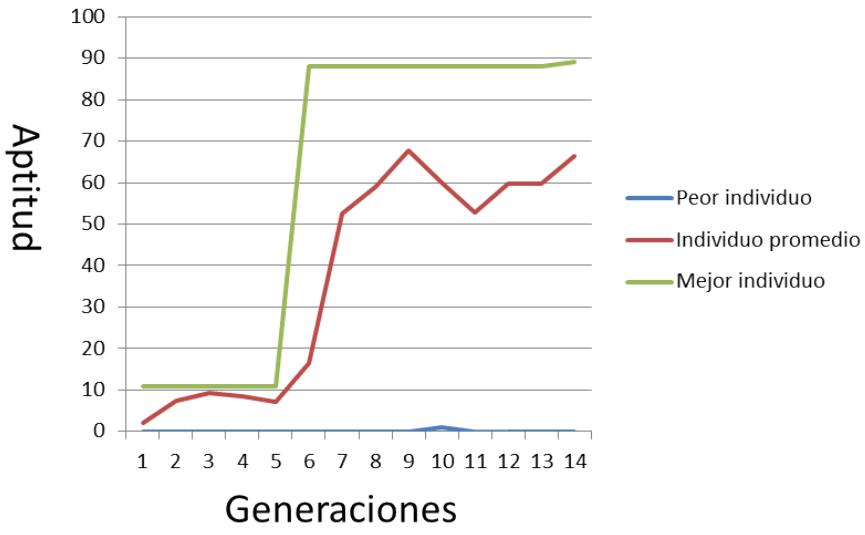


Figura 17: Gráfica que muestra la mejora de las soluciones de una corrida del algoritmo genético.

7.7.4.1. Una gráfica para el algoritmo genético

Para una corrida del algoritmo genético utilizado en esta sección se presenta la gráfica 17 en la que se puede observar como van mejorando los individuos, es decir las posibles políticas de control. En la gráfica se observa la aptitud de el peor individuo de cada generación, la aptitud del mejor individuo y la aptitud promedio de todos los individuos de una generación.

7.8. Resultados: producciones de las políticas de control aprendidas

En esta sección se presentará más a detalle como pasar de la representación numérica a las reglas de producción que la política de control señala.

La política encontrada por los dos métodos utilizados (aprendizaje por refuerzo y el algoritmo genético) fue la misma y esta guía a la hormiga artificial a encontrar las 89 piezas de comida del trayecto de Santa Fe. La política esta representada por una cadena de números enteros, donde cada número representa el indice de la acción optima para algún estado.

3 3 1 3 1

7.8.1. Mapeo de la política numérica a las reglas de producción

Para una regla de producción que pertenece al subconjunto PE, de la forma:

$[E] \rightarrow \ [A_1] \quad | \quad [A_2] \quad | \quad [A_3] \quad | \quad ... \quad [A_x]$

y teniendo la política:

$\pi^*([E]) = n$

donde

n indica que la $n-$esima es la 'acción' óptima a seguir para dicho evento $[E]$

entonces se puede reescribir la política de control que aprendió el agente (**3 3 1 3 1**) de la siguiente manera:

- $\pi^*([START]) = 3$
- $\pi^*([ROT\_360]) = 3$
- $\pi^*([WALL\_INFRONT]) = 1$
- $\pi^*([EVENT_4]) = 3$
- $\pi^*([NO\_FOOD]) = 1$

Agregando unas cuantas etiquetas a las reglas de producción del subconjunto PE, donde las cabezas de producción representan los eventos, y sus cuerpos de producción son las posibles acciones que se pueden realizar desde dicho evento.

$[START]$:

1. $[START] \rightarrow [ROT\_RIGHT]$
2. $[START] \rightarrow [ROT\_LEFT]$
3. $[START] \rightarrow [FORWARD]$

$[ROT\_360]$:

1. $[ROT\_360] \rightarrow [ROT\_RIGHT]$
2. $[ROT\_360] \rightarrow [ROT\_LEFT]$
3. $[ROT\_360] \rightarrow [FORWARD]$

$[WALL\_INFRONT]$:

1. $[WALL\_INFRONT] \rightarrow [ROT\_RIGHT]$
2. $[WALL\_INFRONT] \rightarrow [ROT\_LEFT]$

$[FOOD]$:

1. $[FOOD] \rightarrow [ROT\_RIGHT]$
2. $[FOOD] \rightarrow [ROT\_LEFT]$
3. $[FOOD] \rightarrow [FORWARD]$

$[NO\_FOOD]$:

1. $[NO\_FOOD] \rightarrow [ROT\_RIGHT]$
2. $[NO\_FOOD] \rightarrow [ROT\_LEFT]$
3. $[NO\_FOOD] \rightarrow [FORWARD]$

Para las varias producciones de los distintos eventos, si se toman las producciones etiquetadas con los valores que señala la política, entonces se tiene que las siguientes producciones son las optimas para resolver el trayecto de Santa Fe:

- $[START] \rightarrow [FORWARD]$
- $[ROT\_360] \rightarrow [FORWARD]$
- $[WALL\_INFRONT] \rightarrow [ROT\_RIGHT]$
- $[FOOD] \rightarrow [FORWARD]$
- $[NO\_FOOD] \rightarrow [ROT\_RIGHT]$

7.9. GC # 2 para el trayecto de Santa Fe

En esta y la siguiente sección se demostrarán algunas de las distintas propiedades que una notación en forma de gramática puede atribuir a la generación de comportamientos. En esta sección se revisara la propiedad de recursividad que poseen las gramáticas.

Con el fin de lograr el objetivo de esta y la siguiente sección se tomará como base la gramática de comportamientos # 1 que se utilizó en las secciones previas para resolver el trayecto de Santa Fe, y se realizarán algunos cambios a sus reglas de producción y a la forma de generar los eventos.

Las dos diferencias más notorias que se tendrán con la GC # 1 serán:

Derivación de eventos :

En esta segunda gramática se realizará una omisión por evento repetido, es decir, al generarse un evento se tomará en cuenta el último evento que había generado, si el nuevo evento es diferente entonces la variable de derivación de eventos será expandida de acuerdo a la regla de producción correspondiente a dicho evento; pero si el nuevo evento es el mismo que el anterior entonces la variable de derivación de eventos se expandirá a ϵ (la cadena vacía). Para ello se incluirá la producción correspondiente a la cadena vacía al subconjunto de producciones PE_d.

$PE_d = \{$

$[EVENTO] \rightarrow [START]||[ROT\_360]||[WALL\_INFRONT]||[FOOD]||[NO\_FOOD]||\boldsymbol{\epsilon}$

$\}$

Eliminación de la supresión de variables :

A diferencia de la GC # 1 en esta gramática no se utilizará la supresión de acciones por lo cual el subconjunto de producciones PSE_{ev_1} se modificará de tal forma que no contenga reglas de producción (también podría decirse que se removerá dicho conjunto).

$PE_{ev_1} = \{\}$

Para la selección de las acciones al generarse un evento se utilizará la misma política que se utilizo para la GC # 1.

7.9.1. Discusión y resultados de aplicar la GC # 2

Cabe aclarar que por la naturaleza de este problema, hormiga artificial en trayecto de Santa Fe, aunque se hayan realizado los cambios mencionados, la hormiga (el agente) logra encontrar toda la comida del trayecto, por lo cual este experimento es exitoso.

La palabra derivada a partir de la GC #1 y la GC #2 es la misma, el cambio entre ambas gramáticas radica en la forma en que se realizan las derivaciones. Para la GC #1 el historial de eventos será mayor y el número de acciones básicas ejecutadas estará directamente relacionado a dichos eventos, en cambio para la gramática # 2 el historial de eventos derivados será menor. Debido a que cuando un evento se repite, la variable de derivación de eventos se expande a cadena vacía, y ya que no hay supresión de variables el proceso continua derivando aquellas variables que va encontrando. Debido a esto, se hace patente la naturaleza recursiva de las producciones de acciones complejas, las cuales en la GC # 1, debido a la supresión, no se expresan recursivamente si no que se comportan como si fuesen simplemente acciones básicas (atómicas).

Aun cuando, para este experimento, las cadenas generadas por las GC # 1 y # 2 son exactamente iguales, sus árboles de derivación no lo son.

7.10. GC # 3 para el trayecto de Santa Fe

El propósito de esta sección es similar al de la anterior: demostrar las distintas propiedades de las gramáticas para la generación de comportamientos en agentes artificiales. En esta ocasión se buscará demostrar las capacidades de 'memoria' que se pueden atribuir a un agente mediante el uso de una gramática.

En esta sección se utilizará una tercera gramática de comportamientos para resolver el trayecto de Santa Fe del problema de la hormiga artificial, pero se agregará un objetivo extra al problema el cual consistirá en, una vez encontrada toda la comida regresar al punto de inicio recorriendo de regreso el trayecto.

Para esta extensión del problema se realizará de igual manera una extensión a la gramática utilizada en las secciones anteriores, se tomará como base la GC #2 y se realizarán algunas modificaciones para generar una tercera GC.

Se añadirá un nuevo símbolo terminal, el cual al ejecutarse inhibirá los sensores de la hormiga, es decir, esta dejará de percibir y generar eventos. Debido a esto a partir de que se ejecute dicha acción básica la variable de derivación de eventos se expandirá siempre a ϵ (la cadena vacía).

Nuevo símbolo terminal - (i): Al ejecutarse su acción básica se inhibirán los sensores del agente (hormiga) y dejará de percibir eventos. Se incluye en Σ.

Nueva variable de evento - $[ALL\_FOOD]$: Este símbolo representa un evento que se generará cuando toda la comida del trayecto haya sido consumida (encontrada). Pertenece al subconjunto de variables VE_a.

Para la GC # 3 se modificarán los siguientes subconjuntos de producciones:

Eventos - Subconjuntos PE_a y PE_d:

Generación de eventos - Subconjunto PE_d:

Se añade a PE_d la regla para derivar el evento $[ALL\_FOOD]$:

$[EVENTO] \rightarrow [ALL\_FOOD]$

Selección de acciones - Subconjunto PE_a:

Se añade a PE_a la regla que representa la acción a realizar al generarse el nuevo evento $[ALL\_FOOD]$:

$[ALL\_FOOD] \rightarrow\ [CANCEL\_EVENTS]$

Acciones complejas - Subconjunto PA_c: A las tres reglas de acciones complejas utilizadas se les añadirá una variable auxiliar en su cuerpo de producción:

- $[ROT\_RIGHT] \rightarrow\ (r)[EVENTO][ROT_RIGHT][L]$
- $[ROT\_LEFT] \rightarrow\ (l)[EVENTO][ROT_LEFT][R]$
- $[FORWARD] \rightarrow\ (f)[EVENTO][FORWARD][F]$

y se agregará una cuarta acción compleja:

- $[END\_EVENTS] \rightarrow\ (i)[EVENTO][TURN\_180]$

  El agente consumió toda la comida

Producciones auxiliares - Subconjunto P_{aux}:

Al conjunto de producciones auxiliares se agregarán varias reglas de producción:

- $[R] \rightarrow\ (r)[EVENTO][NULL]$
- $[L] \rightarrow\ (l)[EVENTO][NULL]$
- $[F] \rightarrow\ (f)[EVENTO][NULL]$
- $[TURN\_180] \rightarrow\ (r)[EVENTO][R]$

  Las variables $[R]$, $[L]$, $[F]$, $[NULL]$ y $[TURN\_180]$ se añaden al subconjunto V_{aux} y $[ALL\_FOOD]$ al subconjunto VE_a.

Supresión de acciones - Subconjunto PSE_{ev_1}:

Se añadirá una regla más al subconjunto de supresión de nivel 1 que se utilizó en la gramática # 2

$[NULL] \rightarrow\ \epsilon$

Caso de extensión especial Por último, se realizará un tipo especial de extensión de variables (proceso similar a la supresión), se añadirán dos reglas de producción de nivel N pero la forma de las reglas será diferente a la descrita en la definición de la GC. Las dos reglas a utilizar se incluirán en el subconjunto PSE_{ev_n} y son :

$[R][R][R][R] \rightarrow\ [NULL][NULL][NULL][NULL]$

y

$[L][L][L][L] \rightarrow\ [NULL][NULL][NULL][NULL]$

Estas dos reglas siguen la forma de producciones de una gramática sensible al contexto (GSC), tipo de gramática que tiene más poder de expresión que una independiente de contexto (GIC), pero son más difíciles de tratar. Por dicho motivo se podría decir que la GC # 3 es una GSC, pero para fines prácticos y para no complicar mucho las cosas, diremos que es una GIC extendida.

7.10.1. Discusión y resultados

Las modificaciones hechas para esta gramática de comportamientos generan el siguiente comportamiento: el agente (la hormiga artificial) recorre el trayecto de forma satisfactoria encontrando toda la comida, y al momento de que encuentra la última pieza de comida inhibe sus sensores, da un giro de 180 grados y avanza en sentido contrario por el mismo trayecto recorrido hasta que ya no tiene más variable que expandir, situación que coincide con la llegada a la posición en la que inició el trayecto.

El papel que juegan las producciones que se cambiaron es el siguiente:

- Cada vez que una de las tres acciones principales se deriva, estas van incorporando variables auxiliares a la derecha de la cadena.
- Una vez que el agente encuentra la última pieza de comida en el trayecto, apaga sus sensores, es decir que deja de percibir eventos, da un giro de 180 grados y comienza a derivar las variables restantes de la cadena.
- Después de que el agente encuentra toda la comida y gira 180 grados, las variables restantes son las variables auxiliares, que fueron concatenadas cada vez que se

derivaba una acción compleja, y son el 'complemento' a dichas acciones complejas, es decir, cada acción compleja que ayudo a llevar al agente al fin del recorrido deja un aporte para que el agente pueda regresar.

- Por la forma en que se introdujeron dichas variables auxiliares a la cadena cuando llegue el momento de que se comiencen a expandir, serán derivadas en un orden inverso al que fueron introducidas, y al ser estas los complementos de las acciones utilizadas llevarán de forma satisfactoria al agente hacia el inicio.

- Las reglas de producción especiales que se mostraron (sensibles al contexto), en si no juegan un papel activo en el recorrido del agente, sino que juegan un papel pasivo a la hora que el agente emprende su regreso. El efecto de estas reglas consiste en simplificar de forma breve el camino de regreso del agente. Cuando el agente gira 360 grados se produce un evento el cual selecciona como su acción optima el moverse hacia enfrente; cuando el agente busca comida este giro es importante, pero de regreso se puede omitir y esto es lo que hacen estas producciones, simplifican el paso de las casillas en las que el agente giro 360 grados cancelando las variables auxiliares que hubieran generado el giro al regresar.

El caso en el que el agente gira 270 grados también puede ser simplificado de forma similar, solo es cuestión de jugar un poco con la construcción de producciones y el nivel de los subconjuntos de supresión/extensión. En este texto no se cubrirá dicho caso, pero aquel lector interesado puede intentarlo por su cuenta.

Capítulo 8. Conclusiones y trabajo futuro

8.1. Conclusiones

Como se revisó en la sección de las gramáticas experimentales es posible diseñar agentes basados en gramáticas de comportamientos, los cuales fueron capaces de realizar comportamientos sencillos de forma satisfactoria. Mas aún, se pudo mostrar que un método lingüístico, en este caso gramáticas formales son un método factible para representar y generar comportamientos sencillos.

Para aquellos casos en los que un agente cuente con varias posibles acciones y que el diseñador (programador) tenga dificultad de decidir cual de las acciones es la más indicada para alguno de los eventos que pueden ocurrir, se demostró que es posible utilizar técnicas de aprendizaje de máquina para asistir con el proceso de derivación de las producciones de la gramática para que el agente pueda alcanzar su objetivo. Para el problema del ejemplo tratado en este trabajo, que fue resolver el trayecto de Santa Fe con una hormiga artificial basada en una gramática de comportamiento, se implementó de forma satisfactoria un sistema, que por medio de las dos diferentes técnicas que fueron aprendizaje por refuerzo y algoritmos genéticos, lograra que el agente (la hormiga artificial) aprendiera cual de las posibles acciones era la ideal para ejecutar, dados los posibles eventos que pudiesen ocurrir.

Este tipo de gramáticas es más apto para sistemas simulados en los que se tenga un mejor control sobre las condiciones del agente y su ambiente, ya que debido a las diferencias entre el mundo real y el simulador hubo algunos problemas para generar comportamientos de forma 100 % satisfactoria en agentes físicos (robots). Los robots, por motivos de calibración en sus sensores y/o actuadores o por defectos en el terreno, tuvieron dificultades para seguir las trayectorias que tenían que seguir. Por ejemplo, uno de los robots (humanoide) con el que se trató de sincronizar una de las gramáticas prototipo, cuando el agente en el monitor se movía una unidad, era el equivalente a que el robot diera unos cuantos pasos hacia el frente, pero por falta de calibración en los actuadores el robot poco a poco se comenzaba a desviar. Otro robot (vehículo) que se planeó usar para probar el trayecto de Santa Fe, no fue satisfactorio por motivo de fallas en el sensor

de giro, el cual debería indicar cuando el agente hubiera girado 360 grados.

La falla en gran parte de los experimentos en el mundo real fue por motivos de calibración. El modelo propuesto no es difícil de implementar, de hecho el sistema de aprendizaje se realizó en el lenguaje de programación C#, sistema del cual se obtuvo la política optima para resolver el trayecto de Santa Fe; para probarlo en el mundo real con un robot con ruedas (vehículo) se pasó el sistema al lenguaje Java para teléfonos inteligentes Android, en el que se implementó la comunicación (Bluetooth) con la computadora del robot (lenguaje NXC), por medio de lo cual se enviaban al sistema (teléfono) las lecturas de los sensores, y del sistema al robot, los mandos para los actuadores (el esquema se puede apreciar en la sección A.3 del apéndice A).

8.2. Trabajo futuro

8.2.1. Selección de acciones

En el presente trabajo se presentó el desarrollo de un sistema donde un agente, al generarse un evento en un ambiente puede decidir entre las posibles distintas acciones que puede realizar. Esto se realizó por medio de las técnicas de aprendizaje por refuerzo y de algoritmos genéticos, por medio de las cuales el agente, basado en gramáticas de comportamientos, aprendió a seleccionar la acción óptima para cumplir con su objetivo, que en este caso fue resolver el trayecto de Santa Fe. Como trabajo futuro en este aspecto se puede buscar utilizar otras técnicas de aprendizaje para la discriminación de acciones a derivar, como por ejemplo redes neuronales.

Otra posible manera de tratar la derivación de acciones es modificando un poco la definición de la gramática de comportamientos. Podría extenderse la gramática con reglas de gramáticas sensibles al contexto o inclusive de gramáticas sin restricciones, lo cual podría introducir más posibilidades a los comportamientos, ya que estos tipos de gramáticas pueden describir más lenguajes que las gramáticas independientes de contexto.

8.2.2. Paralelismo

Uno de los aspectos que no cubre la gramática de comportamientos es la ejecución en paralelo de distintas acciones para los eventos que se generan en el mismo instante

de tiempo. De momento, en este trabajo se trata esto mediante la asignación de prioridades para los eventos antes de la ejecución del comportamiento, la cual no cambiará, y entonces al generarse varios eventos al mismo tiempo, realizar una derivación para aquel que tenga la mayor prioridad. Parte del trabajo que se deja pendiente es cambiar la forma en que se realizan las derivaciones cuando suceden varios eventos en el mismo instante de tiempo. Una solución podría ser el utilizar una prioridad dinámica, cambiando la prioridad mientras se ejecuta el comportamiento; por ejemplo, una vez que se ejecute un evento, cambiar su prioridad o la de algún otro evento. Otra posible solución puede surgir al ejecutar varias derivaciones en paralelo, ya sea dentro de una sola gramática de comportamientos, o se podría tener un agente que esté ejecutando varias gramáticas de comportamientos, las cuales manejarían distintos eventos en paralelo.

8.2.3. Automatización

En el presente trabajo las reglas de producción que representan las acciones complejas fueron diseñadas manualmente, ya que el enfoque de este trabajo es el presentar las gramáticas de comportamientos y ejemplos simples de posibles usos.

Un aspecto muy llamativo, pero que involucra el incremento de complejidad, sería el buscar una forma de automatizar el proceso de diseño, generando de forma automática las reglas de producción que representen acciones complejas.

Lista de referencias

Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., y Qin, Y. (2004). An integrated theory of the mind. *Psychological review*, **111**(4): 1036.

Atkeson, C. G., Hale, J. G., Pollick, F. E., Riley, M., Kotosaka, S., Schaul, S., Shibata, T., Tevatia, G., Ude, A., Vijayakumar, S., *et al.* (2000). Using humanoid robots to study human behavior. *IEEE Intelligent Systems and their applications*, **15**(4): 46–56.

Bakker, B., Linaker, F., y Schmidhuber, J. (2002). Reinforcement learning in partially observable mobile robot domains using unsupervised event extraction. En: *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*. IEEE, Vol. 1, pp. 938–943.

Bakker, B., Zhumatiy, V., Gruener, G., y Schmidhuber, J. (2003). A robot that reinforcement-learns to identify and memorize important previous observations. En: *Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*. IEEE, Vol. 1, pp. 430–435.

Bauckhage, C., Gorman, B., Thurau, C., y Humphrys, M. (2007). Learning human behavior from analyzing activities in virtual environments. *MMI-Interaktiv*, **12**: 3–17.

Brockett, R. W. (1988). On the computer control of movement. En: *Robotics and Automation, 1988. Proceedings., 1988 IEEE International Conference on*. IEEE, pp. 534–540.

Csapo, A., Gilmartin, E., Grizou, J., Han, J., Meena, R., Anastasiou, D., Jokinen, K., y Wilcock, G. (2012). Multimodal conversational interaction with a humanoid robot. En: *Proceedings of 3rd IEEE International Conference on Cognitive Infocommunications (CogInfoCom 2012)*. Citeseer, pp. 667–672.

Dantam, N. y Stilman, M. (2012). The motion grammar: Linguistic perception, planning, and control. *Robotics: Science and Systems VII*, p. 49.

Dantam, N. y Stilman, M. (2013). The motion grammar: Analysis of a linguistic method for robot control. *Robotics, IEEE Transactions on*, **29**(3): 704–718.

Dantam, N., Koine, P., y Stilman, M. (2011). The motion grammar for physical human-robot games. En: *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, pp. 5463–5469.

Dongshu, W., Yusheng, Z., y Wenjie, S. (2011). Behavior-based hierarchical fuzzy control for mobile robot navigation in dynamic environment. En: *Control and Decision Conference (CCDC), 2011 Chinese*. IEEE, pp. 2419–2424.

Er, M. J. y Deng, C. (2005). Obstacle avoidance of a mobile robot using hybrid learning approach. *Industrial Electronics, IEEE Transactions on*, **52**(3): 898–905.

Farchy, A., Barrett, S., MacAlpine, P., y Stone, P. (2013). Humanoid robots learning to walk faster: From the real world to simulation and back. En: *Proceedings of the 2013 international conference on autonomous agents and multi-agent systems*. International Foundation for Autonomous Agents and Multiagent Systems, pp. 39–46.

Gamez, D., Fountas, Z., y Fidjeland, A. K. (2013). A neurally controlled computer game avatar with humanlike behavior. *Computational Intelligence and AI in Games, IEEE Transactions on*, **5**(1): 1–14.

Geisler, B. (2004). Integrated machine learning for behavior modeling in video games. En: *Challenges in game artificial intelligence: papers from the 2004 AAAI workshop. AAAI Press, Menlo Park*. pp. 54–62.

Gu, L. y Su, J. (2006). Humanoid robot behavior learning based on art neural network and cross-modality learning. En: *Advances in Natural Computation*. Springer, pp. 447–450.

Hester, T., Quinlan, M., y Stone, P. (2010). Generalized model learning for reinforcement learning on a humanoid robot. En: *Robotics and Automation (ICRA), 2010 IEEE International Conference on*. IEEE, pp. 2369–2374.

Hopcroft, J., Motwani, R., Ullman, J., y tr, A. (2002). *Introducción a la teoría de autómatas, lenguajes y computación*. Pearson Educación. p. 584.

Hristu-Varsakelis, D., Krishnaprasad, P., Andersson, S., Zhang, F., Sodre, P., y D'Anna, L. (2000). The mdle engine: a software tool for hybrid motion control. Reporte técnico, DTIC Document.

Hristu-Varsakelis, D., Egerstedt, M., y Krishnaprasad, P. S. (2003). On the structural complexity of the motion description language mdle. En: *Decision and Control, 2003. Proceedings. 42nd IEEE Conference on*. IEEE, Vol. 4, pp. 3360–3365.

Koza, J. R. (1992). *Genetic programming: On the programming of computers by means of natural selection*. MIT Press.

Manikonda, V., Krishnaprasad, P. S., y Hendler, J. (1995). A motion description language and a hybrid architecture for motion planning with nonholonomic robots. En: *Robotics and Automation, 1995. Proceedings., 1995 IEEE International Conference on*. IEEE, Vol. 2, pp. 2021–2028.

Marsland, S. (2011). *Machine Learning: An Algorithmic Perspective*. CRC Press.

Martın, F., Canas, J. M., Agüero, C., y Perdices, E. (2010). Behavior-based iterative component architecture for robotic applications with the nao humanoid. En: *XI Workshop de Agentes Fısicos. Valencia (Spain)*.

Mitchell, T. M. (1997). Machine learning. 1997. *Burr Ridge, IL: McGraw Hill*, **45**.

Nikolaidis, S. y Shah, J. (2013). Human-robot cross-training: Computational formulation, modeling and evaluation of a human team training strategy. En: *Proceedings of the 8th ACM/IEEE international conference on Human-robot interaction*. IEEE Press, pp. 33–40.

Ogura, T., Okada, K., Inaba, M., y Inoue, H. (2003). Behavior network acquisition in multisensor space for whole-body humanoid. En: *Multisensor Fusion and Integration for Intelligent Systems, MFI2003. Proceedings of IEEE International Conference on*. IEEE, pp. 317–322.

Riemenschneider, H., Krispel, U., Thaller, W., Donoser, M., Havemann, S., Fellner, D., y Bischof, H. (2012). Irregular lattices for complex shape grammar facade parsing. En: *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*. IEEE, pp. 1640–1647.

Russell, S. y Norvig, P. (2009). *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, (3rd ed.). Upper Saddle River, NJ, USA.

Srinivas, M. y Patnaik, L. M. (1994). Genetic algorithms: A survey. *Computer*, **27**(6): 17–26.

Stiefelhagen, R., Fugen, C., Gieselmann, R., Holzapfel, H., Nickel, K., y Waibel, A. (2004). Natural human-robot interaction using speech, head pose and gestures. En: *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*. IEEE, Vol. 3, pp. 2422–2427.

Stiny, G. (1980). Introduction to shape and shape grammars. *Environment and planning B*, **7**(3): 343–351.

Vircikova, M., Fedor, Z., y Sincak, P. (2011). Design of verbal and non-verbal human-robot interactive system. En: *Humanoid Robots (Humanoids), 2011 11th IEEE-RAS International Conference on*. IEEE, pp. 87–92.

Apéndice A. Apéndice

A.1. Gramática prototipo # 2

$S \rightarrow XY$

$X \rightarrow MM$

$Q \rightarrow M$

$M \rightarrow 0M$

$R \rightarrow 1$

$Y \rightarrow ABY$

$A \rightarrow M01$

$B \rightarrow N02$

$P \rightarrow N$

$N \rightarrow 0N$

$L \rightarrow 2$

$W \rightarrow 42$

$Z \rightarrow 41$

$T \rightarrow 0|1|2|3|4$

$M \rightarrow W$

$N \rightarrow Z$

A.2. GC # 1 para el trayecto de Santa Fe

$$\mathbf{GC} = (V, \Sigma, P, S)$$

$$\mathbf{V} = \{VA_c, VE_a, VE_d\}\mathbf{P} = \{PA_c, PE_a, PE_d, PSE_{ev_1}\}\mathbf{S} = \{[START]\}$$

$$\mathbf{\Sigma} = \{(r), (l), (f)\}$$

$$\mathbf{VA_c} = \{[ROT\_RIGHT],\ \ [ROT\_LEFT],\ \ [FORWARD]\}$$

$$\mathbf{VE_a} = \{[START],\ \ [ROT\_360],\ \ [WALL\_INFRONT],\ \ [FOOD],\ \ [NO\_FOOD]\}$$

$$\mathbf{VE_d} = \{[EVENTO]\}$$

$$\mathbf{PA_c} = \{$$

$$[ROT\_RIGHT] \rightarrow\ (r)[EVENTO][ROT\_RIGHT],$$

$$[ROT\_LEFT] \rightarrow\ (l)[EVENTO][ROT\_LEFT],$$

$$[FORWARD] \rightarrow\ (f)[EVENTO][FORWARD]\}$$

$$\mathbf{PE_d} = \{[EVENTO] \rightarrow\ [START]\ \ |\ \ [ROT\_360]$$

$$|\ \ [WALL\_INFRONT]\ \ |\ \ [FOOD]\ \ |\ \ [NO\_FOOD]\}$$

$$\mathbf{PE_a} = \{$$

$$[START] \rightarrow\ [ROT\_RIGHT]\ \ |\ \ [ROT\_LEFT]\ \ |\ \ [FORWARD],$$

$$[ROT\_360] \rightarrow\ [ROT\_RIGHT]\ \ |\ \ [ROT\_LEFT]\ \ |\ \ [FORWARD],$$

$$[WALL\_INFRONT] \rightarrow\ [ROT\_RIGHT]\ \ |\ \ [ROT\_LEFT],$$

$$[FOOD] \rightarrow\ [ROT\_RIGHT]\ \ |\ \ [ROT\_LEFT]\ \ |\ \ [FORWARD],$$

$$[NO\_FOOD] \rightarrow\ [ROT\_RIGHT]\ \ |\ \ [ROT\_LEFT]\ \ |\ \ [FORWARD]\}$$

$$\mathbf{PSE_{ev_1}} = \{[ROT\_RIGHT] \rightarrow\ \epsilon,\ \ \ [ROT\_LEFT] \rightarrow\ \epsilon,\ \ \ [FORWARD] \rightarrow\ \epsilon\ \}$$

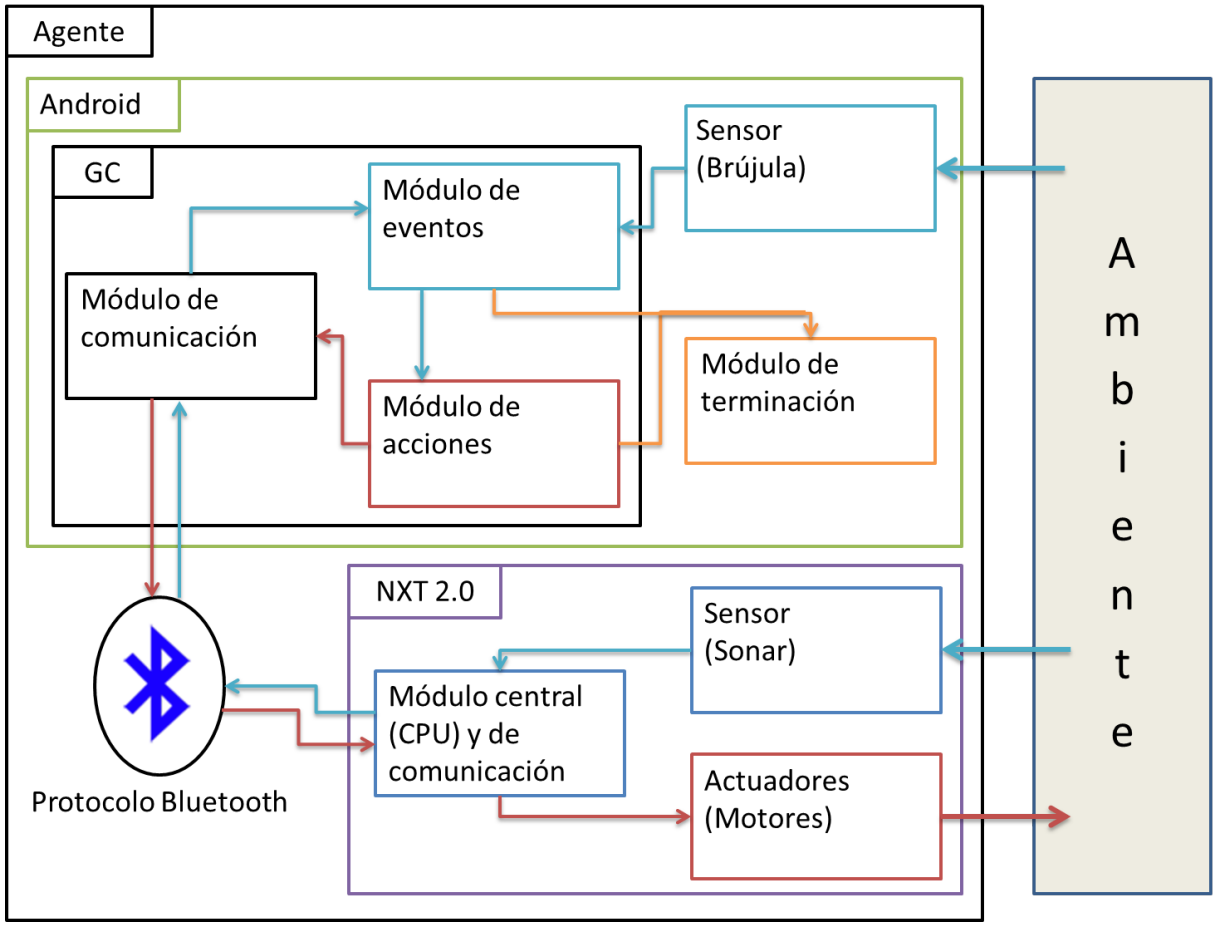


Figura A.1: Modelo que se siguió para la implementación del sistema de gramáticas de comportamientos en el robot NXT 2.0 de Lego. El agente esta compuesto principalmente por dos partes.

A.3. Esquema Android-NXT-GC

La figura A.1 muestra el modelo del agente robótico utilizado para realizar pruebas del sistema. El agente es una combinación de una unidad de Legos NXT 2.0, complementado por un dispositivo móvil Android. El sistema de gramática de comportamientos esta instalado en el dispositivo Android, el cual también cuenta con una brújula. La unidad Lego NXT cuenta con un sonar, motores y con una unidad de procesamiento(bloque NXT). El dispositivo Android y el bloque NXT intercambian información y se coordinan por medio del protocolo de comunicación bluetooth.

Para este robot la parte del sistema correspondiente a las gramáticas fueron implementados en el lenguaje Java para Android, mientras que la parte correspondiente al bloque NXT 2.0 fue implementada en el lenguaje NXC(Not eXactly C).