# INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY


*Uso de aprendizaje computacional para extraer modelos del estudiante en el problema del péndulo invertido*

**Autor**

***Blanca Alicia Vargas Govea***

**Sometido al Programa de Graduados en Informática y Computación en cumplimiento parcial con los requerimientos para obtener el grado de**

**MAESTRA EN CIENCIAS DE LA COMPUTACION.**

**Asesores**
***DR. EDUARDO MORALES MANZANARES***
***DR. RAFAEL MORALES GAMBOA***

Cuernavaca, Morelos a 15 de Agosto del 2002.

**TESIS**

**Uso de aprendizaje computacional para extraer modelos del estudiante en el problema del péndulo invertido.**

Presentada por:

Blanca Alicia Vargas Govea

Aprobada por:

---

Dr. Eduardo Morales Manzanares
Profesor - Investigador ITESM Campus Cuernavaca
Asesor

---

Dr. Rafael Morales Gamboa
Profesor - Investigador
ITESM Campus Cuernavaca - Instituto de Investigaciones Eléctricas
Asesor

---

Dr. Fernando Ramos Quintana
Profesor - Investigador.
Director del Programa de Graduados en Informática ITESM Campus Cuernavaca
Revisor

---

Dr. Enrique David Espinosa Carrillo
Profesor – Investigador
División de Ingeniería y Arquitectura ITESM - Campus Ciudad de México
Revisor

# Agradecimientos

*Dios*

*Papá y Mamá*

*Tulia y Mamá Luz*

*Dr. Eduardo Morales M.*

*Dr. Rafael Morales G.*

*Tere y Don Arturo*

*y a las personas que de alguna manera*
*son parte de este trabajo.*

# Resumen

El aprendizaje computacional estudia y trata de representar simbólicamente la capacidad humana de aprendizaje, así como automatizar éste proceso. Una de las técnicas más usadas es el aprendizaje inductivo, siendo los árboles de decisión y las reglas las representaciones más comunes. El aprendizaje de habilidades humanas presenta dificultades para su representación ya que la persona que aprende no es consciente del proceso llevado a cabo para lograr su objetivo, por lo tanto, no es capaz de proporcionar una descripción adecuada del mismo. Este tipo de aprendizaje involucra una gran cantidad de datos de tipo continuo, lo que dificulta extraer un modelo que describa adecuadamente el desempeño de una persona. Un modelo de esta naturaleza resulta de gran utilidad porque permite inspeccionarlo y comprender la estrategia que la persona sigue. Por esta razón es importante que sea fácil de interpretar.

En esta tesis se extraen modelos de participantes que aprenden a controlar el péndulo invertido. Este es un sistema dinámico en el que un péndulo está unido a la base de un carro que se desplaza sobre un eje horizontal y el objetivo es que se aplique una fuerza al carro para evitar que el péndulo se caiga. Los datos del desempeño (trazas) de los participantes son valores continuos, factor que dificulta el obtener modelos claros. Para solucionar este problema, las trazas se pre-procesaron, transformando los valores cuantitativos en cualitativos a fin de obtener reglas con valores nominales que resultaran más fáciles de entender. Se incorporó conocimiento del dominio: se obtuvieron trazas sobre las tendencias de los valores, la aceleración angular y del carro y la dirección del péndulo y del carro. Posteriormente se indujeron los modelos usando el sistema de aprendizaje de reglas RIPPER. Se obtuvieron los siguientes resultados: el porcentaje de error para el modelo numérico es en promedio 20.03%, al agregar el conocimiento sobre tendencias el porcentaje se incrementa a 24.83%. Para el modelo con aceleración disminuye a 23.07% y al agregar dirección disminuye a 22.12%. No se logra una disminución en el porcentaje de error con respecto al modelo numérico; sin embargo, los resultados sugieren que los modelos se pueden interpretar más fácilmente.

Posteriormente se definieron en Prolog dos predicados a aprender con base en tendencias y aceleración más dirección, respectivamente. Se transformaron los ejemplos originales al formato de esos predicados y se definió el conocimiento del dominio. Esto fue la entrada al sistema de programación lógica inductiva PROGOL, obteniéndose mejores resultados que con RIPPER. El modelo de tendencias obtuvo un porcentaje de error en promedio de 10.5% y al agregar aceleración más dirección disminuye a 5.53%. Además los modelos parecen ser más fáciles de interpretar. El proceso anterior también se hizo con la traza de un experto a fin de comparar sus resultados con los novatos. Los resultados obtenidos sugieren que la incorporación del conocimiento del dominio permite obtener modelos más claros y en el caso de programación lógica inductiva se logra una mayor exactitud. En este trabajo se logran obtener modelos verbalizados a partir de valores numéricos con un porcentaje de error similar o mayor al modelo numérico original para el caso de RIPPER y con un porcentaje bajo de error en el enfoque de programación lógica inductiva usando PROGOL. Se observa que es posible obtener modelos nominales con buen poder predictivo a partir de trazas numéricas.

# Indice general

# Indice de tablas

# Indice de Figuras

Capítulo 1

# Introducción

Cuando una persona aprende una materia como biología, requiere aprender conceptos que posteriormente aplica para resolver problemas. La persona es capaz de describir lo que sabe y puede transmitir su conocimiento, sea correcto o no, de forma explícita. Al resolver un problema, puede proporcionar un conjunto de pasos o reglas que describan la forma en que llegó a una solución y si ésta no es correcta, es posible revisar el proceso y encontrar el error. La descripción que la persona hizo es el modelo de su conocimiento y puede ser inspeccionado y programado en un sistema.

Si lo que se trata de describir es una habilidad, como manejar una bicicleta o batear una pelota, la dificultad se incrementa ya que este tipo de aprendizaje se realiza a nivel inconsciente y es muy difícil que una persona pueda describir paso a paso lo que hizo. Modelar este tipo de habilidades representa un reto, puesto que el objetivo es extraer descripciones explícitas sobre algo que por naturaleza no lo es.

Una vez obtenido el modelo, éste puede tener diversas funciones: puede utilizarse como medio de comunicación, como herramienta de introspección, o de predicción y control, entre las más importantes. En el área de tutores inteligentes, por ejemplo, el modelo del estudiante es un componente fundamental para aplicar la estrategia pedagógica adecuada a cada individuo. Modelar el comportamiento de personas que están aprendiendo presenta además otros problemas, entre ellos, la inconsistencia en sus respuestas o acciones y si lo que la persona está aprendiendo es una tarea de control entonces la dificultad se incrementa, pues generalmente este tipo de tareas involucran una gran cantidad de datos de tipo contínuo y ruido.

Con el fin de comprender y modelar las habilidades humanas, se han estudiado sistemas simples de control dinámico, uno de ellos es el péndulo invertido. En este sistema, un péndulo rígido está unido en la parte superior un carro que se desplaza sobre un eje horizontal y el objetivo es que se aplique una fuerza al carro para evitar que el poste se caiga. Este sistema de control ha sido objeto de estudio de numerosas investigaciones.

En este trabajo se presenta una metodología para extraer modelos de las habilidades de control de un péndulo invertido. El objetivo es obtener modelos fáciles de interpretar que sean útiles para comprender la estrategia que la persona sigue al desempeñar su tarea. Para lograrlo, se incorpora conocimiento del dominio, con lo que se espera incrementar el poder descriptivo del modelo sin perder exactitud. El proceso involucra pre-procesamiento de datos, aplicación de la técnica conocida como clonación de comportamiento y el uso de dos sistemas de aprendizaje inductivo: RIPPER y PROGOL, lo que nos da dos enfoques diferentes en la construcción de los modelos.

## 1.1 Objetivo general

El objetivo de esta tesis es construir modelos del estudiante descriptivos y fáciles de interpretar que sean útiles para inspeccionar la estrategia seguida por el estudiante en el desempeño de una tarea de control. Esto se logra mediante la transformación de datos e incorporación de conocimiento del dominio. Esto último es de gran utilidad porque permite obtener modelos más descriptivos y con menor error de predicción.

## 1.2 Objetivos específicos

El desarrollo de este trabajo tiene por objetivos específicos los siguientes:

- Aplicar diversas técnicas de aprendizaje computacional, como la clonación de comportamiento y el aprendizaje inductivo de reglas.
- Obtener modelos del estudiante con RIPPER y PROGOL y comparar los resultados.

## 1.3 Hipótesis

Demostrar que la incorporación de conocimiento del dominio permite construir modelos más descriptivos y fáciles de interpretar.

## 1.4 Estructura de la tesis

La tesis está estructurada de la siguiente forma:

**Capítulo 1.** Presenta la introducción, la hipótesis, el objetivo general, los objetivos específicos y la estructura de la tesis.

**Capítulo 2.** Muestra un panorama general de aprendizaje computacional. Describe RIPPER, un algoritmo de aprendizaje de reglas, y el sistema de programación lógica inductiva PROGOL, herramientas utilizadas en el desarrollo de este trabajo.

**Capítulo 3.** Describe la importancia del modelo del estudiante como componente de un sistema tutor inteligente.

**Capítulo 4**. Describe el problema del péndulo invertido. Explica la técnica conocida como clonación de comportamiento (*behavioural cloning*) para la extracción de modelos de actividades humanas y trabajos relacionados.

**Capítulo 5.** Explica la metodología utilizada para construir los modelos, el planteamiento del conocimiento del dominio para inducción y los resultados obtenidos con los sistemas de aprendizaje RIPPER y PROGOL. Muestra los resultados de los experimentos.

**Capítulo 6.** Presenta las conclusiones y trabajo futuro.

Capítulo 2

# Aprendizaje

Aprender es una actividad realizada por el hombre desde antes de nacer y continúa desarrollándola durante toda su vida. Utilizar el término aprendizaje involucra muchos aspectos: la adquisición de nuevo conocimiento, su clasificación, el desarrollo de habilidades a través de la práctica, el descubrir nuevos hechos con base en conocimiento previo, así como la búsqueda de facilitar el proceso de aprendizaje de modo que sea más rápido y eficiente. En el ser humano este proceso se da de forma natural y siempre es posible mejorarlo.

Desde que el uso de las computadoras se incorporó a las actividades humanas se han dedicado grandes esfuerzos para lograr que las habilidades características de una persona tales como visión y lenguaje, entre muchas más, puedan representarse e implantarse en computadoras. La capacidad de aprendizaje es una de las que ha constituido uno de los mayores retos en inteligencia artificial.

El estudio de los procesos de aprendizaje y la creación de sus correspondientes modelos computacionales es de lo que trata el aprendizaje computacional.

## 2.1 Tipos de aprendizaje

De acuerdo a Carbonell, Michalski y Mitchell [Carbonell,1983], existen dos formas básicas de aprendizaje:

1) Adquisición de conocimiento y

2) Refinamiento de habilidades.

La adquisición de conocimiento consiste en el aprendizaje de nuevos conceptos sobre un tema específico a través de descripciones, imágenes, representaciones, ejemplos y su aplicación, conocido también como conocimiento declarativo [Michie, 1994]. Se dice que una persona ha aprendido más si su conocimiento permite explicar y resolver una amplia gama de situaciones, es preciso y puede predecir lo que ocurrirá con base en el conocimiento adquirido. La adquisición de conocimiento declarativo se define entonces como el aprendizaje de nueva información simbólica unida a la habilidad de aplicar esa información de manera efectiva.

El segundo tipo de aprendizaje consiste en el mejoramiento gradual de habilidades cognoscitivas y motoras a través de la práctica, como por ejemplo, manejar una bicicleta, tocar un instrumento o controlar un dispositivo. En este caso, la adquisición de

conocimiento es el primer paso para desarrollar las habilidades, el resto del proceso de aprendizaje consiste en el mejorar las habilidades adquiridas practicando y corrigiendo errores. El aprendizaje del control de un péndulo invertido, que es el modelo de estudio de este trabajo, corresponde a este tipo.

A simple vista no se percibe la diferencia entre los dos enfoques ya que puede pensarse que en ambos tipos de aprendizaje la destreza mejora con la práctica; si una persona estudia matemáticas y resuelve ejercicios constantemente su habilidad mejora, al igual que aprende a tocar magistralmente el piano si practica diario, sin embargo, la diferencia es la forma en que ocurre el proceso: la adquisición de conocimiento es un proceso consciente cuyo resultado es la creación de nuevas estructuras simbólicas de conocimiento y modelos mentales mientras que el refinamiento de habilidades se lleva a cabo a un nivel inconsciente como resultado de la práctica constante y la reacción ante eventos inesperados. En la práctica, ambos tipos de aprendizaje se mezclan.

## 2.2 Estrategias de aprendizaje automático

El aprendizaje computacional busca el desarrollo de sistemas que sean capaces de inferir conocimiento nuevo a partir del adquirido previamente, que no tenga que ser programado directamente; es decir, automatizar el proceso de aprendizaje.

Las estrategias de aprendizaje pueden clasificarse de acuerdo a la cantidad de inferencia que el sistema tiene que desarrollar [Carbonell, et al., 1983]. En orden ascendente, se tiene:

1) Aprendizaje de memoria. El nuevo conocimiento se implanta directamente, sin ninguna inferencia. Como ejemplo se tiene el programar directamente las reglas dadas por un experto humano.

2) Aprendizaje supervisado (Inducción). Al sistema se le proporciona un conjunto de ejemplos de entrenamiento consistente en entradas y salidas y tiene que descubrir la relación entre ellos. Ejemplos: el aprendizaje de un conjunto de reglas o el aprendizaje que tiene lugar en un árbol de decisión.

3) Aprendizaje no supervisado (Descubrimiento). Se le proporciona al sistema un conjunto de ejemplos de entrenamiento que consiste sólo de entradas y tiene que descubrir las salidas. Ejemplo: Red de Kohonen.

4) Aprendizaje por Refuerzo. Retroalimentación de una recompensa ya sea positiva o negativa dada al final de una secuencia de pasos.

Algunas de las técnicas de con base en la representación del conocimiento aprendido:

**Arboles de decisión.** En esta representación los nodos corresponden a atributos del objeto y las ramas a los distintos valores que pueden tomar los atributos. Las hojas son las clases en las que los objetos se clasifican.

**Reglas de producción.** Es un par condición - acción {C=>A}, donde C es el conjunto de condiciones y A es la secuencia de acciones. Si todas las condiciones en una regla de producción se satisfacen, entonces se ejecuta la secuencia de acciones.

**Lógica formal.** Son representaciones de propósito general que se han usado para elaborar descripciones de objetos individuales (entrada de un sistema de aprendizaje) y para formular las descripciones conceptuales resultantes (salida). Toman la forma de expresiones lógicas formales cuyos componentes son proposiciones, predicados arbitrarios, variables de valores finitos o expresiones que limitan el rango de las variables o expresiones.

**Grafos y redes**. En muchos dominios los grafos y las redes proporcionan una representación más conveniente y eficiente que las expresiones lógicas.

**Frames y esquemas**. Proporcionan unidades mayores de representación que las expresiones lógicas o reglas de producción. Pueden verse como conjuntos de atributos o "slots", cada uno con un fin específico en la representación.

## 2.3 Aprendizaje inductivo

El aprendizaje inductivo consiste obtener una teoría a partir de un conjunto de ejemplos. Dado un conjunto de ejemplos, el objetivo es determinar si un ejemplo nuevo es una instancia de una clase o no. Si es una instancia, se dice que el ejemplo es positivo, si no, el ejemplo es negativo. Para un conjunto de entrenamiento de ejemplos positivos y negativos, se construye una descripción que clasifique con exactitud éstos y los futuros ejemplos.

Cada ejemplo puede considerarse un vector de características V, que contiene los atributos relevantes para clasificar los ejemplos y la clase a la que pertenece. Por ejemplo:

V = [pelo: no, plumas: si, patas: 2, leche: no, doméstico: si, clase: pollo]

Este vector muestra los atributos para identificar a un animal, éstos pueden tomar diferentes valores y la clase correspondiente. En este caso, el ejemplo describe a un pollo. El número de atributos es fijo y cada atributo tiene en general un número finito de posibles valores.

Los árboles de decisión y las reglas son los métodos más utilizados.

## 2.4 Arboles de decisión

Los árboles de decisión son una de las representaciones más simples y poderosas existentes. Es por esta razón que es una de las más utilizadas en comparación con otras, como redes semánticas, lógica de primer orden, etc.

Los primeros sistemas que construían árboles se desarrollaron a principios de la década de los 60's; entre ellos: **EPAM** (Elementary Perceiver and Memorizer) [Feigenbaum, 1961] era un modelo de simulación cognoscitiva del aprendizaje humano y **CLS** (Concept Learning System) [Hunt, 1966], sistema desarrollado por psicólogos como modelo del proceso cognitivo de formación de conceptos .

Los algoritmos para inducción de árboles de decisión se basan en un enfoque de "divide y vencerás", buscando en un vector de características, un atributo que separe las clases lo mejor posible y procesar las particiones resultantes de forma recursiva.

### 2.4.1 Conceptos generales

Dado un conjunto de ejemplos estructurados en la forma atributo-valor, un nodo del árbol de decisión corresponde a un atributo, las ramas corresponden a valores posibles que puede tomar el atributo y las hojas representan las clases. De esta manera, al tomar un ejemplo, la clasificación se realiza probando cada valor de atributo, tomando la clase de la hoja correspondiente.

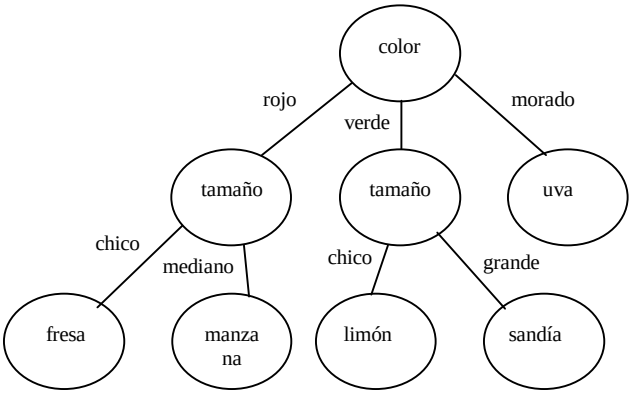



Figura 2.1: Arbol de decisión

Así, un árbol de decisión puede ser interpretado como un conjunto de reglas de clasificación para un conjunto de ejemplos.

La salida de un árbol puede expresarse por reglas:

Si (color = rojo)
    Si (tamaño = chico) entonces (clase=fresa).
    Si (tamaño = mediano) entonces (clase = manzana).
Si (color = verde)
    Si (tamaño = chico) entonces (clase = limón).
    Si (tamaño = grande) entonces (clase = sandía).
Si (color = morado) entonces (clase = uva).

Para cada par atributo-valor se construye una expresión siguiendo cada rama del árbol. Las hojas representan la clase en la que se clasifica el ejemplo.

Los árboles de decisión no permiten representar cualquier conjunto, se limitan a tratar con un solo objeto; sin embargo, cualquier función booleana puede representarse como un árbol de decisión.

El algoritmo ID3 [Quinlan, 1979] es importante por ser el algoritmo básico para diversas implementaciones que se han utilizado para inducir modelos de habilidades humanas. A continuación se describe brevemente el funcionamiento de ID3.

## 2.4.2 ID3

ID3 [Quinlan, 1979] incorpora una heurística basada en la teoría de la información que permite construir árboles más pequeños y eficientes, seleccionando el atributo que es más útil para clasificar los ejemplos.

Pero ¿cómo selecciona el mejor atributo?, esto lo hace con base en una medida conocida como ganancia de información que mide qué tan bien un atributo separa a los ejemplos de entrenamiento de acuerdo a su clasificación.

### Ganancia de información y entropía

La ganancia de información está definida con base en el concepto de entropía, que es una medida usada comúnmente en la teoría de la información. La entropía representa la pureza de un conjunto arbitrario de ejemplos. Si tenemos un conjunto E, conteniendo ejemplos positivos y negativos, la entropía se representa como:

Entropía(E) = $-p_p \log_2 p_p - p_n \log_2 p_n$

donde $p_p$ es la proporción de ejemplos positivos en E y $p_n$ es la proporción de ejemplos negativos, para este caso, se considera que sólo se tienen dos clases. Generalizando, si existen más de dos clases para un conjunto S de ejemplos, entonces se tiene:

$$\text{Entropía(S)} = \sum_{i=1}^{c} - p_i \log_2 p_i$$

donde *c* es el número total de clases y $p_i$ es la proporción de S que corresponde a la clase *i*.

Se observa que la entropía es 0 si todos los ejemplos son de la misma clase.

La ganancia de información es la reducción esperada en la entropía causada por la partición de ejemplos al seleccionar un determinado atributo. De esta manera, la ganancia de un atributo A, relativo a un conjunto de ejemplos S se define como:

$$Ganancia(S, A) = Entropía(S) - \sum_{v \in Valores(A)} \frac{|S_v|}{|S|} Entropía(S_v)$$

donde *Valores(A)* es el conjunto de todos los posibles valores para el atributo *A* y $S_v$ es el subconjunto de *S* para el cual el atributo *A* tiene el valor *v*.

Resumiendo:
ID3 para cada atributo que no ha sido usado:

- Calcula la ganancia de información
- Selecciona el atributo que proporcione la mayor ganancia.
- Genera los nodos hijos, uno para cada posible valor del atributo y se asignan los subconjuntos de los ejemplos a su correspondiente nodo hijo.

Se repite para cada nodo hijo hasta que se cumpla cualquiera de las siguientes condiciones:

1) Ya no hay ejemplos.
2) Los ejemplos de entrenamiento asociados con este nodo pertenecen a la misma clase.
3) Ya no hay atributos, aunque existen ejemplos de diferentes clases.

El algoritmo usa una búsqueda avara ( *greed y*); esto es, selecciona el mejor atributo dado un conjunto de ejemplos y nunca regresa a reconsiderar selecciones anteriores.

Para manejar conjuntos grandes de ejemplos puede utilizarse una técnica conocida como *windowing*, que consiste en seleccionar una muestra del conjunto de ejemplos de entrenamiento y construir el árbol. Posteriormente se prueba el resto de los ejemplos y aquellos que no se clasifiquen correctamente se van agregando de acuerdo al número máximo permitido en cada iteración y el árbol vuelve a construirse hasta que cubra todos los ejemplos de entrenamiento.

### 2.4.2.1 Manejo de ruido

Los valores de los atributos en el conjunto de ejemplos pueden presentar ruido, esto ocurre cuando:

- Dos ejemplos tienen el mismo par atributo-valor, pero diferente clase.
- Puede haber valores de atributos incorrectos debido a errores en la adquisición de datos o en la fase de preprocesamiento.
- La clase puede estar equivocada.
- Se observan valores faltantes.
- Existen atributos irrelevantes al proceso de toma de decisión.

Los atributos irrelevantes pueden originar sobreajuste (*overfitting)* ya que estos datos son tomados como significativos por lo que el resultado no será el más adecuado.

Para manejar el sobreajuste se han usado dos clases de enfoque:

- Pruning: Podar el árbol antes de que clasifique todos los ejemplos de entrenamiento.
- Postpruning: Permitir que el árbol crezca y podar el árbol después.

En la práctica el post-pruning ha obtenido mejores resultados debido a la dificultad para estimar el momento preciso para dejar de crecer el árbol.

### 2.4.2.2 Extensiones

A partir de ID3 se han hecho extensiones que incorporan mejoras, a continuación se mencionan algunas de ellas:

**C4.5** [Quinlan,1993] . Algunas características adicionales que agrega son:

- Manejo de valores continuos.
- Post-pruning.
- Manejo de valores faltantes.

**CART** [Breiman, 1984]. Basado en árboles de regresión (los valores de atributo de cada ejemplo son valores numéricos reales, en lugar de valores nominales). Utiliza una metodología conocida como particionamiento binario recursivo; la selección del atributo que separa mejor los ejemplos se efectúa con la medida de ganancia Gini, aunque puede utilizarse otra. Es ampliamente utilizado a nivel comercial.

**RETIS** (Karalic, 1992) y **M5** [Quinlan, 1992]. Aprenden modelos de regresión lineal en las hojas. CART se limita a tener valores promedio en las hojas, lo que origina una pobre extrapolación de los datos originales.

El uso de árboles de decisión se ha extendido a diversas áreas del conocimiento, entre ellas se encuentran: astronomía, biología molecular, visión, física, procesamiento de textos y medicina entre otras.

## 2.5 Aprendizaje inductivo de reglas

Veamos ahora un enfoque alternativo, los algoritmos de inducción de reglas. Esta representación tiene muchas ventajas, una de las más importantes es su facilidad de interpretación, lo que ha hecho que este tipo de aprendizaje sea uno de los más usados.

A diferencia de los algoritmos de árboles de decisión que buscan el atributo que separe las clases lo mejor posible (*splitting*), el aprendizaje de reglas se basa en una estrategia de *covering* que consiste en ir agregando condiciones a la regla que se está construyendo tratando de que cubra todos los ejemplos de una clase y excluyendo a los que no pertenecen a la misma. El algoritmo de *covering* selecciona un par atributo-valor para maximizar la probabilidad de la clasificación deseada y por lo tanto, maximizar la exactitud de la regla construida.

El algoritmo básico de *covering* es el siguiente:

| |
|---|
| Sea E el conjunto de ejemplos positivos<br>Aprende una regla R que cubra el mayor número de ejemplos positivos sin cubrir ningún negativo.<br>Agrega R al conjunto de reglas aprendidas.<br>Elimina los ejemplos positivos cubiertos de E<br>Repite hasta que E esté vacío. |

Tabla 2.1 Algoritmo básico de covering

Para aprender una regla existen dos posibles estrategias:

**De general a específico** (*top-down*): Inicia con una regla general a la que se le van agregando condiciones que cubran los ejemplos positivos y elimine los negativos. Termina cuando solo se hayan cubierto los ejemplos positivos.
**De específico a general** (*bottom-up*): Inicia con la regla más específica, es decir, la descripción de un ejemplo y se van eliminando las condiciones a fin de cubrir más ejemplos positivos. Termina cuando posteriores generalizaciones cubren negativos.
Para seleccionar la regla que se agregará al conjunto de reglas aprendidas pueden utilizarse diversos criterios:

**Frecuencia relativa:** es la relación entre el número de ejemplos positivos ($n^+$) y los ejemplos totales cubiertos por la regla ($t$).

$$p(R) = \frac{n^+}{t}$$

**Estimador Laplaciano:** Supone una distribución uniforme en las clases

$$p(R) = \frac{n^+ + 1}{t + k}$$

donde $k$ es el número de clases.

**Estimador m:** Toma en cuenta probabilidades *apriori* ($P_a(C)$) de las clases en lugar de una distribución uniforme, es independiente del número de clases k y m es dependiente del dominio, a mayor ruido, m es más grande.

$$p(R) = \frac{n^+ + m \cdot P_a(C)}{t + k}$$

La regla que maximice la medida de probabilidad es la que se elige; sin embargo, esto no garantiza que sea la mejor. A través de un proceso de simplificación, las reglas pueden mejorarse, y al igual que los árboles pueden ser podadas (*pruning*). El proceso consiste en eliminar la última condición a la regla construída y verificar si ésta es mejor a la original, la verificación se hace con base en una medida de probabilidad como las anteriores ($p(R)$). En la siguiente sección se describe brevemente un algoritmo que utiliza simplificación.

### 2.5.1 REP

Los sistemas más usados de árboles de decisión utilizan una estrategia para manejar el sobreajuste ocasionado por datos con ruido, que consiste en construir el árbol incluyendo los atributos irrelevantes y después podar o simplificar el árbol resultante. Uno de los métodos más efectivos para podar árboles es REP (*Reduced Error Pruning*) que puede adaptarse fácilmente a sistemas de aprendizaje de reglas [Pagallo & Haussler, 1990; Brunk & Pazzani, 1991].

La estrategia de REP consiste en dividir los datos de entrenamiento en un conjunto de crecimiento y un conjunto de poda. Se forma un conjunto de reglas iniciales, que incluyen atributos irrelevantes y posteriormente este conjunto se va simplificando hasta que en lugar de disminuir el error en el conjunto de poda, éste aumenta.

REP es ineficiente para conjuntos grandes por lo que posteriormente se desarrolló IREP, algoritmo que mejora esta deficiencia.

### 2.5.2 IREP

IREP *(Incremental Error Pruning*) construye un conjunto de reglas en un estilo *greedy*, es decir, una regla a la vez. Después de que una regla ha sido encontrada, todos los ejemplos

cubiertos por la regla son eliminados. Este proceso se repite hasta que no hay ejemplos positivos o si el conjunto de reglas es largo [Fürnkranz & Widmer, 1994 ].

El algoritmo es el siguiente:

```
 IREP(Pos,Neg)
inicia
        conjuntoReglas=0
        Mientras Pos ≠ 0 y  conjuntoReglas no es largo
                /* genera y poda una nueva regla */
                divide (Pos,Neg) en (GrowPos,GrowNeg) y  (prunePos,pruneNeg)

                  regla = GrowRule(GrowPos,GrowNeg)
                  regla = pruneRule(regla,prunePos,pruneNeg)

                agrega regla a conjuntoReglas
                elimina ejemplos  cubiertos por la regla

        fin Mientras
        regresa conjuntoReglas
fin
```

Tabla 2.2 Algoritmo IREP

Para construir una regla, IREP usa la siguiente estrategia:

1) Los ejemplos no cubiertos se particionan aleatoriamente en dos subconjuntos: uno de crecimiento y uno de poda (*pruning*).

2) Se genera una nueva regla con (*GrowRule*). Inicia con una conjunción vacía de condiciones. *GrowRule* agrega repetidamente la condición hasta que la regla cubre los ejemplos positivos del conjunto de datos de crecimiento.

3) Después de hacer crecer una regla, ésta es inmediatamente podada (*pruned*), para ello se eliminan las condiciones que maximizan la siguiente función:

$$p(regla, prunePos, pruneNeg) = \frac{p-n}{p+n}$$

donde p es el número de ejemplos positivos y n es el número de ejemplos negativos en el conjunto de poda cubiertos por la nueva regla.

La regla se agrega al conjunto de reglas y los ejemplos cubiertos por ella son eliminados.

Después de agregar cada regla, se calcula la longitud de descripción total del conjunto de reglas y de los ejemplos. IREP se detiene cuando esta longitud es mayor que la descripción más pequeña obtenida hasta ahora, o cuando ya no hay ejemplos positivos.

Posteriormente, se agrega una mejora a este algoritmo, que toma el nombre de RIPPER (*Repeated Incremental Pruning to Produce Error Reduction*) [Cohen, W., 1995].

## 2.5.3 RIPPER

Para mejorar el desempeño general de IREP se agregó una fase de optimización, que se describe a continuación:

### Optimización de reglas.

Del conjunto de reglas construido, se va tomando una a una tratando de reemplazar la regla en turno por una que la mejore. Para construir la regla de reemplazo se sigue una estrategia similar a la de la primer fase: se crea una regla y se simplifica pero ahora con el objetivo de reducir el error del conjunto total de reglas en el conjunto de poda. Finalmente se decide si la regla de reemplazo o la original será la que se agregue al conjunto de reglas; para seleccionarla se utiliza la heurística MDL o principio de longitud de descripción mínima. Este principio establece que la mejor teoría derivable de los datos de entrenamiento será aquella que requiere el mínimo número de bits.

Esta fase de optimización permite obtener modelos más compactos y de mayor exactitud. RIPPER es entonces: IREP + Fase de optimización.

### Ventajas de RIPPER

RIPPER presenta diversas ventajas sobre otros sistemas de aprendizaje, entre ellas se pueden mencionar:

1) La hipótesis es expresada como un conjunto compacto de reglas *if-then*, fáciles de entender.

2) Es más rápido que otros algoritmos de aprendizaje de reglas, por lo que es muy útil para conjuntos de datos grandes.

3) Los atributos pueden ser nominales, contínuos o de conjunto (*set-valued*). El valor de un atributo de este tipo es un conjunto de átomos.

4) Manejo eficiente de ruido.

Debido a las características mencionadas, RIPPER es uno de los sistemas utilizados en esta tesis para la obtención de los modelos.

## 2.6 Programación Lógica Inductiva (ILP)

Hasta el momento, los algoritmos vistos se basan en el aprendizaje utilizando ejemplos formados por un conjunto fijo de atributos, sin embargo en ocasiones esto no es suficientemente expresivo para representar situaciones del mundo real, como por ejemplo, áreas donde se requiere de conocimiento relacional, razonamiento temporal, razonamiento cualitativo y espacial, etc.

La programación lógica inductiva (ILP) permite este tipo de representaciones, para ello, permite introducir ejemplos y además, conocimiento del dominio, ambos expresados en forma de programas lógicos (cláusulas de Horn). El resultado de un sistema ILP es un conjunto de reglas que cubre todos los ejemplos positivos y ningún negativo. Por esta razón, ILP puede considerarse la intersección entre el aprendizaje computacional y la programación lógica.

ILP utilizando lógica computacional como formalismo de representación de hipótesis y observaciones trata de superar las limitaciones de las técnicas clásicas de aprendizaje computacional.

### 2.6.1 El problema general de ILP

El problema en ILP se define de la siguiente manera:

Dados:
Un conjunto de ejemplos positivos $E^+$ .
Un conjunto de ejemplos negativos $E^-$ .
Un conocimiento del dominio *B*.

Donde el $E^+$ y $E^-$ son hechos aterrizados y B es un conjunto de predicados, el objetivo es encontrar una hipótesis tal que:

$$B \wedge H \models E^+ \quad y \quad B \wedge H \not\models E^-$$

Esto significa que la hipótesis debe cubrir todos los ejemplos positivos (completez) y no cubrir ningún negativo (consistencia).

### 2.6.2 Métodos de búsqueda

La búsqueda de hipótesis puede realizarse de tres formas:

1) General a específica (*top-Down*)
Este tipo de búsqueda parte de predicados con todos los argumentos con variables y empieza a especializar hasta excluir los ejemplos negativos. Inicia con una cláusula general

corta y la especializa aplicando sustituciones de variables por términos y/o agregando literales.

2) Específica a general (*bottom-up*)
Inicia con cláusulas largas, específicas, a las que generaliza aplicando sustituciones inversas (constantes en variables)) y/o eliminando literales, eliminando condiciones, cambiando conjunciones en disjunciones.

3) Bidireccional. Utiliza generalización y especialización (sección 2.6.3).

### 2.6.3 Reglas de inferencia inductiva y operadores

La inducción puede ser vista como el inverso de la deducción. Dada la fórmula $B \wedge H \models E^+$ derivar $E^+$ de $B \wedge H$ es deducción, y derivar H de B y $E^+$ es inducción. Por lo tanto, las reglas de inferencia inductivas pueden obtenerse invirtiendo las deductivas [Muggleton, 1994]. A continuación se explican más detalladamente estas reglas y los operadores de inferencia inductiva.

### Subsumsión

Dadas dos cláusulas C y D, C subsume a D ($C \preceq D$), si y sólo si existe una sustitución $\theta$ tal que $C\theta \subseteq D$. Por ejemplo, si tenemos las siguientes cláusulas

C = papa(X,Y) ← progenitor(X,Y),
hombre(X).
D = papa(luis,maria) ← progenitor(luis,juan),
progenitor(luis,maria),
hombre(luis),
mujer(maria).

si tomamos la substitución $\theta$ = {X/luis, Y/maria}

$C\theta$ = papa(luis,maria) ← progenitor(luis,maria),hombre(luis)

Escribiendo las cláusulas en notación de conjuntos, tenemos

$C\theta$ = **{**papa(luis,maria), $\overline{\text{progenitor(luis,maria)}}$, $\overline{\text{hombre(luis)}}$**}**

D = **{** papa(luis,maria), progenitor(luis,juan), progenitor(luis,maria), hombre(luis),
mujer(maria) **}**

podemos concluir que C es un subconjunto propio de D y C θ subsume a D bajo la substitución θ = {X/luis, Y/maria}.

La subsumsión es importante porque proporciona una forma de ordenar las cláusulas por su generalidad. Si C θ subsume a D, C es una generalización de D y D es una especialización de C. Es una forma de definir el espacio de búsqueda o *lattice*, el cual es un conjunto parcialmente ordenado en el que cada par de elementos tiene un límite inferior más grande (*glb: greatest lower bound*) y un límite superior más bajo (*lub: least upper bound*).

Ejemplo:

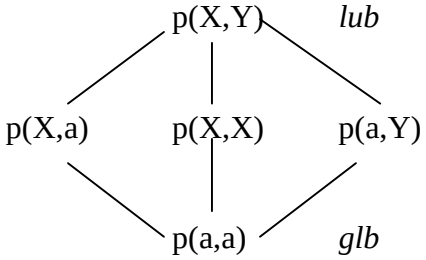



Figura 2.2 *Lattice.* p(X,Y) subsume a p(X,a) con la substitución θ = {Y/a}. p(X,Y) subsume a p(X,X) con la substitución θ = {Y/X}. p(X,Y) subsume a p(a,Y) con la substitución θ = {X/a}. p(X,Y) subsume a p(a,a) con la substitución θ = {X/a,Y/a}.

## Generalización menos general ( lgg )

Este operador es el dual de la unificación. Con el ordenamiento obtenido usando subsumsión, se puede definir la generalización menos general de dos cláusulas C y D. Veamos el siguiente ejemplo:

C = hija(maria,ana) ← mujer(maria), progenitor(ana,maria).
D = hija(rosa,tomas) ← mujer(rosa), progenitor(tomas,rosa).
*lgg*(C,D) = hija(X,Y) ← mujer(X), progenitor(Y,X)

## Generalización menos general relativa a una teoría ( rlgg )

Se puede definir como la generalización menos general de dos cláusulas *lgg*(C,D) relativa al conocimiento del dominio B. De esta manera, si K es la conjunción de todas las cláusulas en B, podemos calcular el *rlgg*(C,D) como vemos a continuación:

$$rlgg(C,D) = lgg((C \leftarrow K), (D \leftarrow K))$$

Consideremos los siguientes ejemplos $e_1$ y $e_2$:

hija(maria,ana).
hija(rosa,tomas).

y el conocimiento del dominio:

progenitor(ana,maria).
progenitor(tomas,rosa).
progenitor(tomas,juan).
mujer(ana).
mujer(maria).
mujer(rosa).

K representa la conjunción:
progenitor(ana,maria) $\wedge$ progenitor(ana,tomas) $\wedge$ progenitor(tomas,rosa) $\wedge$ progenitor(tomas, juan) $\wedge$ mujer(ana) $\wedge$ mujer(maria) $\wedge$ mujer(rosa).

entonces, el rlgg($e_1$,$e_2$) = *lgg*(hija(maria,ana) $\leftarrow$ K),hija(rosa,tomas) $\leftarrow$ K))
= hija(X,Y) $\leftarrow$ mujer(X),progenitor(Y,X).

Resumiendo, el *lgg* relativo a una teoría se obtiene agregando a una cláusula todas las literales implicadas por el conocimiento del dominio dado y obtener posteriormente su *lgg*.

## Resolución inversa

Si tenemos dos cláusulas disjuntas $C_1$ y $C_2$, mediante resolución se puede deducir una nueva cláusula de la siguente manera:
Sean $l_1$ y $l_2$ literales de $C_1$ y $C_2$. Sea $\theta$ el unificador más general (*mgu*) de $l_1$ y $l_2$ tal que : $\neg l_1 \theta = l_2 \theta_2$, *y* $\theta = \theta_1\theta_2$. Entonces el resolvente $C$ de $C_1$ y $C_2$ *es:*

$$C=(C_1-\{L_1\})\,\theta_1 \cup (C_2-\{L_2\})\theta_2$$

De acuerdo a [Muggleton,1994], la resolución inversa se basa en los operadores V y W, que se describen a continuación.

## 1) Operador V

Existen dos casos: absorción e identificación.

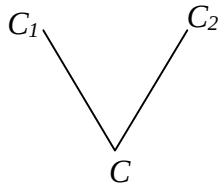



Figura 2.3 Operador V

**Primer caso: Absorción**: Dado un conjunto de cláusulas, el cuerpo de una está contenida completamente en el cuerpo de las otras. Ejemplo:

$C$ = p ← A, B.
$C_1$ = q ← A.

el cuerpo de $C_1$ es absorbido en el cuerpo de $C$, entonces se obtiene

$C_2$ = p ← q, B.

**Segundo caso: Identificación**: Identifica parte del cuerpo de $C_2$ en la $C$. Trata de construir una nueva cláusula $C_1$ usando como cabeza la literal de $C_2$ que no está en $C$ y como cuerpo, la parte de la $C$ que no está en $C_2$. Ejemplo:

$C$ = p ← A, B.
$C_2$ = p ← A, q.

q es la literal de $C_2$ que no está en $C$, por lo que se usa como cabeza de $C_1$ y B es la literal de $C$ que no está en $C_2$, por lo que $C_1$ queda de la siguiente forma:

$C_1$ = q ← B.

## 2) Operador W

El operador W combina dos operadores V.

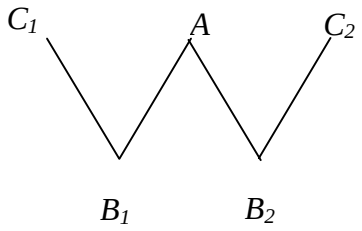



Figura 2.4 Operador W

Si $C_1$ y $C_2$ se resuelven sobre una misma literal $l$, el operador W encuentra $A$, $C_1$ y $C_2$ dadas $B_1$ y $B_2$. Cuando $l$ no está ni en $B_1$ ni en $B_2$, el operador W tiene que inventar un nuevo predicado.

**Intra-Construcción**: Toma un grupo de reglas con la misma cabeza. Las literales comunes se comprimen en una cláusula y construye una nueva cláusula que tiene como cuerpo las literales diferentes. Ejemplo:

B1 = p ← A,B.
B2 = p ← A,C.

A = p ← A,q.
C1 = q ← B.
C2 = q ← C.

**Inter-Construcción:** Toma un grupo de reglas con diferente cabeza y construye una nueva cláusula con las literales comunes. Ejemplo:

B1 = p ← A, B.
B2 = q ← A, C.

A = r ← A.
C1 = p ← r, B.
C2 = q ← r, C.

## Implicación inversa *(Inversed Entailment)*

Dado un conjunto de ejemplos (*E*), conocimiento del dominio (*B*) e hipótesis (*H*). Tenemos entonces la relación

$$B \wedge H \models E$$

Consideremos que *H* y *E* son cláusulas de Horn simples, entonces, la expresión anterior se puede reordenar así

$$B \wedge \neg E \models \neg H$$

Como *H* y *E* son cláusulas simples, $\neg H$ y $\neg E$ serán programas lógicos formados sólo de cláusulas unitarias skolemnizadas es decir, sin cuantificadores y aterrizadas. Sea $\neg \perp$ un conjunto potencialmente infinito de todas las literales aterrizadas que son la consecuencia de *B* y $\neg E$, entonces, de acuerdo a [Muggleton, 1995]:

$$B \wedge \neg E \models \neg \perp$$

Como $\neg H$ debe ser verdadero en cada modelo de $B \wedge \neg E$, debe contener un subconjunto de las literales aterrizadas en $\neg \perp$. De esta manera

$$B \wedge \neg E \models \neg \perp \models \neg H$$

Esto implica que $H \models \perp$, donde a $\perp$ se le conoce como la cláusula más específica y es también una cláusula de Horn. Se puede encontrar un subconjunto de soluciones para *H* considerando las cláusulas que subsumen a $\perp$.

Por ejemplo:
Sea B: animal(X) ← mascota(X).

mascota(X) ← perro(X).

Sea E: bonito(X) ← perro(X).

Entonces:

⊥ es: bonito(X) ← perro(X), mascota(X), animal(X).

La implicación inversa es utilizada por PROGOL, sistema de ILP que se describe a continuación y que es utilizado en el desarrollo de esta tesis.

### 2.6.4 PROGOL

PROGOL es un sistema de ILP que construye la hipótesis utilizando implicación inversa (*inversed entailment*) y usa declaraciones de modo para restringir la búsqueda de cláusulas que subsumen a ⊥.

PROGOL efectúa una búsqueda de general a específica (top-down) a través del lattice de subsumciones. Selecciona la cláusula a ser generalizada y encuentra una cláusula consistente que cubra el ejemplo. La selección de ejemplos se repite hasta que se cubren todos los ejemplos. A diferencia de otros sistemas de búsqueda de general a específico, PROGOL procesa las cláusula más específica cubriendo el ejemplo original. La cláusula más específica limita el lattice de $\theta$-subsumciones inferior. En la parte superior, el lattice tiene por límite la cláusula vacía. La estrategia de búsqueda es un algoritmo semejante a A* que utiliza una medida de compresión. Cada vez que se efectúa la búsqueda se regresa una cláusula que garantiza la máxima compresión de los datos, sin embargo, esto no garantiza que se obtenga el conjunto de cláusulas más compacto para los ejemplos dados.

PROGOL utiliza tres algoritmos:

1) Construcción de la cláusula más específica. Obtenerla reduce el espacio de búsqueda.
2) Búsqueda en el lattice de subsumciones (*top-down*). Para cada ejemplo, la búsqueda se limita por , □ ≺ H ≺⊥ donde el límite superior es el elemento más general □ y el límite inferior es el menos general ⊥. Esta es una de las ventajas de PROGOL con respecto a otros sistemas *top-down*.
3) Algoritmo de covering. Selecciona la regla que cubra el mayor número de ejemplos positivos sin cubrir ningún negativo eliminando los positivos y la agrega al conjunto de reglas aprendidas.

La principal ventaja de PROGOL es su restricción en la búsqueda, el limitarla lo hace más eficiente, sin embargo, si existe un número grande de literales su proceso es muy lento.

Es conveniente destacar que aún no es posible determinar el punto óptimo de generalización o especialización para encontrar una hipótesis. Si tenemos pocos ejemplos es posible que se obtenga un modelo sobre-generalizado puesto que se obtendría un conjunto de reglas pequeño pero que no cubre muchos casos. El caso contrario es la sobre-especialización en el que se obtiene un gran número de reglas pero muy adecuadas a los ejemplos dados por lo que el conjunto de reglas no sería útil más que para los ejemplos de entrenamiento.

Otros sistemas de ILP son:

- **FOIL** [Quinlan,1990]. Usa una búsqueda de general a específica basada en el algoritmo de covering de AQ [Michalsky, 1983].

  Encuentra una conjunción de condiciones que cubre algunos ejemplos positivos y ningún negativo y la pone al final como una disyunción de la expresión lógica que está construyendo. Posteriormente elimina los ejemplos que satisfacen la conjunción y si todavía quedan elementos de la clase, se repite el proceso.

- **GOLEM** [Muggleton,1992]. Su algoritmo básico es el siguiente:

Selecciona aleatoriamente dos cláusulas del conjunto de ejemplos positivos.
Encuentra el rlgg.
Si el rlgg no cubre ningún ejemplo negativo
    Elimina todas los ejemplos cubiertas por el rlgg
    Agrega el rlgg al conjunto de reglas
Repetir hasta que se cubran todos los ejemplos positivos

En este capítulo se describieron conceptos generales de aprendizaje y se revisaron técnicas de aprendizaje inductivo de árboles y reglas incluyendo los conceptos fundamentales de programación lógica inductiva. Se hizo énfasis en RIPPER y PROGOL, sistemas que son usados en esta tesis para la extracción de modelos. En el siguiente capítulo se describirá la importancia que tiene el modelo del estudiante como componente de un tutor inteligente.

Capítulo 3

# Modelo del Estudiante

Esta tesis tiene por objetivo la construcción de modelos del estudiante, por ello, en este capítulo se describe la función y la importancia del mismo como componente de un sistema tutor inteligente.

## 3.1 Tutores Inteligentes

La enseñanza es un área de gran interés para la aplicación de técnicas computacionales que permitan facilitar el proceso de aprendizaje en diferentes dominios. En la primera generación de sistemas de apoyo a la enseñanza conocidos como *Computer Aided Instruction systems* (CAI) el conocimiento a transmitir se almacenaba en bloques o frames, que eran diseñados por un experto y desplegados al estudiante bajo determinadas condiciones [Wenger, 1987]. En los 70's, Carbonell fue uno de los principales críticos de estos sistemas ya que decía que no darían resultados de acuerdo al nivel de cada estudiante, hasta que no incluyeran una base de conocimiento similar a la de un experto en el tema. A partir de su trabajo se iniciaron los esfuerzos por desarrollar sistemas de instrucción inteligentes, conocidos como sistemas tutores inteligentes, que se caracterizan por el uso de una base de conocimiento y el uso de técnicas de inteligencia artificial para adaptarse a una situación particular de aprendizaje. Esto permite que la instrucción automatizada se aproxime a una instrucción realizada por un tutor humano.

Los componentes básicos de un tutor inteligente son:

- **Módulo Experto**. En este componente radica el conocimiento que se desplegará al estudiante. Esto incluye explicaciones y procedimientos; tareas y preguntas que el estudiante resolverá. Uno de los aspectos más importantes es que tiene la capacidad de generar múltiples formas de solución por lo que el estudiante puede evaluarse analizando los pasos intermedios que efectúa para resolver un problema.

- **Modelo del Estudiante.** Este componente analiza el comportamiento del estudiante y hace inferencias acerca de su aprendizaje. Su objetivo es individualizar el proceso de enseñanza.

- **Módulo Pedagógico.** Su función es determinar la forma de presentar los tópicos de acuerdo a las necesidades del estudiante; evaluar si es necesaria una intervención, ya sea una sugerencia o una interrupción, y lo que se debe decir en cada caso. Proporciona una guía para desarrollar las actividades, explicaciones y solución. Estas decisiones las hace tomando como referencia el modelo del estudiante y el módulo experto.

- **Interfaz.** Es el componente que permite la comunicación entre el estudiante y el sistema tutor inteligente, además de que permite la presentación del material educativo ( ejercicios, ejemplos, etc.).

Algunos de los primeros sistemas tutores inteligentes fueron SCHOLAR [Carbonell, 1970], SOPHIE [John Seely Brown, R. R. Burton, y J. de Kleer,1970], WUSOR-1 [Carr y Goldstein, 1977] y WEST [Burton y Brown, 1982].

## 3.2 Importancia del modelo del estudiante

El modelo de estudiante es uno de los componentes más importantes de un sistema tutor inteligente ya que la eficiencia del sistema depende en gran parte de la exactitud con la que el conocimiento y comportamiento del estudiante es modelado. La construcción de un buen modelo es una tarea difícil; se tienen que tomar en cuenta muchos factores, desde los aspectos del dominio que hay que incorporar hasta los aspectos de comportamiento y factores externos que ocurren durante el proceso de aprendizaje, que en ocasiones son difíciles de detectar hasta por un instructor humano.

Las tareas de este componente [Wenger,1987], son:

1) Obtener datos del estudiante. Puede ser a través de preguntas (forma explícita) o analizando su comportamiento (forma implícita).

2) Utilizar estos datos para crear una representación del conocimiento del estudiante. Para ello, se obtenían modelos de errores (*buggy model*) para representar el conocimiento del estudiante en términos de desviaciones en comparación con el experto. El sistema usa este modelo para predecir el tipo de respuesta que el estudiante dará en situaciones subsecuentes, compara esa predicción con la respuesta actual del estudiante y usa la información para refinar el modelo.

2) Desarrollar un diagnóstico tanto del estado del conocimiento del estudiante como en términos de la selección óptima de estrategias pedagógicas para presentar la información subsecuente. El diagnóstico se desarrolla mediante un análisis de los datos del estudiante obtenidos por el sistema y actualizando el modelo con base en ello. Uno de los grandes retos es el "ruido", el hecho de que los estudiantes no siempre responden consistentemente, especialmente cuando su conocimiento es frágil y no están seguros sobre las respuestas.

3)

Para construir el modelo del estudiante se han utilizado diferentes técnicas, entre ellas:

**Modelo de *overlay*.** Este modelo es aplicable siempre que el conocimiento del experto sea expresado como un conjunto de reglas. El conocimiento del estudiante es tratado como

un subconjunto del conocimiento del experto y se espera que el subconjunto crezca a medida que el estudiante avanza en su trabajo [WUSOR-II, Carr y Goldstein, 1977]. Su desventaja es que no permite modelar conocimiento incorrecto, sólo conocimiento presente y faltante.

**Modelo diferencial.** Es una extensión del modelo de overlay y consiste en separar el conocimiento del estudiante en dos partes. Una corresponde al conocimiento que le ha sido presentado y la otra es el conocimiento que aun no conoce. Las comparaciones se efectúan tomando en cuenta el conocimiento que ya se presentó, y únicamente en este caso se registran las diferencias [WEST, Burton y Brown, 1982].

**Modelo de errores (*buggy model*) o desviaciones.** Consiste en representar la comprensión de las habilidades del estudiante en una red de procedimientos. Existe un catálogo de posibles errores y el modelo se hace identificando el error cometido y sustituyendo este procedimiento en la red. Al ser una catálogo fijo, limita el modelado a los errores conocidos, otra limitante es que no explica los errores [BUGGY, Brown, et al., 1975].

## 3.3 Consideraciones para la construcción de un modelo

El tipo de conocimiento que el estudiante está adquiriendo es determinante en la técnica de modelado que se aplicará. No es lo mismo construir el modelo en un entorno pasivo, por ejemplo, el aprendizaje de geografía o un idioma nuevo que el representar el desempeño del estudiante en un entorno dinámico como lo es el aprender a controlar un dispositivo. En este tipo de sistemas, el estudiante reacciona ante los eventos que ocurren y modelar su desempeño implica otros factores:

- Se necesita una gran cantidad de datos para poder hacer un modelo útil.
- Los valores de las variables que intervienen en el proceso son números reales, lo que dificulta que el modelo obtenido sea exacto y entendible. Encontrar otra forma de presentar estos valores es una posible solución a este problema.
- El conjunto obtenido de datos presenta mucho ruido.

Obtener modelos en este tipo de dominios no es fácil y se han utilizado diversas técnicas para construir modelos descriptivos y exactos. En el siguiente capítulo se describen algunos trabajos relacionados y se explica en qué consiste el problema del péndulo invertido que es el dominio en el que se desarrolló esta tesis.

Capítulo 4

# Modelado de habilidades en distintos dominios

En este capítulo se describen las dificultades y características del modelado de habilidades humanas, la técnica conocida como clonación de comportamiento, el problema del péndulo invertido y los trabajos relacionados más representativos de esta técnica. En esta tesis se utiliza la clonación de comportamiento para extraer modelos de habilidades en el dominio del péndulo invertido.

## 4.1 Aprendizaje de habilidades humanas

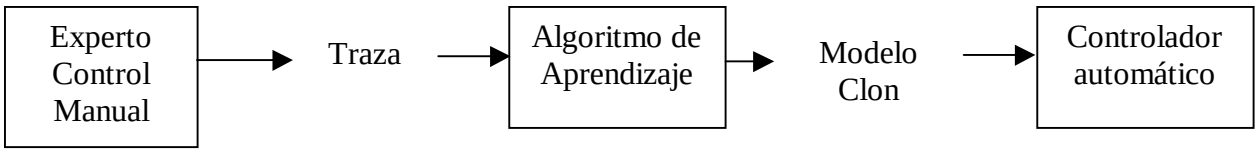
En el aprendizaje de habilidades se combinan dos tipos de conocimiento: en su fase inicial requiere de conocimiento declarativo ("saber qué"), el cual consiste en descripciones, imágenes, ejemplos. En esta fase, la persona es capaz de explicar su conocimiento. El resto del proceso de aprendizaje consiste en perfeccionar la habilidad a través de la práctica; a este tipo de conocimiento se le conoce como procedural ("saber cómo"). Sin embargo, este último se desarrolla a nivel inconsciente y la persona ya no puede explicarlo; se dice que es tácito [Michie, 1995]. Por esta razón, resulta difícil obtener un modelo preciso y entendible de lo que la persona hizo.

Lograr construir un modelo de este tipo de tareas representa grandes beneficios, ya que permiten la construcción de controladores automáticos para entrenar a otras personas [Camacho, 1995] o analizar el comportamiento de los operadores mediante una descripción simbólica. Una técnica que se ha utilizado con resultados favorables para resolver este problema es la que se describe a continuación.

## 4.2 Clonación de comportamiento

La clonación de comportamiento, o *behavioural cloning,* es una técnica para extraer modelos de habilidades de control humanas y su validez ha sido probada en distintos dominios como veremos más adelante. Este término fue utilizado por primera vez por Donald Michie (1990) y consiste básicamente en obtener trazas del comportamiento de un experto humano al desempeñar una tarea de control, procesar esta traza con un algoritmo de aprendizaje, obtener un modelo y posteriormente insertarlo en el sistema controlador.

El siguiente diagrama muestra el proceso:




Figural 4.1 Clonación de comportamiento. A partir del desempeño de un experto se obtiene una traza, ésta se procesa por el algoritmo de aprendizaje y el modelo obtenido es un "clon" del experto que constituirá la base de un sistema controlador.

Una traza es el registro de los valores que toman las variables inherentes al dispositivo controlado y que es almacenado al momento de que el experto humano desempeña la tarea.

En la aplicación de esta técnica se han hecho diversas observaciones [Urbancik, 1994].

- Efecto *clean-up*. El clon inducido presenta un mejor desempeño que el humano del cual se obtuvo la traza. Esto se explica porque los algoritmos de inducción efectúan las mismas acciones para el mismo estado de las variables, considerando las inconsistencias como ruido. El humano, sobre todo inexperto presenta inconsistencias en sus acciones; el experto rara vez varía su estrategia.

  Esta característica es deseable si el objetivo es construir un controlador automático, sin embargo, si lo que se quiere es una descripción exacta de la habilidad del operador, esto no es favorable porque no está reflejando sus errores adecuadamente.

- Los mejores clones se han obtenido cuando se utilizan ejemplos de una sola persona.

- En la presencia de cambios en la tarea; los clones demuestran fragilidad. Esto significa que al modificar parámetros del modelo, como por ejemplo velocidad, el clon no sabe qué hacer y demuestra un comportamiento errático o diferente del esperado.

- Hay pérdida de información importante. Por ejemplo, no puede detectarse la diferencia entre acciones críticas de las que no lo son, ni tampoco es posible detectar si una acción es realizada por estrategia o por error.

Un factor importante es el tiempo de retardo (*delay*), esto es, el intervalo de tiempo entre la percepción de la situación por parte del operador, la decisión de la acción que tomará y su ejecución. Una inadecuada selección del tiempo de retardo puede originar que se registren comportamientos falsos en la traza, generando un modelo erróneo.

En esta tesis la técnica de clonación de comportamiento se aplicó a operadores novatos; a partir de su traza se obtuvo su modelo. El dominio de aplicación en el que se trabajó es el problema del péndulo invertido que a continuación se explica.

## 4.3 El problema del péndulo invertido

El objetivo de esta tarea consiste en mantener en equilibrio un péndulo que se encuentra colocado sobre un carrito que rueda sobre un riel limitado. El péndulo está sujeto a la gravedad, originando que se mueva de un lado a otro y el objetivo es que el usuario aprenda a controlar el carro para evitar que el péndulo se caiga.

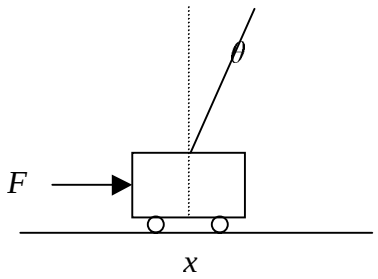



Figura 4.2 **El péndulo y el carro**. $\theta$ = 0 es el ángulo de inclinación del péndulo en equilibrio. Para $\theta$<0, el péndulo se inclina a la izquierda. Para $\theta$>0, se inclina a la derecha. $x$ es la posición del carro. $x$ < 0 significa que el carro está en la mitad izquierda del riel, mientras que para $x$>0 significa que el carro está en la mitad derecha.

Para lograr el objetivo, se permite aplicar una fuerza predefinida al carro cada vez. Esta fuerza puede aplicarse solamente de cualquiera de los lados del carro con una magnitud predefinida y constante. A esto se le conoce como método de control "bang-bang".

El estado del sistema está definido por cuatro elementos: posición del carrito, velocidad del carro, ángulo del péndulo y velocidad angular del péndulo. A partir de esta información, se debe determinar la dirección en la cual se empuja el carrito, aplicando una fuerza constante a la derecha o izquierda.

De acuerdo a [Wieland,1991], este es uno de los problemas prototipos de control y es conocido también como el problema del péndulo invertido, *cart-pole* o equilibrio de la escoba. Es un problema de interés porque describe un sistema inherentemente inestable que es representativo de una amplia gama de problemas. Resulta muy útil para experimentación; sin embargo, tiene la característica de que es una tarea fácil de aprender y repetitiva por lo que un operador humano se aburre rápidamente.

## 4.4 Trabajos relacionados

Se han realizado experimentos para extraer modelos de las habilidades de control humano utilizando técnicas de aprendizaje. El objetivo principal de estos trabajos es la construcción de controladores automáticos. A continuación se describirán los más representativos.

### El carro y el péndulo.

[Michie,1994]. El experimento se efectuó entrenando a personas a equilibrar el péndulo con un joystick en un simulador, registrando su comportamiento. El algoritmo de inducción utilizado fué C4.5 y el retardo se fijó en 400 ms.

Las reglas se evaluaron de dos formas, en base a su capacidad predictiva y en la habilidad de sustituir al humano en la tarea. En la primer forma de evaluación, el modelo no predijo muy acertadamente la acción que la persona hubiera efectuado, mostrando un porcentaje de error alto, generalmente mayor al 20%, sin embargo, cuando los modelos inducidos fueron probados en el simulador, el desempeño fue mejor al del humano, observándose un efecto de *clean-up*.

### Habilidades de vuelo

- [Sammut y Michie,1992]. Los datos fueron obtenidos de un simulador de vuelo en el que se aprendía a controlar un avión con un plan de vuelo predeterminado. Algunas de las tareas correspondientes eran: despegue, ascenso, descenso y aterrizaje. Como ejemplo de las variables involucradas en estas tareas se pueden mencionar: posición del timón, posición de los alerones, altitud y velocidad del aire entre otras. El simulador se modificó para que registrara las acciones realizadas por el humano en trazas de comportamiento. Las tareas se dividieron en grupos, se obtuvieron clones especializados para cada uno de ellos y se siguió un plan de vuelo predefinido. Los modelos obtenidos fueron inducidos en árboles de decisión y fueron probados corriendo el simulador en piloto automático. Los algoritmos utilizados fueron C4.5 y CART. En las pruebas, se observó que el piloto automático se comportaba con las características propias del humano, sin embargo, su desempeño era más consistente debido al efecto *clean-up*.

- [Camacho y Sammut, 1990]. Extrajo los modelos de habilidades de vuelo de pilotos de un avión F-16 utilizando un simulador. Se manejaron dos enfoques: árboles de decisión, para el que se usó el algoritmo C4.5 y programación lógica inductiva, en el que se usaron dos sistemas: FOIL e Indlog.

  Los datos se dividieron en dos grupos, el primero se usó para construir los modelos y el segundo para probar la capacidad de predicción del modelo resultante. Para las pruebas con ILP se redujo el conjunto de atributos y se aprendieron dos teorías.

Los árboles de decisión tuvieron un porcentaje de exactitud predictiva bajo, esto se explica por el tipo de valor de los atributos. El controlador puede hacer una réplica del vuelo hecho por el humano, pero su poder preditivo es bajo. El uso de ILP tampoco mostró una mejora en el poder predictivo. Se lograron obtener modelos reducidos, con un pequeño incremento en su exactitud y robustez en comparación a experimentos realizados previamente.

- [Camacho,1994]. Modificó la estrategia anterior incorporando el uso de metas y acciones de control adaptivas que se utilizan para corregir la acción previa. Este nuevo modelo reduce la complejidad de los árboles por lo que los modelos son compactos y lograron una mayor robustez en los controladores. Su desempeño fue cercano al del humano bajo situaciones similares.

## Control de una grúa

[Urbancik,1994]. El problema es el control de una grúa que introduce y saca contenedores de los barcos. Se desea minimizar el tiempo de transportar un contenedor. La grúa está constituida por una polea que se mueve hacia adentro y hacia fuera y una cuerda que se mueve hacia arriba y hacia abajo.

Las variables del sistema son seis: posición-velocidad de la polea, longitud-velocidad de la cuerda y ángulo-velocidad de la cuerda. Las acciones que el operador puede efectuar son dos: aplicar una fuerza $F_x$ a la polea y aplicar una fuerza $F_l$ al cable.

Las trazas se obtuvieron de operadores que trasladaban una caja hacia un bote. Ellos desconocían el tipo de sistema que estaban controlando y únicamente se registraron los traslados exitosos, se les pidió también que describieran la estrategia que siguieron.

Para inducir los modelos se usó RETIS [Karalic, 1992] y M5 [Quinlan, 1992] y se obtuvieron los siguientes resultados:

- Los clones obtenidos mostraron el efecto *clean-up* y no eran confiables puesto que no se comportaban igual al operador del que se extrajo el modelo. Eran además sensibles a cambios en los parámetros del modelo; para este sistema, los parámetros son la masa de la carga, la masa de la polea, la longitud de la cuerda, y la gravedad entre otros.

- Se observó también que las descripciones que los humanos daban de lo que hacían no correspondía a sus acciones reales, por lo que el aprendizaje computacional puede ser una forma de obtener resultados más precisos en este aspecto.

- RETIS y M5 no permiten representar los planes de acción de la misma forma en que lo hace un humano.

## 4.5 Modelado participativo

Generalmente, el modelo del estudiante es un módulo del cual el estudiante no está consciente. Se limita a realizar las actividades que el sistema le indica. En su trabajo de tesis doctoral, Morales, R., demostró que la participación activa del estudiante en la tarea de modelado es de gran ayuda para mejorar la capacidad de describir mejor sus acciones.

Su dominio de aplicación es el aprendizaje de habilidades para el control del péndulo invertido. Para construir los modelos de los estudiantes, utilizó la técnica de clonación de comportamiento. La variante que manejó consiste en que las trazas fueron obtenidas de participantes novatos, no de un experto, como es la técnica original.

Implementó un sistema (PACMOD) para construir modelos de novatos que aprenden a controlar el péndulo invertido. El algoritmo de aprendizaje seleccionado fué RIPPER [Cohen,1992], ya que su salida, un conjunto de reglas de producción es una representación natural y entendible, además de que permite un mejor control en el proceso de inducción que, por ejemplo, C4.5.

Un ejemplo de los modelos obtenidos es el siguiente:

right 163 20 **if** $\dot{x} \leq$ -1.454 **and** $a \geq$ 0.029 .
left 164 24 **if** $\dot{x} \geq$ 1.446 **and** $a \leq$ 0.030 .
left 163 24 **if** $x \geq$ 1.280 **and** $a \leq$ 0.017 .
right 383 76 **if** $x \leq$ 0.852 **and** $a \geq$ 0.145 .
left 383 83 **if** $x \geq$ -0.794 **and** $a \leq$ -0.073 **and** $\dot{a} \leq$ -0.022 .
left 440 110 **if** $x \geq$ -1.214 **and** $a \leq$ -0.132 .
right 159 43 **if** -1.086 $\leq x \leq$ 1.592 **and** $a \geq$ -0.098 **and** 0.330 $\leq \dot{a} \leq$ -0.973 .
right 328 92 **if** $x \leq$ 1.931 **and** $\dot{x} \leq$ 1.079 **and** $a \geq$ -0.100 **and** $\dot{a} \geq$ 0.274 .
right 378 121 **if** $a \geq$ 0.120 **and** $\dot{a} \leq$ 0.892 .
left 325 110 **if** $x \geq$ -2.163 **and** $\dot{x} \geq$ -1.426 **and** $a \leq$ 0.118 **and** $\dot{a} \leq$ -0.348 .
wait 0 0 .

donde $\dot{a}$ es la velocidad angular, $a$ es el ángulo del péndulo, $\dot{x}$ es la velocidad del carro y $x$ es la posición del carro. El par de números a la izquierda de la acción corresponden al número de positivos y falsos positivos respectivamente para la regla.

Entonces, la primera regla se interpretaría así: Si la velocidad del carro es menor o igual a -1.454 y el ángulo del péndulo es mayor o igual a 0.029 entonces empuja el carro a la derecha; esto es, se empuja a la derecha cuando el carro se desplaza rápidamente a la izquierda mientras el péndulo está inclinado a la derecha, aunque esté muy cerca del equilibrio.

Los modelos obtenidos son predictivos, por lo que son útiles para ser analizados por los participantes y permiten ser evaluados.

Después de estos estudios, se llevó a cabo un experimento para mostrar los efectos de la participación activa del estudiante en el proceso de modelado, en él participaron 15 personas.

A los participantes se les solicitó tratar de controlar el péndulo al menos por 7 minutos sin ninguna distracción. Tuvieron un periodo de familiarización de 1 minuto, los siguientes 5 fueron para tratar de controlar el dispositivo. Los modelos obtenidos corresponden a éstos últimos 5 minutos y tenían de 5 a 17 reglas. Se comprobó que los modelos tenían un poder predictivo de 70% aproximadamente.

La siguiente fase consistió en mostrar a los participantes los modelos que fueron presentados usando una interfaz gráfica, en forma de tabla donde cada renglón representaba una regla. Era necesaria una explicación previa de cada representación gráfica y la correspondiente acción. Una dificultad es que un estado puede satisfacer más de una regla, por lo que el orden en que se encuentran las reglas en cada modelo es importante. El participante podía explorar su modelo y ejecutarlo iniciando con un estado que favoreciera la ejecución de una regla en particular. También podía modificar las precondiciones sobre el ángulo y la velocidad angular del péndulo, así como la posición y la velocidad del carrito, manipulando directamente la presentación gráfica. Adicionalmente, se generaba una descripción en forma textual de la regla. Se expresaba en forma de regla if...then, donde las precondiciones y acciones se traducían a palabras. Esto se hizo dividiendo en intervalos los posibles valores de cada variable y asociándoles una palabra descriptiva. El resultado fue una explicación extensa y repetitiva pero que resultó útil para aclarar el significado de la descripción gráfica de las reglas.

Se evaluó entonces la facilidad con la que los participantes podían entender los modelos. En los resultados se observó que la principal confusión se debía a la poca confianza en su interpretación de la interfaz gráfica, por lo que ésta fue mejorada. Los resultados obtenidos mostraron que los participantes sí lograron entender, con cierto nivel de detalle los datos mostrados por el sistema.

Los siguientes experimentos tuvieron por objetivo probar los cambios en el desempeño de los novatos después de inspeccionar sus modelos. Lo que se esperaba es que los participantes fueran capaces de reportar mejor su conocimiento sobre la tarea e inclusive usarlo mejor en otras tareas diferentes a la original. Los resultados obtenidos indican que la participación activa del estudiante incrementa su habilidad para articular su conocimiento; es decir, está más consciente del mismo. De esta manera, el estudiante es capaz de formular mejor su estrategia de juego. No se mostró mejoría en otras tareas como transferencia de conocimiento y flexibilidad.

## 4.6 Metodología propuesta

El trabajo anterior presenta una solución al problema de interpretación de los modelos, Esta consiste en mostrar diferentes representaciones de los mismos a través de una interfaz gráfica. Una de las principales dificultades observadas fue la confusión de los participantes para entender la representación de las reglas aunque después de recibir explicaciones y utilizando la forma textual como apoyo, se observó una mejoría en su comprensión. La verbalización obtenida no resulta muy satisfactoria y la forma en que se realiza es básicamente una traducción a texto del modelo numérico obtenido utilizando rangos de valores. Si el modelo numérico completo fuera traducido a palabras, ¿tendría la misma exactitud que el modelo original?

Al observarse la utilidad de esta representación resulta deseable obtener una forma textual que sea clara, no repetitiva y que represente adecuadamente el desempeño del participante. En esta tesis se presenta una solución a este problema. Para lograrlo, se transforman las trazas numéricas originales en valores cualitativos agregando conocimiento del dominio. La inducción de los modelos se hace utilizando dos enfoques: aprendizaje inductivo de reglas (RIPPER), sistema usado también en el trabajo previo [Morales. R., 2000] y programación lógica inductiva (PROGOL). En el primer enfoque, el conocimiento del dominio se agrega a través de un pre-procesamiento de la traza mientras que en el segundo caso, el sistema permite incorporarlo directamente. Se espera que el agregar conocimiento del dominio permita obtener modelos más claros y exactos. En el siguiente capítulo se detalla la metodología que se siguió y se muestran los resultados obtenidos.

Capítulo 5

# Experimentos y resultados

Como se planteó en el capítulo 1, el objetivo de este trabajo es construir modelos del estudiante que sean descriptivos y fáciles de interpretar. Se espera que incorporar conocimiento del dominio proporcionará mayor claridad y facilidad de interpretación al modelo.

El dominio de aplicación es el control del péndulo invertido y se utiliza la técnica de clonación de comportamiento para trazas de novatos; se aplica también en la traza de un experto con el fin de observar las diferencias entre ambos. Para la inducción de los modelos se utilizaron los algoritmos RIPPER y PROGOL.

En este capítulo se describe la metodología desarrollada y los resultados obtenidos.

## 5.1 Datos originales

Los datos utilizados en este trabajo son las trazas correspondientes a 22 participantes que tomaron parte en sesiones de 5 minutos de aprendizaje de control del péndulo invertido practicando en el sistema PACMOD [Morales, R., 2000]. También se obtuvo el modelo de la traza de un experto [Freyre, J., 2000]. Los valores de esta última traza presentan únicamente de 2 a 3 decimales a diferencia de las trazas de los participantes cuyos valores tienen de 5 a 6 decimales. Las trazas constan de 4 atributos: velocidad angular ($\dot{a}$), ángulo $a$, velocidad del carro ($\dot{x}$) y posición del carro ($x$), todos con valores continuos. La clase puede tomar cualquiera de los valores *right*, *left* o *inertia*. A continuación se muestran 3 ejemplos de una traza original:

| $\dot{a}$ | $a$ | $\dot{x}$ | $x$ | clase |
|---|---|---|---|---|
| -1.32811 | 0.138182 | 2.4245 | 0. 915903 | *right* |
| -1. 57452 | 0. 11162 | 2.61763 | 0. 964393 | *left* |
| -2. 37603 | 0. 001579 | 3. 20006 | 1. 13308 | *inertia* |

Tabla 5.1 Traza original.

Las trazas de los participantes tienen entre 1320 y 1902 ejemplos mientras que la del experto tiene 3530. Esta última tiene un mayor número de ejemplos que el resto porque se observó que existen pocos cambios de acción. Esto significa que hay bloques de ejemplos que permanecen iguales, por lo tanto, se decidió que incluyera más ejemplos a fin de que los sistemas de inducción pudieran obtener más información sobre las diversas acciones que realiza el experto. A continuación vemos un bloque de ejemplos de la traza del experto.

### Ejemplo de traza del experto.

-0.001,-0.027,-0.05,-0.632,inertia.
-0.003,-0.027,-0.05,-0.633,inertia.
-0.004,-0.027,-0.05,-0.633,inertia.
-0.006,-0.027,-0.05,-0.633,inertia.
-0.007,-0.027,-0.05,-0.634,inertia.
-0.009,-0.027,-0.05,-0.634,inertia.
-0.01,-0.027,-0.05,-0.634,inertia.
-0.012,-0.027,-0.05,-0.635,inertia.
-0.013,-0.027,-0.05,-0.635,inertia.
-0.015,-0.027,-0.05,-0.635,left.
0.01,-0.028,-0.08,-0.635,left.
0.034,-0.027,-0.12,-0.636,left.
0.058,-0.027,-0.15,-0.637,left.
0.082,-0.027,-0.18,-0.638,left.
0.107,-0.026,-0.22,-0.639,left.
0.131,-0.026,-0.25,-0.64,left.
0.155,-0.025,-0.29,-0.642,left.
0.18,-0.024,-0.32,-0.643,left.

## 5.2 Metodología

La metodología que se siguió para la construcción de los modelos consiste básicamente en los siguientes pasos.

1) Pre-procesar la traza original, transformando los valores contínuos en valores cualitativos incorporando conocimiento del dominio. Esto se hizo de tres formas diferentes:

    - Obteniendo tendencias.
    - Agregando aceleración.
    - Agregando dirección del péndulo.

    Se obtienen así tres trazas con valores nominales para cada participante.

2) Inducir el modelo con RIPPER para cada una de las tres trazas obtenidas en 1). El resultado es un conjunto de reglas if..then fáciles de interpretar.

3) Plantear el conocimiento del dominio, expresar los ejemplos en forma de programas lógicos (Prolog) y definir el predicado que se va a aprender.

4) Transformar las traza original en hechos (Prolog) de acuerdo al predicado definido en 3).

5) Inducir el modelo en PROGOL, obteniendo un conjunto de reglas de Prolog que describe el desempeño del participante.

6) Analizar resultados.

El siguiente diagrama muestra la metodología que se siguió:

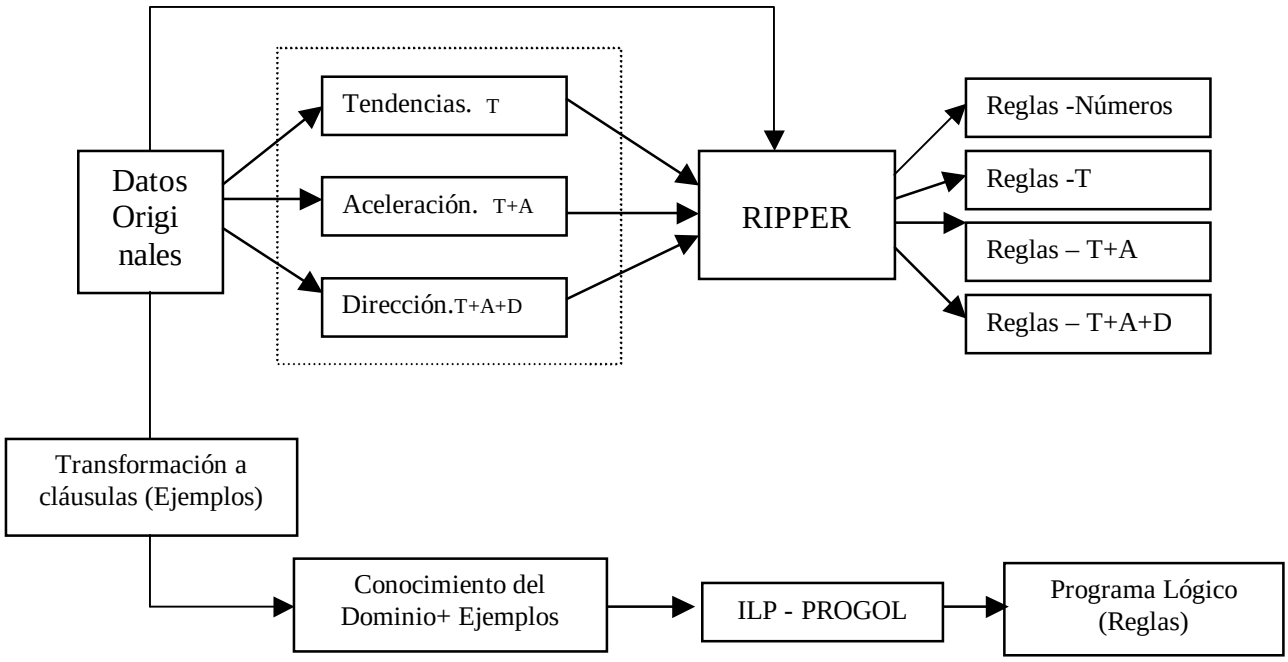



Figura 5.1 Diagrama de la metodología para obtención de modelos. Tanto para RIPPER como para PROGOL se agrega como conocimiento del dominio las tendencias, aceleración y dirección. La diferencia es que para RIPPER esto se hace pre-procesando la traza antes de introducirla al sistema. Por esta razón se representan las transformaciones en tres bloques separados. PROGOL permite incorporar conocimiento del dominio directamente en su definición.

## 5.3 Pre-procesamiento

Como se mencionó en la sección anterior, se realizaron tres transformaciones a la traza original: (1) en base a las tendencias, (2) a la aceleración y (3) a la dirección del péndulo y el carro. Para esta etapa se desarrolló una herramienta para pre-procesar las trazas. Véase el apéndice I.

A continuación se describe en qué consiste cada transformación:

1) **Tendencias.** Los valores de los atributos se transformaron de acuerdo a su comportamiento ascendente o descendente, los valores nominales que cada atributo puede tomar son: “aumenta”, “disminuye” o “noCambia”, éste último valor se

obtiene en base a un umbral (véase sección 5.4). Se requieren dos estados (ejemplos) para obtener su tendencia.

| **Atributos** | | **Valores** |
|---|---|---|
| velocidadAngular ( $\dot{\alpha}$ ) | = | { aumenta, disminuye, noCambia } |
| anguloPendulo ( $\alpha$ ) | = | { aumenta, disminuye, noCambia } |
| velocidadCarro ( $\dot{x}$ ) | = | { aumenta, disminuye, noCambia } |
| posicionCarro ( $x$ ) | = | { aumenta, disminuye, noCambia } |
| clase | = | { *left*, *right*, *inertia* } |

Por ejemplo, la traza de la Tabla 5.1 quedaría:

| $\dot{\alpha}$ | $\alpha$ | $\dot{x}$ | $x$ | clase |
|---|---|---|---|---|
| disminuye | noCambia | aumenta | aumenta | *right* |
| disminuye | noCambia | aumenta | aumenta | *left* |
| disminuye | disminuye | aumenta | aumenta | *inertia* |

Tabla 5.2 Tendencias

2) **Aceleración.** Se agregaron dos atributos a la traza, éstos son: la aceleración angular ( $\ddot{\alpha}$ ) y la aceleración del carro ( $\ddot{x}$ ). Los valores que los nuevos atributos pueden tomar son: “aumenta”, ”disminuye” o ”constante”. Los valores del atributo $\alpha$ se cambian por: “alaDerecha”, “alaIzquierda” o “noCambia”, que indican la dirección en la que se mueve se mueve el péndulo, información que no es posible obtener utilizando únicamente las tendencias. Se utilizaron tres estados para obtener la aceleración. En la figura 5.2 se muestra el comportamiento del péndulo de acuerdo a estos valores.

| **Atributos** | **Valores** |
|---|---|
| aceAngular ( $\ddot{\alpha}$ ) | = { aumenta, disminuye, constante } |
| velocidadAngular ( $\dot{\alpha}$ ) | = { aumenta, disminuye, noCambia } |
| anguloPendulo ( $\alpha$ ) | = { alaIzquierda, alaDerecha, noCambia } |
| aceCarro ( $\ddot{x}$ ) | = { aumenta, disminuye, constante} |
| velocidadCarro ( $\dot{x}$ ) | = { aumenta, disminuye, noCambia } |
| posicionCarro ( $x$ ) | = { aumenta, disminuye, noCambia } |
| clase | = { *left*, *right*, *inertia* } |

Además de los dos atributos de aceleración se agregó al péndulo la dirección en la que se mueve si está cayendo o levantándose. No se consideró de la misma manera la posición del carro ya que la atención se centró en el resultado del modelo al agregar los nuevos atributos.

anguloPendulo = “alaDerecha” anguloPendulo = “alaIzquierda”


Figura 5.2 Interpretación de los valores “alaDerecha” y “alaIzquierda” para $\alpha$**.** Este comportamiento no indica si el péndulo cae o se levanta.

Este es un ejemplo de la traza transformada.

| $\alpha$ | $\dot{\alpha}$ | $a$ | $x$ | $\dot{x}$ | $x$ | Clase |
|---|---|---|---|---|---|---|
| constante | disminuye | noCambia | constante | aumenta | aumenta | *right* |
| constante | disminuye | noCambia | constante | aumenta | aumenta | *left* |
| disminuye | disminuye | alaIzquierda | aumenta | aumenta | aumenta | *inertia* |

Tabla 5.3 Aceleración

3) **Dirección.** En esta transformación se modificaron los valores de $\alpha$ ya que en el caso anterior no se sabe si el péndulo cae o se levanta. El atributo puede tomar los siguientes valores:“levantandoDerecha”,“cayendoIzquierda”, levantandoIzquierda”, “cayendoDerecha” o “equilibrio”. También se modificaron los valores de $x$ para indicar si el carro se aleja o acerca del centro y si se encuentra a la izquierda o a la derecha del centro. Los valores que puede tomar ahora son: “alejaIzq”,”alejaDer”,”acercaIzq”,”acerdaDer”,”noCambia”. En la figura 5.3 se muestra la interpretación de estos valores en el péndulo.

**Atributos** **Valores**

aceAngular ( $\ddot{\alpha}$ ) = { aumenta, disminuye, constante }
velocidadAngular ( $\dot{\alpha}$ )= { aumenta, disminuye, noCambia }
anguloPendulo ( $\alpha$ ) = { levantandoDerecha,cayendoIzquierda,
levantandoIzquierda, cayendoDerecha,equilibrio}
aceCarro ( $\ddot{x}$ ) = { aumenta, disminuye, constante}
velocidadCarro ( $\dot{x}$ ) = { aumenta, disminuye, noCambia }
posicionCarro ( $x$ ) = { alejaIzq, alejaDer,acercaIzq,acercaDer,noCambia }
clase = { *left*, *right*, *inertia* }

## Péndulo

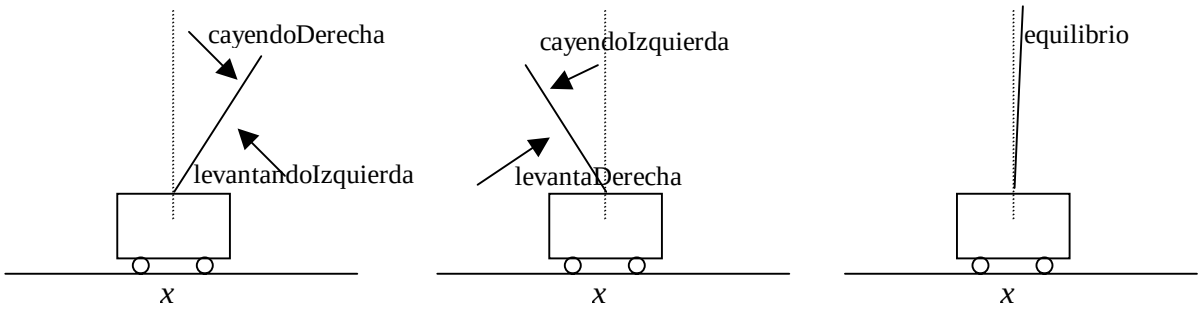



Figura 5.3 Interpretación de los valores "levantandoDerecha", "cayendoIzquierda", "levantandoIzquierda", "cayendoDerecha" y "equilibrio" para ( $a$ ). De esta manera ya es posible identificar el estado del péndulo.

## Carro

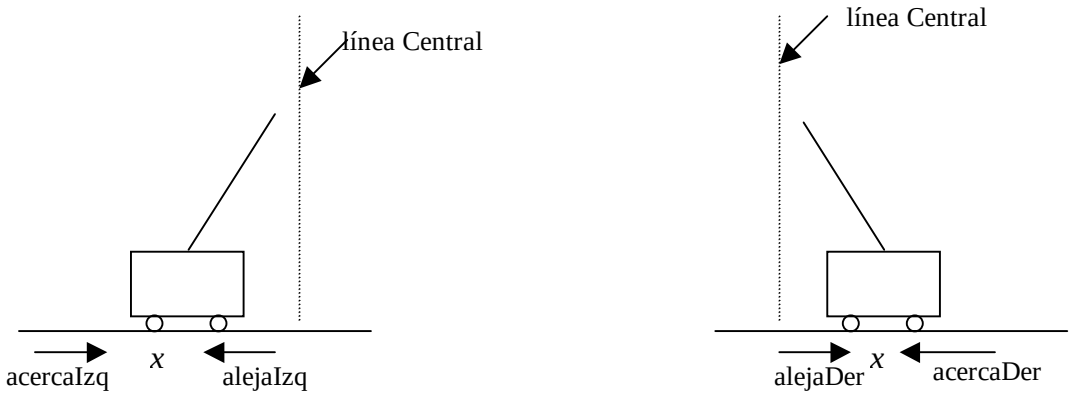



Figura 5.4 Interpretación de los valores, en este caso indican en qué parte se encuentra el carro y si se aleja o acerca al centro.

| $\dot{a}$ | $\dot{a}$ | $a$ | $\dot{x}$ | $\dot{x}$ | $x$ | Clase |
|---|---|---|---|---|---|---|
| constante | disminuye | levantandoIzquierda | constante | aumenta | alejaDer | *right* |
| constante | disminuye | levantandoIzquierda | constante | aumenta | alejaDer | *left* |
| disminuye | disminuye | levantandoIzquierda | aumenta | aumenta | alejaDer | *inertia* |

Tabla 5.4 Dirección

## 5.4 Inducción de modelos en RIPPER

Antes de obtener los modelos para las trazas pre-procesadas, se obtuvieron los modelos de la traza original con el fin de observar el porcentaje de error obtenido por el modelo, los números de reglas obtenidos y la claridad de las reglas.

Es conveniente recordar que los participantes, de los que se obtuvieron las trazas, son novatos por lo que los porcentajes de error que los modelos generan son relativamente altos en comparación con los porcentajes de error generados por el modelo del experto, ya que presentan muchas inconsistencias en sus acciones. En la tabla 5.5 se concentran los resultados.

### 5.4.1 Definición de umbral

Como se presentó en la sección anterior, existen atributos con el valor "noCambia" o "constante". Este valor indica que el cambio entre un estado y otro del atributo es poco significativo. Esto quiere decir que, si en un estado, la posición del carro es 0.56268 y en el siguiente estado es 0.56269, en realidad, no existe un cambio notable en la posición como para darle el valor "aumenta".

Pero ¿cómo determinar el valor de "poco significativo"?. El umbral que define el valor "noCambia" o "constante" es importante para mantener o disminuir el porcentaje de error original del modelo y obtener reglas claras. Valores de umbral muy pequeños o muy grandes no son de utilidad ya que generan un incremento notable en el porcentaje de error del modelo, en la mayoría de los casos lo duplican, además de que las reglas obtenidas no describen estados aceptables. Se observó que para los participantes 1 y 4 la variación de umbral no afecta significativamente el error de sus modelos. Estos participantes presentan características especiales. En el caso del participante 1 se observa una inconsistencia importante en su comportamiento; de acuerdo a la tabla 5.5 vemos que realiza cambios constantes en sus acciones lo que se refleja en un alto porcentaje de error (30.55 %) obtenido por el modelo generado directamente de la traza numérica. Para el participante 4 el caso es opuesto ya que el porcentaje de error obtenido por el modelo generado a partir de la traza original es el más bajo de todos los participantes (10.81 %) y esta característica la conserva para todas las transformaciones.

Considerando lo anterior, se seleccionó a un participante que no tuviera porcentajes de error muy altos o muy bajos a partir de su traza original. En este caso, se eligió al participante 3 para determinar un umbral general ya que presenta un porcentaje de error intermedio (17.42 %) . Primero se obtuvo un umbral para tendencias variando los valores hasta obtener un modelo con un porcentaje de error aproximado al original o menor si era posible. En cada prueba se verificaba que las reglas fueran útiles. Esto significa que si el valor de umbral era muy alto, entonces se obtenían muchos valores de "noCambia" por lo que las reglas generadas no describían situaciones reales. Una vez determinado un valor adecuado, se procedió a obtener el umbral para aceleración, manteniendo fijo el umbral de tendencias obtenido previamente y siguiendo los mismos criterios de selección. No se hicieron pruebas

variando simultáneamente los umbrales para tendencias y aceleración por lo que no puede garantizarse que los valores obtenidos sean realmente los mejores. A continuación se muestran las gráficas que representan los valores probados para determinar el umbral para tendencias y aceleración para el participante 3.

**Participante 3 (Error: 17.42%)**

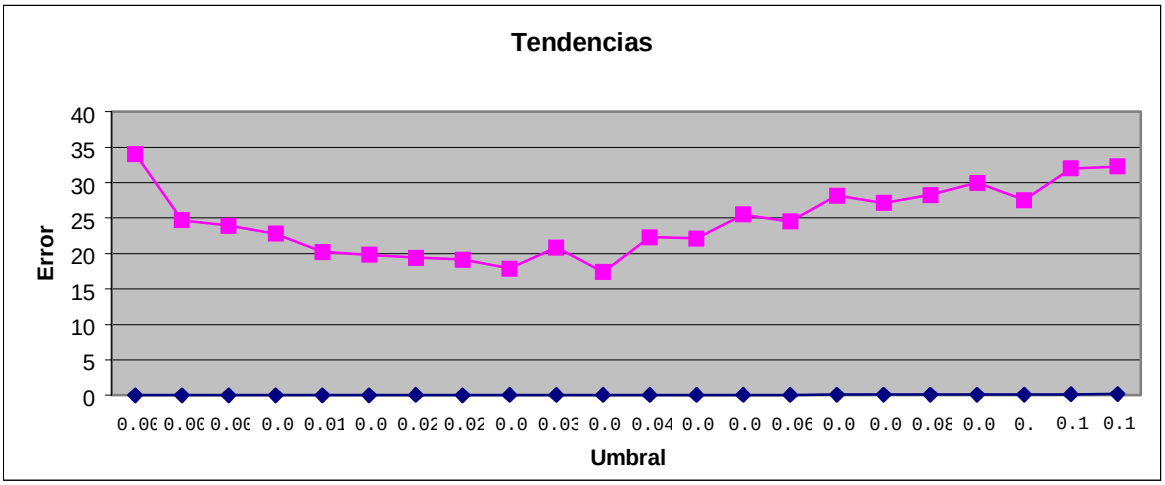



Figura 5.5 Obtención del umbral para tendencias. Con 0.04 se obtiene el mínimo error.

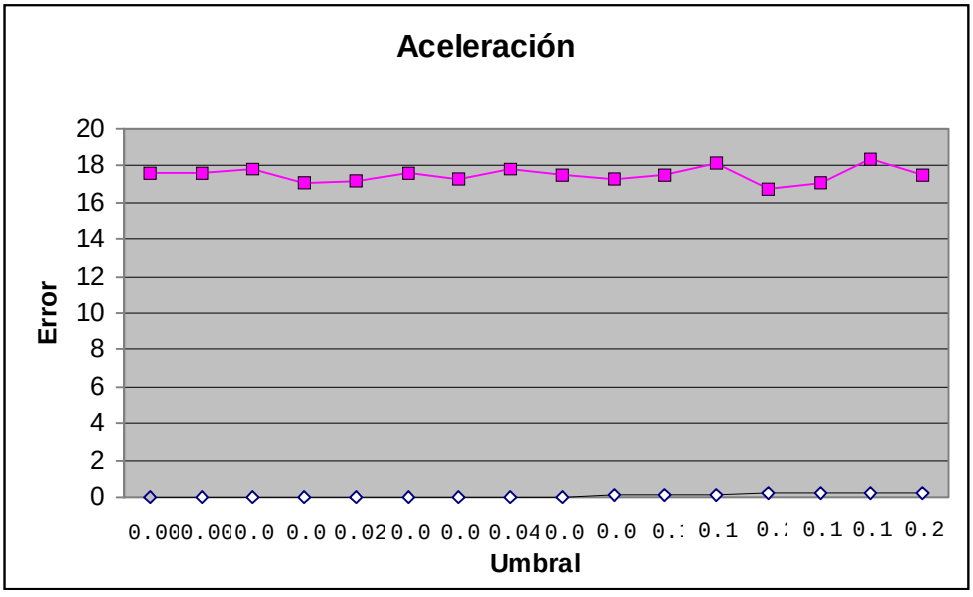



Figura 5.6 Obtención del umbral para aceleración. Para 0.021 se obtiene el mínimo error

Una vez obtenidos los umbrales se obtuvieron los modelos para las tres trazas de cada participante.

## 5.4.2 Resultados

Los modelos inducidos por RIPPER se obtuvieron utilizando el parámetro *–agiven*, que ordena las clases de acuerdo a su definición en el archivo correspondiente. En este caso, el orden de las clases es: *right, left, inertia.*

Se obtuvieron también los resultados utilizando el parámetro *–aunordered*, que separa cada clase de las clases restantes, de esta manera existen reglas para cada clase (Apéndice II).

A continuación se muestran los resultados obtenidos:

| | | | | | Numéricos | | Tendencias | | Aceleración | | Ace/Dirección | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| P | Cambios | *right* | *left* | *inertia* | % Error | # Reglas | % Error | # Reglas | % Error | # Reglas | % Error | # Reglas |
| 1 | 1500 | 894 | 897 | 1924 | 30.55 | 11 | 47.34 | 1 | 43.05 | 9 | 38.36 | 5 |
| 2 | 505 | 591 | 452 | 498 | 19.97 | 9 | 28.16 | 9 | 29.50 | 9 | 20.01 | 9 |
| 3 | 509 | 695 | 661 | 354 | 17.42 | 14 | 17.37 | 10 | 16.45 | 19 | 16.74 | 15 |
| 4 | 414 | 1027 | 819 | 234 | 10.81 | 13 | 10.24 | 10 | 8.95 | 19 | 9.10 | 16 |
| 5 | 427 | 572 | 639 | 391 | 21.21 | 14 | 23.60 | 11 | 20.19 | 17 | 20.69 | 16 |
| 6 | 348 | 416 | 534 | 376 | 22 | 11 | 33.33 | 8 | 34.44 | 8 | 30.97 | 10 |
| 7 | 440 | 949 | 896 | 212 | 13.56 | 16 | 11.58 | 12 | 9.69 | 18 | 10.61 | 16 |
| 8 | 461 | 360 | 482 | 545 | 19.96 | 10 | 23.58 | 10 | 20.58 | 16 | 20.58 | 16 |
| 9 | 387 | 570 | 660 | 312 | 21.97 | 9 | 29.77 | 11 | 26.36 | 14 | 25.52 | 9 |
| 10 | 444 | 453 | 571 | 398 | 22.14 | 15 | 32.91 | 9 | 32.32 | 13 | 30.99 | 11 |
| 11 | 453 | 892 | 790 | 282 | 13.28 | 16 | 15.17 | 11 | 14.42 | 14 | 12.69 | 17 |
| 12 | 499 | 729 | 550 | 318 | 24.28 | 12 | 34.31 | 8 | 33.48 | 13 | 30.53 | 16 |
| 14 | 515 | 687 | 613 | 381 | 22.06 | 11 | 22.07 | 11 | 20.01 | 17 | 18.64 | 16 |
| 16 | 510 | 717 | 515 | 381 | 25.03 | 9 | 28.77 | 10 | 26.75 | 13 | 26.44 | 17 |
| 17 | 546 | 900 | 722 | 252 | 19.68 | 16 | 20.97 | 14 | 17.74 | 17 | 16.61 | 22 |
| 18 | 587 | 649 | 560 | 404 | 20.01 | 16 | 26.91 | 12 | 25.02 | 15 | 21.97 | 15 |
| 19 | 542 | 677 | 836 | 305 | 19.96 | 13 | 19.97 | 13 | 17.46 | 19 | 17.29 | 19 |
| 20 | 395 | 611 | 606 | 320 | 17.49 | 14 | 25.76 | 9 | 23.71 | 22 | 22.93 | 14 |
| 21 | 468 | 802 | 768 | 283 | 14.89 | 19 | 13.82 | 12 | 13.02 | 20 | 14.75 | 17 |
| 25 | 348 | 560 | 550 | 233 | 19.79 | 12 | 31.05 | 8 | 27.96 | 15 | 28.26 | 12 |
| 27 | 426 | 499 | 496 | 320 | 30.78 | 15 | 33.92 | 5 | 30.62 | 13 | 36.33 | 8 |
| 29 | 490 | 796 | 777 | 282 | 13.79 | 16 | 15.74 | 12 | 15.92 | 13 | 16.62 | 17 |
| | | | | | | | | | | | | |
| Prom. | | | | | 20.03 | 13.23 | 24.83 | 9.82 | 23.07 | 15.14 | 22.12 | 14.23 |
| D.S. | | | | | 5.06 | 2.78 | 9.00 | 2.81 | 8.75 | 3.69 | 7.99 | 4.06 |
| | | | | | | | | | | | | |
| Exper | 61 | 224 | 215 | 1897 | 0.88 | 19 | 2.62 | 2 | 2.65 | 2 | 2.66 | 3 |

Tabla 5.5 Resultados para los modelos inducidos con RIPPER

La columna “Cambios” indica el número de transiciones entre acciones, es decir, cuántos cambios *right-left, left-right, right-inertia, inertia-right, inertia-left, left-inertia* existen. Esta información es útil para analizar el desempeño del participante. Puede observarse que en el rango de 400-499 cambios se encuentran los menores porcentajes de error ( inferiores a 15% ) tomando en cuenta únicamente a los participantes. Para el caso del experto esto no es válido porque presenta 61 cambios únicamente a pesar de que su traza es la que contiene el mayor número de ejemplos. Como se comentó previamente, su traza presenta bloques grandes de ejemplos iguales antes de realizar una acción diferente. Por esta razón, aunque el número de ejemplos es mayor que el de los participantes, el número de cambios es menor.

Las columnas “*right*”, “*left*” e “*inertia*” indican el número acciones para cada clase efectuadas. Se presentan primero los resultados de los modelos generados a partir de la traza numérica en términos de porcentaje de error y número de reglas. Finalmente se muestran los promedios y los resultados para los modelos del experto. El conjunto de entrenamiento y prueba es el mismo, es decir, toda la traza. En la siguiente sección se presentan las observaciones sobre los resultados de la tabla 5.5.

## 5.4.3 Observaciones

### 1) Participantes

- Se observa que en el modelo para la traza de tendencias, el porcentaje de error tiende a aumentar con respecto al modelo obtenido a partir de la traza numérica. Para la traza numérica el error en promedio es de 20.03%; para la traza de tendencias aumenta a 24.83%. Al agregar la aceleración, el porcentaje disminuye a 23.07% y cuando se incluye la dirección del péndulo y el carro, disminuye a 22.12%. La diferencia entre el porcentaje de la última transformación con respecto al porcentaje del modelo generado a partir de la traza sin transformar muestra un incremento de 2.09 %. Sin embargo, transformar los datos y agregar información permite que el modelo se interprete más fácilmente. Esto se ilustra en la sección 5.4.4 donde se presentan ejemplos de modelos.

- El número de reglas por participante es entre 5 y 22, por lo que los modelos son compactos, y aunque en algunos casos el número de reglas es mayor que con el modelo numérico original, la claridad parece aumentar. En promedio, para la traza numérica el número de reglas es de 13.23, para tendencias es de 9.82, para aceleración es de 15.14 y para aceleración más dirección es de 14.23.

- Cuando el participante muestra un porcentaje de error muy alto en el modelo original, el porcentaje obtenido con las trazas transformadas se incrementa mucho en comparación con el resto de los participantes. En el caso del participante 1 por

ejemplo, el error se incrementa de 30.55% a 38.36%, sin embargo, al analizar su desempeño en cuanto al número de transiciones de una acción a otra vemos que tiene el número más alto de todos, es la traza con el mayor número de ejemplos también. Esto indica que el participante generó muchas acciones en su sesión, su desempeño fue notablemente inconsistente y esto se refleja en el porcentaje de error tan alto.

- Los participantes con un porcentaje de error bajo en el modelo numérico original no presentan cambio notable al realizar las transformaciones, su porcentaje permanece prácticamente igual y en algunos casos disminuye.

- En general no se logró una disminución del porcentaje de error en los modelos. Al realizar las transformaciones de la traza de valores numéricos a cualitativos se está restando precisión ya que se pierde información. En el caso del valor para el ángulo por ejemplo, aunque los modelos nos indican la dirección del péndulo, no podemos saber con precisión en qué parte del cuadrante se encuentra. Podemos saber si se está cayendo hacia la derecha pero no sabemos si está casi en posición horizontal, si está en medio o cerca del equilibrio. Esta pérdida de información puede originar que el porcentaje de error no disminuya. Otro posible factor es el valor del umbral ya que no puede garantizarse que los obtenidos sean los que proporcionen los mejores resultados.

## 2) Experto

El experto presenta un comportamiento muy distinto al de los participantes novatos. Como se ve en la tabla, el porcentaje de error obtenido por el modelo numérico es muy bajo. Al obtener los modelos a partir de las transformaciones, el número de reglas disminuye notablemente. Esto en realidad no es tan deseable ya que se esperaría que un modelo obtenido del experto proporcionara una buena estrategia, sin embargo no es así ya que las reglas no muestran situaciones claras. Esto puede deberse a que el experto tiene un rango de acción muy pequeño, lo que significa que el péndulo siempre está en equilibrio y nunca se encuentra en situaciones críticas de las que se tenga que recuperar, por lo tanto, el modelo no proporciona ayuda en ese sentido. Puede observarse en la tabla que en comparación con los novatos, el número de transiciones entre acciones es muy bajo. Esto indica que su desempeño es sumamente consistente, aunque su porcentaje de error se incrementa con las transformaciones. Este incremento puede deberse a varios factores, uno de ellos es la característica especial de comportamiento del experto que origina que su traza no presente muchos cambios, es consistente, no hay mucho ruido por lo que el aplicar transformaciones parece introducir ruido en lugar de suavizar la traza, que por sus características, no lo necesita. Otro factor que puede influir es que, al igual que los participantes, se está perdiendo información al transformar la traza. Esto empeora el resultado puesto que por si misma, esta traza no proporciona mucha información, entonces al transformarla se pierde aun más detalle y esto se refleja en los porcentajes de error.

El porcentaje de error aumenta de 0.88% para la traza original, a 2.62% para tendencias, 2.65% para aceleración y 2.66% para aceleración más dirección. Parece que el

conocimiento adicional no influye. Es notable que el número de reglas disminuye de 19 a 2, 2, y 3 respectivamente. Esto se explica porque no existe en su traza un gran diversidad de estados y acciones que originen que se obtengan reglas para diferentes estados del dispositivo. El experto es capaz de mantener el estado del dispositivo en estados similares sin realizar muchas acciones.

### 3) Limitaciones

- Se observa una pérdida de precisión al realizar las transformaciones de la traza. Sin embargo los modelos obtenidos parecen mejorar su claridad.

- El número de reglas presenta una tendencia similar al porcentaje de error del modelo ya que no logra disminuir aunque su incremento en promedio no es grande (de 13.23 a 14.23 reglas). No se hizo un análisis estadístico para determinar qué tan significativos son los valores obtenidos.

- Falta detalle en el valor nominal de los atributos, específicamente en el comportamiento del péndulo y la posición del carro. En cuanto al péndulo no podemos saber su posición y en el caso del carro sería más claro si se indicara si va lento o rápido.

## 5.4.4 Ejemplos de modelos

A manera de ejemplo se muestran los modelos del participante 20 obtenidos a partir de la traza original, agregando tendencias y aceleración más dirección. Se eligió a este participante porque en él se observa una disminución en el porcentaje de error para cada transformación. Para el modelo obtenido a partir de la traza numérica el porcentaje de error es de 25.76%, para tendencias es de 23.71% y para aceleración más dirección es de 22.93%.

### Modelo a partir de la traza original del participante 20

right 220 9 IF anguloPendulo >= 0.083107 anguloPendulo <= 0.243095
posicionCarro <= 0.878857
right 174 12 IF anguloPendulo >= 0.031515 velocidadCarro <= -0.741018 .
right 169 27 IF anguloPendulo >= 0.052427 posicionCarro >= 0.4915 posicionCarro <= 1.18322
velocidadAngular >= -0.749538 .
right 217 7 IF anguloPendulo >= 0.051728 anguloPendulo <= 0.294672
posicionCarro <= 1.9886 .
right 375 52 IF anguloPendulo >= -0.02515 velocidadAngular >= 0.229292
posicionCarro <= 1.87932 .
right 207 34 IF anguloPendulo >= 0.052427 velocidadCarro <= 0.002189 .
right 41 0 IF anguloPendulo >= 0.051728 velocidadCarro >= 3.69283 .
left 386 38 IF anguloPendulo <= -0.053704 anguloPendulo >= -0.150085 .
left 148 17 IF anguloPendulo <= 0.030009 velocidadCarro >= 0.900853

```
                posicionCarro >= 1.09078 .
left 45 3       IF anguloPendulo <= 0.030009 velocidadAngular <= -0.175595
                posicionCarro >= -0.534309 velocidadCarro >= -0.713509
                velocidadCarro <= 0.028256
                velocidadAngular <= -0.238535 .
left 158 21     IF anguloPendulo <= 0.043965 anguloPendulo >= -0.044215
                posicionCarro <= -0.593793 velocidadAngular <= 0.228334 .
left 35 0       IF anguloPendulo <= -0.00705 posicionCarro <= 0.541057
                velocidadAngular <= 0.781878 velocidadCarro <= -1.84181
                posicionCarro <= -0.918624 .
inertia 63 31   IF anguloPendulo >= -0.142194 anguloPendulo <= 0.124909
                velocidadCarro <= 0.700513 posicionCarro >= -0.211109
                velocidadAngular <= 0.43228 .
inertia 75 46   IF anguloPendulo >= -0.150044 anguloPendulo <= 0.124909
                posicionCarro <= 0.440796 velocidadCarro >= -0.706658 .
inertia 23 5    IF posicionCarro >= 2.07998 anguloPendulo >= 0.014736
                velocidadCarro >= -0.098696 .
inertia 64 103 IF .
```

Como puede observarse, este modelo es difícil de interpretar. Se necesita conocer los posibles valores de los atributos y su significado, por ejemplo, la primera regla:

```
 right 220 9    IF anguloPendulo >= 0.083107 anguloPendulo <= 0.243095
                posicionCarro <= 0.878857
```

se interpretaría de la siguiente manera:

“Si el péndulo se está cayendo a la derecha y el carro está cerca del centro o saliéndose por la izquierda, empujar a la derecha”

## Modelo a partir de la traza con tendencias del participante 20

```
right 414 128 IF velocidadCarro = aumenta .
right 50 20     IF velocidadCarro = noCambia velocidadAngular = aumenta posicionCarro =
                noCambia anguloPendulo = noCambia .
right 11 0      IF anguloPendulo = aumenta velocidadAngular = noCambia .
right 21 4      IF velocidadCarro = noCambia velocidadAngular = aumenta posicionCarro =
                disminuye .
left 356 67     IF velocidadCarro = disminuye anguloPendulo = noCambia .
left 97 24      IF velocidadAngular = disminuye posicionCarro = noCambia .
left 53 17      IF anguloPendulo = disminuye .
left 7 4        IF velocidadAngular = noCambia posicionCarro = aumenta .
left 8 3        IF anguloPendulo = noCambia velocidadAngular = disminuye posicionCarro =
                aumenta .
inertia 124 129 IF .
```

Este modelo no resulta muy claro ya que no se puede determinar el estado del carro ni del péndulo, por ejemplo en el caso de la regla:

right 50 20 IF velocidadCarro = noCambia velocidadAngular = aumenta posicionCarro = noCambia

no podemos saber en qué parte del riel se encuentra el carro, si a la izquierda o a la derecha del centro. En el caso del péndulo ocurre lo mismo, por ejemplo, en la regla

left 53 17 IF anguloPendulo = disminuye .

no se puede saber hacia dónde se mueve el péndulo, por esta razón, es necesario agregar más conocimiento, como vemos en el modelo siguiente.

## Modelo a partir de la traza con aceleración y dirección del participante 20

```
right 217 14  IF anguloPendulo  = cayendoDerecha acelAngular = constante .
right 98 18   IF velocidadCarro = aumenta anguloPendulo = cayendoDerecha .
right 73 23   IF velocidadCarro = aumenta acelAngular = constante .
right 34 20   IF velocidadCarro = aumenta posicionCarro = alejaIzq .
right 9 4     IF velocidadCarro = aumenta acelCarro = constante acelAngular = aumenta .
right 30 8    IF velocidadCarro = noCambia anguloPendulo = cayendoDerecha
              acelCarro = disminuye .
right 6 1     IF velocidadCarro = aumenta posicionCarro = acercaDer .
right 26 11   IF anguloPendulo = equilibrio posicionCarro = alejaIzq
                velocidadAngular = noCambia
right 7 0     IF acelAngular = disminuye anguloPendulo = cayendoDerecha
                acelCarro = constante .
left 306 35   IF anguloPendulo = cayendoIzquierda .
left 129 30   IF velocidadCarro = disminuye acelAngular = constante .
left 26 12    IF posicionCarro = alejaDer velocidadCarro = disminuye
                anguloPendulo = equilibrio .
left 21 9     IF posicionCarro = alejaDer velocidadAngular = noCambia .
left 14 6     IF posicionCarro = alejaDer anguloPendulo = equilibrio acelCarro = constante
              acelAngular = disminuye .
inertia 187 161 IF .
```

En este modelo se agrega la aceleración angular, la aceleración del carro, además la dirección del péndulo. De esta forma se sabe si se está cayendo o levantando y hacia la derecha o la izquierda. En el caso del carro, se indica si se encuentra a la izquierda o derecha del centro y si se está acercando o alejando del mismo. Esto hace más fácil su interpretación.

Veamos ahora el modelo del experto. Como se mencionó anteriormente, el número de reglas disminuye demasiado y no muestra con detalle lo que hace el experto.

### Modelo del experto para tendencias

right 134 1 IF velocidadCarro=aumenta.
left 117 7 IF velocidadCarro=disminuye.
inertia 4033/679 IF.

### Modelo del experto para aceleración

right 134 1 IF velocidadCarro=aumenta.
left 117 7 IF velocidadCarro=disminuye.
inertia 4031/679 IF.

### Modelo del experto para aceleración más dirección

right 195 6 IF velocidadAngular = disminuye acelCarro = aumenta .
right 240 30 IF velocidadAngular = disminuye velocidadCarro = aumenta .
left 425 40 IF velocidadAngular = aumenta .
inertia 4010 57 IF .

Como se comentó en la sección 5.4.3, los modelos del experto no proporcionan información significativa sobre su desempeño al controlar el péndulo. El número de reglas se redujo demasiado con respecto al modelo que se obtuvo con su traza original. Su número se reduce de 19 reglas para el modelo numérico a 3 reglas para el modelo de aceleración/dirección. En las transformaciones se pierde información y dadas las características especiales que presenta esta traza, se refleja negativamente en los porcentajes de error de los modelos obtenidos de trazas transformadas.

## 5.5 Inducción de modelos con PROGOL

Para construir los modelos, PROGOL requiere como entrada un conjunto de ejemplos, conocimiento del dominio y definiciones de modo. Estos tres elementos deben ser expresados en forma de programa lógico, en este caso Prolog.

### Definiciones de modo

Las definiciones de modo permiten especificar a PROGOL el predicado que se desea aprender. Constan de una definición para la cabeza del predicado y las definiciones necesarias para las literales. Permiten especificar las variables de entrada y salida, asignar tipos y constantes entre otras características. Ver apéndice III para mayor detalle.

### Proceso para inducir los modelos

El proceso seguido fue el siguiente:

1) Definición de los predicados a aprender para tendencias y aceleración.

2) Plantear el conocimiento del dominio en Prolog

3) Transformar la traza original al formato requerido por la definición del paso 1.

4) Correr el algoritmo

5) Probar con un conjunto de ejemplos de prueba la exactitud del modelo obtenido. La exactitud es el porcentaje de ejemplos que el modelo clasifica correctamente.

El conjunto de ejemplos se dividió en dos grupos. Se tomó la traza de un participante aleatoriamente y se hicieron pruebas con 100, 300 y 500 ejemplos de entrenamiento. Posteriormente el modelo se probó con la traza completa y el modelo inducido con 300 ejemplos fue el que obtuvo menor porcentaje de error (1.19%) en comparación con los obtenidos para 100 (6.36%) y 500 (30.84%). Por lo tanto, para la inducción del modelo se utilizó un conjunto de 300 ejemplos de entrenamiento y como ejemplos de prueba se usó la traza completa. Cada traza contiene de 3600 a 1613 ejemplos aproximadamente.

Se obtuvieron los resultados para catorce participantes y el experto. No se indujeron los modelos para los veintidós participantes debido a que el tiempo que PROGOL utiliza para generarlos aumenta considerablemente conforme se va agregando conocimiento del dominio. Sin embargo esta muestra es útil para el objetivo de este trabajo.

## 5.5.1 Predicados definidos y determinación del umbral

Se definieron los predicados a aprender y se expresaron utilizando las definiciones de modo. Son dos predicados, uno para tendencias y otro para aceleración más dirección. A continuación se muestran.

Para tendencias:

```
:- modeh(1,ej(dato1(+ap1,+a1,+xp1,+x1),dato2(+ap2,+a2,+xp2,+x2),#ej))?
:- modeb(1,tendApunto(+ap1,+ap2,#ten))?
:- modeb(1,tendA(+a1,+a2,#ten))?
:- modeb(1,tendXpunto(+xp1,+xp2,#ten))?
:- modeb(1,tendX(+x1,+x2,#ten))?
```

Este predicado utiliza de entrada dos estados y su cuerpo consiste en las tendencias de cada valor de atributo.

Para aceleración y dirección:

```
:-
modeh(1,ej(dato1(+ap1,+a1,+xp1,+x1),dato2(+ap2,+a2,+xp2,+x2),dato3(+ap3,+a3,+xp3,+x3),#ej))
?
:- modeb(1,pendulo(+a2,+a3,#dirP))?
:- modeb(1,posicionCarro(+x2,+x3,#posiCarro))?
:- modeb(1,aceAngulo(+ap1,+ap2,+ap3,#ace))?
:- modeb(1,aceCarro(+xp1,+xp2,+xp3,#ace))?
```

Este predicado utiliza tres estados, el cuerpo está formado por la dirección, la posición del carro y las aceleraciones. No se incluyó velocidad como se hizo en RIPPER ya que al probarlo de ese modo, se observó que la exactitud del modelo disminuye.

En las pruebas iniciales se utilizaron los mismos valores de umbral que para RIPPER pero los resultados no fueron buenos. El porcentaje de error fue alto. Se observó que PROGOL redondea los valores al realizar el proceso de inducción. Esto influye en que el mismo valor de umbral no sea útil. Mientras que en RIPPER se utilizó un mismo umbral para todos los valores de atributo en PROGOL no fue así. Se realizaron pruebas con el participante tres y se seleccionó el umbral que generara un modelo con el menor número de cláusulas sin generalizar, de esta forma se determinaron los siguientes valores: para las tendencias de todos los valores de atributo se utilizó un umbral de 0.01, para la aceleración angular y del carro el umbral es de 0.11 y para la posición del carro el umbral es 0.1.

### 5.5.2 Resultados

Una vez planteadas las definiciones de modo, conocimiento del dominio, formato de ejemplos y umbral, se indujeron los modelos para 14 participantes. Las definiciones pueden verse en el Apéndice IV.

Se obtuvieron modelos para tendencias y aceleración más dirección, los resultados del porcentaje de error y el número de reglas se muestran en la siguiente tabla:

| | | | | | Tendencias | | Ace/Dirección | |
|---|---|---|---|---|---|---|---|---|
| P | Cambios | right | left | inertia | % Error | # Reglas | % Error | # Reglas |
| 1 | 1500 | 894 | 897 | 1924 | 3.68 | 15 (10) | 1.00 | 19 (0) |
| 2 | 505 | 591 | 452 | 498 | 10.06 | 13 (11) | 2.21 | 21 (0) |
| 3 | 509 | 695 | 661 | 354 | 18.36 | 11 (23) | 1.87 | 17 (0) |
| 4 | 414 | 1027 | 819 | 234 | 8.56 | 11 (17) | 5.68 | 16 (1) |
| 5 | 427 | 572 | 639 | 391 | 13.55 | 9 (25) | 0.69 | 22 (0) |
| 6 | 348 | 416 | 534 | 376 | 6.49 | 14 (14) | 1.28 | 17 (0) |
| 7 | 440 | 949 | 896 | 212 | 13.76 | 12 (13 ) | 2.19 | 9 (1) |
| 8 | 461 | 360 | 482 | 545 | 5.05 | 12 (16 ) | 15.37 | 10 (1) |
| 9 | 387 | 570 | 660 | 312 | 23.28 | 13 (18 ) | 8.57 | 19 (1) |
| 10 | 444 | 453 | 571 | 398 | 7.24 | 10 (35 ) | 1.13 | 16 (0) |
| 11 | 453 | 892 | 790 | 282 | 3.82 | 10 (12 ) | 11.21 | 16 (1) |
| 12 | 499 | 729 | 550 | 318 | 10.9 | 10 (25 ) | 12.28 | 17 (0) |
| 14 | 515 | 687 | 613 | 381 | 11.15 | 14 ( 12) | 4.88 | 12 (0) |
| 16 | 510 | 717 | 515 | 381 | 11.16 | 13 (13 ) | 9.06 | 17 (0) |
| | | | | | | | | |
| Prom. | | | | | 10.5 | 11.93 | 5.53 | 16.29 |
| D.S. | | | | | 5.53 | 1.82 | 4.91 | 3.75 |
| | | | | | | | | |
| Exper | 61 | 224 | 215 | 1897 | 1.19 | 11 (0) | 0 | 18(0) |

Tabla 5.6 Resultados de los modelos inducidos con PROGOL

Para la columna de reglas, el número entre paréntesis indica el número de cláusulas que no fueron generalizadas. Cuando PROGOL trata de encontrar una generalización de la cláusula aterrizada (ejemplo) y no tiene éxito, lo incluye en el modelo resultante. Lo que se espera es que el modelo no tenga cláusulas aterrizadas.

## 5.5.3 Observaciones

### 1) Participantes

- Los modelos de tendencias tienen cláusulas que no pudieron ser generalizadas, esto se explica porque los participantes son novatos y no presentan consistencia en su desempeño, su traza tiene mucho ruido y el conocimiento del dominio que se agrega no es suficiente para que el modelo mejore su exactitud.

- Para las pruebas de aceleración más dirección no se incluye la velocidad. Se realizó una prueba previa para el participante 3 y se obtuvo un porcentaje de error alto (36.43%), aunque no puede afirmarse que el agregar velocidad reste exactitud debido a que no se hicieron pruebas exhaustivas con valores de umbral distintos para este caso.

- Al agregar más conocimiento del dominio, en este caso la aceleración más la dirección, el número de cláusulas sin generalizar prácticamente desapareció. El porcentaje de error obtenido también disminuye. De 10.50% que obtiene en tendencias se reduce a 5.53% en promedio para aceleración más dirección.

- En cuanto al número de reglas se observa un incremento que en promedio sube de 11.93 para tendencias a 16.29 para aceleración más dirección.

## 2) Experto

A diferencia de los participantes, el modelo del experto nunca presenta cláusulas sin generalizar. Su porcentaje de error es el más bajo; en el modelo para aceleración más dirección, su porcentaje de error es de 0%. Como se ha visto a lo largo de las pruebas, este modelo es el que obtiene menor porcentaje de error con respecto a los participantes.

## 3) Generales

- Conforme se va agregando conocimiento del dominio y el predicado a aprender incluye más variables, el tiempo que tarda PROGOL en obtener los modelos se incrementa considerablemente. La ventaja es que el modelo presenta mayor exactitud. Al obtener el modelo para tendencias y para aceleración más dirección es posible observar si el incorporar más conocimiento del dominio mejora o no el modelo.

- Se observa que la incorporación del conocimiento del dominio directamente en la definición, sin necesidad de pre-procesar la traza mejora los resultados.

- Los modelos obtenidos permiten ver las relaciones entre las variables. Para el caso de las tendencias, el predicado tiene como entrada dos estados con sus respectivas variables : velocidad angular, ángulo, velocidad del carro y posición. En el cuerpo del predicado se observa la forma en que estas variables se relacionan y su efecto. Por ejemplo:

```
ej(dato1(A,B,C,D),dato2(E,F,G,H),right) :- tendApunto(A,E,aumenta),
                                            tendA(B,F,aumenta),
                                            tendXpunto(C,G,disminuye).
```

Se observa las tendencias de las variables y la acción efectuada. A y E representan la velocidad angular, B y F representan el ángulo y C y G la velocidad del carro.

Esta característica resulta favorable si las relaciones que se obtienen tienen un significado en el modelo. En este ejemplo, cada variable se relaciona con otra del

mismo tipo en el segundo estado por lo que no existe confusión pero esto no siempre es así. Cuando en el modelo aparecen relaciones de variables que no tienen sentido o al menos no lo tienen para nosotros, es necesario definir en PROGOL otras restricciones de búsqueda.

## 5.5.4 Explicación de los modelos

A continuación se muestran y se explican algunos predicados de los modelos obtenidos. Los modelos del experto y del participante 5 pueden verse en el apéndice IV.

### Ejemplos de predicados de un modelo de tendencias:

**1) ej(dato1(A,B,C,D),dato2(E,F,G,H),*inertia*) :- tendXpunto(C,G,noCambia).**

El predicado consta de 3 argumentos, los dos primeros, representan dos estados: dato1(A,B,C,D) es un estado que ocurre antes que dato2(E,F,G,H). El tercer argumento es la acción que el participante realiza y puede tomar los valores *right, left* e *inertia*.

A,B,C,D y E,F,G,H corresponden a las cuatro variables de estado del dispositivo donde
A ,E= Velocidad angular
B,F = Angulo
C,G = Velocidad del carro
D,H = Posición del carro

Por lo tanto, la regla se interpretaría de la siguiente manera:
"Si la velocidad del carro no cambia no hagas nada".

**2) ej(dato1(A,B,C,D),dato2(E,F,G,H),*right*) :- tendApunto(A,E,aumenta),**
**tendA(B,F,aumenta),**
**tendXpunto(C,G,disminuye).**

"Si la velocidad angular aumenta, el ángulo se hace más grande y la velocidad del carro disminuye, entonces empuja a la derecha".

## Ejemplos de predicados de un modelo de aceleración/dirección:

La interpretación es similar a la del modelo de tendencias, sin embargo, al predicado se le ha agregado un nuevo estado y al cuerpo se le incorporó dirección del péndulo y el carro así como también la aceleración. Veamos algunos ejemplos:

1) **ej(dato1(A,B,C,D),dato2(E,F,G,H),dato3(I,J,K,L),*left*) :-**
   **pendulo(F, J,cayendoIzquierda),**
   **aceAngulo(E,E,I,constante),**
   **aceCarro(C,G,C,aumenta).**

Se interpreta como: " Si el péndulo está cayendo a la izquierda, la aceleración del ángulo del péndulo es constante y la aceleración del carro aumenta, empuja a la izquierda".

2) **ej(dato1(A,B,C,D),dato2(E,F,G,H),dato3(I,J,K,L),*right*) :-**
   **pendulo(F,J,cayendoDerecha),**
   **posicionCarro(H,L,noCambia),**
   **aceCarro(C,G,K,disminuye).**

Su interpretación es: "Si el péndulo está cayendo a la derecha, la posición del carro no cambia y la aceleración del carro disminuye, entonces empuja a la derecha"

## 5.6 Resumen de resultados: RIPPER / PROGOL

Como hemos visto, los resultados obtenidos en cada sistema de aprendizaje y los modelos generados por cada uno de ellos presentan diferentes características que a continuación se resumen.

Los siguientes puntos no tienen como fin determinar si un sistema es mejor que otro ya que las características (número de ejemplos, valor de umbral, etc.) de las pruebas para cada sistema son diferentes.

- RIPPER construye los modelos más rápido que PROGOL. Conforme se va incrementando el conocimiento del dominio y el predicado que se quiere aprender se hace más complejo, el tiempo para obtener el modelo aumenta. Sin embargo, para RIPPER, el hecho de agregar dos atributos más a la traza no le resta mucha rapidez.

- En PROGOL el conocimiento del dominio se representa en forma de programa en Prolog por lo que se integra en la misma definición. RIPPER no permite incorporar conocimiento del dominio. Se necesita pre-procesar la traza para transformarla, integrando en esta transformación la relación que se desea aprender.

- Las definiciones de modo con PROGOL hacen necesario que el conjunto de ejemplos de entrenamiento y prueba sean transformados al formato que tiene el predicado que se quiere aprender. La entrada de RIPPER es una tabla donde las columnas son los atributos y la clase y los renglones son los ejemplos.

- Los modelos obtenidos con PROGOL presentan un porcentaje de error más bajo que los generados por RIPPER.

- El umbral utilizado en RIPPER no fue bueno para PROGOL por lo que se definieron diferentes valores.

- Los modelos obtenidos con PROGOL permiten ver las relaciones de las variables que intervienen en el modelo. Esto es favorable cuando las relaciones mostradas tienen significado para quien lee el modelo. Ver esas relaciones permite una mejor comprensión del mismo. Por el contrario, si se desconocen esas relaciones, puede causar confusión.

- En las pruebas con RIPPER no se logra disminuir el porcentaje de error. Para la traza numérica el error en promedio es de 20.03%; para la traza de tendencias aumenta a 24.83%. Al agregar la aceleración, el porcentaje disminuye a 23.07% y cuando se incluye la dirección del péndulo y el carro, disminuye a 22.12%. La diferencia entre el porcentaje de la última transformación con respecto al porcentaje del modelo generado a partir de la traza sin transformar muestra un incremento de 2.09 %. Sin embargo, transformar los datos y agregar información permite que los modelos se interpreten más fácilmente. El número de reglas presenta un

comportamiento similar ya que aumenta de 13.23 reglas en promedio a 14.23 para el modelo con dirección más aceleración.

- En las pruebas con PROGOL se obtiene un porcentaje de error bajo. Para tendencias, se tiene en promedio un porcentaje de 10.5% que disminuye a 5.53% al agregar aceleración más dirección. El número de reglas para los modelos aumenta de 11.93 reglas en promedio para tendencias a 16.29 reglas para aceleración.

- En el trabajo previo [Morales, R., 2000], los modelos verbalizados eran largos y repetitivos sin obtenerse el porcentaje de error de los mismos. En este trabajo se obtienen modelos verbalizados con un porcentaje de error similar al obtenido por los modelos numéricos para el caso de RIPPER. Se observan mejores resultados con el enfoque de programación lógica inductiva (PROGOL) en el que se obtienen modelos con un porcentaje de error muy bajo como se detalla en el punto anterior.

Capítulo 6

# Conclusiones y trabajo futuro

En este trabajo se obtuvieron modelos de habilidades humanas de novatos que aprenden a controlar un péndulo invertido. Esta tarea de control implica variables continuas, por lo que los modelos obtenidos a partir de las trazas originales resultan difíciles de interpretar. Para solucionar este problema, las trazas se pre-procesaron, transformando los valores cuantitativos a cualitativos a fin de obtener reglas con valores nominales que resultaran más fáciles de entender. Se incorporó conocimiento del dominio; en este caso, se obtuvieron trazas sobre las tendencias, la aceleración y la dirección. Estas fueron la entrada al sistema de aprendizaje de reglas RIPPER, obteniéndose los resultados concentrados en la tabla 5.4. Posteriormente se definieron en Prolog dos predicados a aprender con base en tendencias y aceleración respectivamente, se transformaron los ejemplos originales al formato de esos predicados y se definió el conocimiento del dominio. Esto fue la entrada al sistema de programación lógica inductiva PROGOL, obteniéndose los resultados de la tabla 5.5. El proceso también se hizo con la traza de un experto a fin de comparar sus resultados con los novatos.

En base al trabajo desarrollado se obtuvieron las siguentes conclusiones:

## 6.1 Conclusiones

- Los resultados obtenidos en este trabajo sugieren que el incorporar conocimiento del dominio permite construir modelos más fáciles de interpretar y con un porcentaje de error ligeramente mayor o similar al de los modelos de las trazas originales para RIPPER y con un porcentaje de error bajo para PROGOL.

- En la fase de pre-procesamiento resulta importante definir un umbral adecuado para la transformación de las trazas. Si se utilizan valores muy pequeños o muy grandes, el porcentaje de error del modelo se incrementa y las reglas no son muy útiles. El umbral utilizado se obtuvo a partir de un participante promedio, sin embargo esto no significa que sea el mejor para todos. Se observó que el umbral utilizado en RIPPER no resultó bueno para PROGOL. Este último sistema redondea los valores numéricos, esto puede ser un factor que origine que un valor de umbral no tenga el mismo efecto para ambos.

- El modelo del experto no resulta descriptivo de una buena estrategia a seguir ya que nunca se encuentra en una situación crítica para superar. Su rango de acciones es muy reducido por lo que al transformar su traza y perder precisión su porcentaje de error incrementa.

- El uso de programación lógica inductiva permite incorporar conocimiento del dominio directamente. Se observó que al agregar más conocimiento y hacer más complejo el predicado que se quiere aprender, el tiempo para obtener los modelos se incrementa, pero el modelo generado presenta mayor exactitud.

- En este trabajo se logra obtener modelos verbalizados con un porcentaje de error similar o mayor al obtenido por los modelos numéricos para el caso de RIPPER (de 13.23% para valores numéricos disminuye a 9.82% para tendencias, incrementa a 15.14% para aceleración y disminuye a 14.23% para aceleración más dirección en promedio) . Se observan mejores resultados con el enfoque de programación lógica inductiva usando PROGOL en el que se obtienen modelos con un porcentaje de error muy bajo (de 10.5% para tendencias disminuye a 5.53% para aceleración más dirección en promedio). Se observa de este modo que es posible obtener modelos nominales con buen poder predictivo a partir de trazas numéricas.

## 6.2 Trabajo futuro

Los resultados obtenidos en este trabajo pueden mejorarse, a continuación se mencionan algunas posibles extensiones:

- Determinar una forma de obtener el umbral para la transformación de manera que garantice que el error será el mínimo. Se pueden utilizar puntos de inflexión y podría probarse un mismo umbral para todos los atributos o un umbral diferente por cada uno.

- Realizar el mapeo de los valores numéricos de las trazas a valores cualitativos utilizando otras técnicas, como por ejemplo lógica difusa.

- Mejorar la calidad descriptiva del modelo. Agregar otros valores nominales que describan mejor el estado del dispositivo. Es posible que esto permita reducir el porcentaje de error para RIPPER en combinación con los valores de umbral.

- Mejorar el planteamiento del dominio para PROGOL para restringir la búsqueda. En ocasiones, se obtienen relaciones de variables que no son relevantes. Esto se puede hacer definiendo restricciones de búsqueda en el archivo de entrada. Al restringir el espacio de búsqueda, el tiempo que PROGOL consume en obtener el modelo se reduce también.

- Evaluación del modelo como controlador. Un modelo puede tener diversos propósitos; este trabajo se enfocó al uso del modelo como herramienta descriptiva para entender el desempeño de una persona al aprender a controlar un dispositivo y se evaluó en base al porcentaje de error con ejemplos de prueba. Es conveniente

probar los modelos obtenidos en el simulador del que se tomaron las trazas para evaluar su comportamiento. [Morales, R., 2000] realizó un trabajo al respecto.

- Integración de la herramienta de pre-procesamiento con los sistemas de aprendizaje con los que se esté trabajando, no únicamente con los utilizados en esta tesis. Esto sería muy útil pues el proceso completo por traza se haría en un paso y facilitaría la labor de trabajar con otros sistemas. Actualmente se tiene que hacer por etapas, lo que hace que el proceso sea lento especialmente para programación lógica inductiva.

# Referencias


[Bratko, 1991] Bratko, I., Muggleton, S., y Varsek, A. Learning qualitative models of dynamic systems. En Proceedings of the Eighth International Machine Learning Workshop, San Mateo, Ca. Morgan-Kaufmann.

[Camacho,1995] Camacho, R. Using Machine Learning to extract Models of Human Control Skills in the Second International Workshop on Artificial Intelligence Techniques (AIT-95), Brno, Chec Republic.

[Camacho,1998] Camacho, R. Inducing Models of Human Control Skills. En 10th European Conference on Machine Learning (ECML-98) , Chemnitz, Germany.

[Carbonell,1983] Carbonell, J.G., Michalski, R.G., y Mitchell, T.M. An Overview of Machine Learning. En Michalski, R.S., Carbonell, J.G. y Mitchell, T.M. (Eds.), Machine Learning: An Artificial Intelligence Approach. Morgan Kaufmann Publishers, Inc.

[Cohen,1996] Cohen, W. y Yoram Singer. Context-sensitive Learning Methods for Text Categorization. En Proceedings of SIGIR-96, 19th ACM International Conference on Research and Development in Information Retrieval.

[Cohen,1995] Cohen, W. Fast Effective Rule Induction. En Proceedings of the Twelfth International Conference.

[Hume,1991] Hume, D. y Sammut, C. Applying Inductive Logic Programming in Reactive Environments. En Proceedings of the 1st International Workshop on Inductive Logic Programming.

[Michie,1992] Michie, D., Camacho, R. Building Symbolic Representations of Intuitive Real-Time Skills from Performance Data. En the Machine Intelligence 13 Workshop, Glasgow, U.K.

[Michie,1995] Michie, D. y Sammut, C. Behavioural Clones and Cognitive Skill Models. En K. Furukawa, Machine Intelligence 14. Oxford University Press.

[Morales,E.,2001] Morales, E. Notas del curso Descubrimiento de conocimiento en bases de datos. ITESM, Campus Cuernavaca.

[Morales,R.,2000] Morales, R. Exploring Participative Learner Modelling and its Effects on Learner Behaviour. Ph.D. Thesis University of Edinburgh.

[Morales,R.,2001] Morales, R., Pain, H., Conlon, T. Effects of Inspecting Learner Models on Learners' Abilities. Proc. AIED 2001, pp. 434-445. 13.

[Muggleton,1995] Muggleton, Stephen. Inverting Entailment and Progol.Machine Intelligence 14: Applied Machine Intelligence.

[Muggleton,1991] Muggleton, S. Inductive Logic Programming. New Generation Computing, 8(4):295-318.

[Muggleton,1994] Muggleton, S. y De Raedt. Inductive Logic Programming: Theory and Methods, Journal of Logic Programming:19,20:629-679.

[Muggleton,2001] Muggleton, S. y J. Firth. CProgol4.4: A Tutorial Introduction. En S. Dzeroski y N. Lavrac, editors, Relational Data Mining, pp. 160-188. Springer-Verlag.

[Sammut,1992] Sammut, C., Hurst, S., Kedzier, D. & Michie, D. Learning to Fly. En Proceedings of the Ninth International Conference on Machine Learning, Aberdeen: Morgan Kaufmann, pp. 385-393.

[Pagallo, 1990] Pagallo y Haussler. Boolean Feature Discovery in Empirical Learning. Machine Learning 5(1).

[Quinlan,1986] J. R. Quinlan. Induction of Decision Trees. Machine Learning, vol. 1, pp. 81-106.

[Thornton,1992] Thornton, C.J. Techniques in Computational Learning. An Introduction. Chapman & Hall.

[Wenger,1987] Wenger, E. Artificial intelligence and Tutoring Systems. En Computational approaches to the communication of knowledge. Los Altos, Morgan Kaufmann.

[Wieland, 1991] Wieland , A. P. Evolving Neural Network Controllers for Unstable Systems . En Proceedings of the International Joint Conference on Neural Networks, pp. 2,667-2,673.

Apéndice I

# Herramienta para pre-procesamiento

Para la fase de pre-procesamiento se desarrolló una herramienta que realiza las transformaciones en las trazas. El lenguaje utilizado es Java (jdk1.3.1).

El programa acepta como entrada una traza de valores continuos y la transforma en:

1) Tendencias. Transforma la traza numérica en un archivo con valores nominales que incorpora las tendencias de los valores de los atributos.
2) Aceleración. Transforma la traza numérica en un archivo con valores nominales que incorpora las aceleración angular y la aceleración del carro.
3) Dirección. Transforma la traza numérica en un archivo con valores nominales que incorpora las dirección del péndulo y del carro.
4) Transforma la traza original en cláusulas de Prolog en el formato requerido para aprender el predicado para dos estados. (PROGOL).
5) Transforma la traza original en cláusulas de Prolog en el formato requerido para aprender el predicado para tres estados. (PROGOL).
6) Permite variar los umbrales para tendencias y aceleración.
7) Obtiene el número de cambios de acción que existen en la traza, por ejemplo, *right-left*, *left-inertia*, *inertia-right*, etc. Esto es útil para analizar el comportamiento del participante.

La figura I-1 muestra la interfaz del programa:

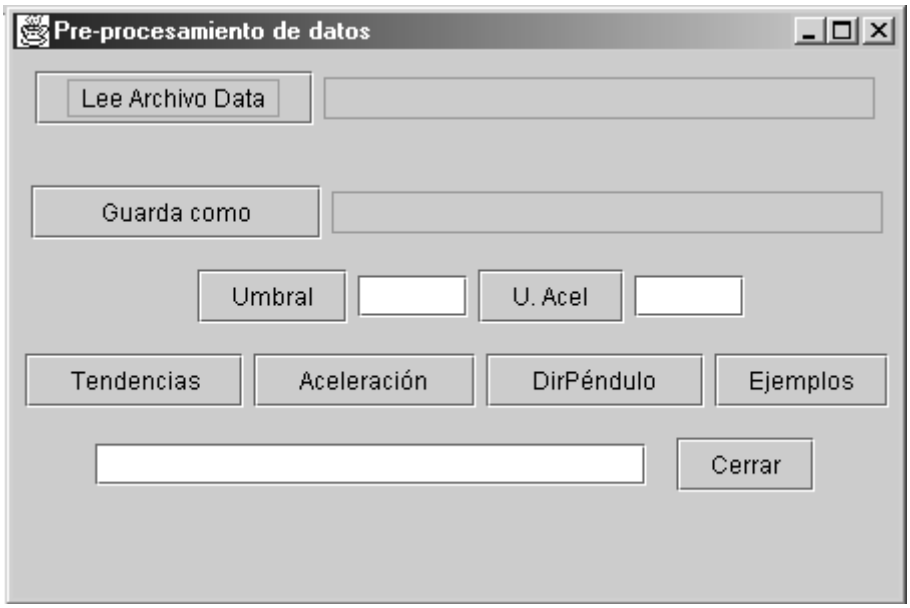



Figura I-1 Interfaz del programa para pre-procesamiento de trazas

Apéndice II

# Tabla de resultados : RIPPER (-aunordered)

La tabla II-1 muestra los resultados de los modelos utilizando la opción –aunordered.

| | | | | | Numéricos | | Tendencias | | Aceleración | | Ace/Dirección | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| P | Cambios | *right* | *left* | *inertia* | % Error | # Reglas | % Error | # Reglas | % Error | # Reglas | % Error | # Reglas |
| 1 | 1500 | 894 | 897 | 1924 | 28.89 | 16 | 47.01 | 3 | 43.41 | 11 | 38.36 | 9 |
| 2 | 505 | 591 | 452 | 498 | 20.82 | 9 | 28.36 | 12 | 28.14 | 16 | 20.34 | 18 |
| 3 | 509 | 695 | 661 | 354 | 18.35 | 11 | 17.37 | 17 | 16.92 | 25 | 16.33 | 25 |
| 4 | 414 | 1027 | 819 | 234 | 10.33 | 22 | 9.62 | 16 | 8.57 | 28 | 8.61 | 34 |
| 5 | 427 | 572 | 639 | 391 | 21.52 | 16 | 22.35 | 19 | 19.94 | 27 | 20.75 | 24 |
| 6 | 348 | 416 | 534 | 376 | 21.78 | 17 | 33.33 | 15 | 33.31 | 13 | 31.12 | 17 |
| 7 | 440 | 949 | 896 | 212 | 11.96 | 25 | 11.14 | 18 | 10.18 | 25 | 10.18 | 31 |
| 8 | 461 | 360 | 482 | 545 | 19.09 | 16 | 23.94 | 16 | 20.22 | 25 | 20.65 | 27 |
| 9 | 387 | 570 | 660 | 312 | 20.93 | 11 | 27.69 | 13 | 26.56 | 17 | 25 | 14 |
| 10 | 444 | 453 | 571 | 398 | 22.35 | 16 | 32.98 | 11 | 31.76 | 16 | 31.69 | 16 |
| 11 | 453 | 892 | 790 | 282 | 14.30 | 15 | 15.12 | 16 | 13.61 | 28 | 12.95 | 37 |
| 12 | 499 | 729 | 550 | 318 | 23.09 | 18 | 33.38 | 11 | 30.91 | 20 | 31.29 | 17 |
| 14 | 515 | 687 | 613 | 14 | 21.28 | 12 | 22.07 | 17 | 19.95 | 27 | 18.11 | 27 |
| 16 | 510 | 717 | 515 | 381 | 22.06 | 13 | 28.83 | 15 | 27.44 | 19 | 26.44 | 25 |
| 17 | 546 | 900 | 722 | 252 | 19.68 | 21 | 19.74 | 18 | 17.74 | 26 | 16.93 | 29 |
| 18 | 587 | 649 | 560 | 404 | 19.45 | 19 | 27.28 | 17 | 24.58 | 24 | 21.23 | 21 |
| 19 | 542 | 677 | 836 | 305 | 20.56 | 16 | 20.19 | 16 | 17.84 | 28 | 17.84 | 29 |
| 20 | 395 | 611 | 606 | 320 | 18.34 | 15 | 25.89 | 11 | 23.58 | 23 | 21.95 | 25 |
| 21 | 468 | 802 | 768 | 283 | 16.24 | 17 | 13.87 | 18 | 13.29 | 29 | 14.64 | 27 |
| 25 | 348 | 560 | 550 | 233 | 21.13 | 15 | 28.74 | 14 | 27.74 | 21 | 27.74 | 18 |
| 27 | 426 | 499 | 496 | 320 | 31.79 | 16 | 32.17 | 8 | 31.15 | 15 | 34.35 | 13 |
| 29 | 490 | 796 | 777 | 282 | 14.49 | 22 | 15.63 | 21 | 15.65 | 24 | 16.08 | 26 |
| | | | | | | | | | | | | |
| Prom. | | | | | 19.93 | 16.27 | 24.39 | 14.64 | 22.84 | 22.14 | 21.94 | 23.14 |
| D.S. | | | | | 4.83 | 3.89 | 8.86 | 4.07 | 8.56 | 5.44 | 7.97 | 7.15 |
| | | | | | | | | | | | | |
| Exper | 61 | 224 | 215 | 1897 | 1.01 | 35 | 2.62 | 2 | 2.62 | 3 | 2.60 | 5 |

Tabla II-1. Resultados de los modelos inducidos por RIPPER con -aunordered

Los resultados en cuanto a porcentaje de error son similares a los obtenidos con el parámetro –agiven.

Apéndice III

# Definiciones de modo

Los modos describen los predicados que serán usado para especificar la cabeza (modeh) o el cuerpo (modeb) de las cláusulas que formarán la hipótesis.

Una declaración de modo puede ser de la forma modeh(n,atomo) ó modeb(n,atomo), donde :

1) n (*recall*), es un entero mayor que cero ó * . Se usa para limitar el número de alternativas para instanciar el átomo. Cuando es * indica que se desea obtener todas las soluciones.
2) atomo es un atomo aterrizado.
3) Los términos en el atomo pueden ser normales ó marcadores.
    - Un término normal puede ser una constante o símbolo de función seguido de una tupla de términos escritos entre corchetes.
    - Un marcador puede ser +*type*, -*type* o # donde *type* es una constante

A continuación se muestra un ejemplo de definición de cabeza del predicado:

modeh(1,tia_de (+persona,+persona))?

Esta declaración establece que el predicado tia_de tiene dos variables de tipo persona como argumentos. El símbolo '+' indica que el primer argumento es una variable de entrada. Un símbolo '-' indicaría que es una variable de salida, y el símbolo '#' indicaría que el argumento es una constante.

Las siguientes son definiciones del cuerpo del predicado:

:- *modeb*(*,progenitor_de(-persona,+persona))?
:- *modeb*(*,progenitor_de(+persona,-persona))?
:- *modeb*(*,hermana_de(+persona,-persona))?

Las primer declaración se usa para agregar el predicado progenitor_de al cuerpo de la hipótesis. Puede dar por resultado uno o más progenitores para un niño dado. De la misma manera, la segunda definición permite que progenitor_de sea usado en el cuerpo para encontrar a uno o más niños dado un progenitor. Por último, en el cuerpo puede introducirse el predicado hermana_de para encontrar a una o más hermanas dada una persona.

Apéndice IV

# Definiciones de entrada para PROGOL

Se hicieron dos definiciones: una para aprender tendencias y otra para aprender aceleración y dirección. En la primera se requieren dos estados, en la segunda, tres. A continuación se muestran las definiciones.

## 1) Definiciones de modo, conocimiento del dominio y ejemplos para aprender tendencias.

```
:- set(posonly)?
:- set(inflate,601)?
:- set(h,10000)?
:- modeh(1,ej(dato1(+ap1,+a1,+xp1,+x1),dato2(+ap2,+a2,+xp2,+x2),#ej))?
:- modeb(1,tendApunto(+ap1,+ap2,#ten))?
:- modeb(1,tendA(+a1,+a2,#ten))?
:- modeb(1,tendXpunto(+xp1,+xp2,#ten))?
:- modeb(1,tendX(+x1,+x2,#ten))?

% Types
ej(right).
ej(inertia).
ej(left).
ten(aumenta).
ten(disminuye).
ten(noCambia).

ap1(Y1) :- ej(dato1(Y1,_,_,_),dato2(_,_,_,_),_).
a1(Z1):-ej(dato1(_,Z1,_,_),dato2(_,_,_,_),_).
xp1(W1):-ej(dato1(_,_,W1,_),dato2(_,_,_,_),_).
x1(E1):-ej(dato1(_,_,_,E1),dato2(_,_,_,_),_).
ap2(Y2):-ej(dato1(_,_,_,_),dato2(Y2,_,_,_),_).
a2(Z2):-ej(dato1(_,_,_,_),dato2(_,Z2,_,_),_).
xp2(W2):-ej(dato1(_,_,_,_),dato2(_,_,W2,_),_).
x2(E2):-ej(dato1(_,_,_,_),dato2(_,_,_,E2),_).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%  Tendencias
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

compara(V1,V2,'noCambia') :-
        Resta is V2-V1,
        Resta < 0,
        AbsResta is Resta*(-1),
        AbsResta < 0.01.

compara(V1,V2,'noCambia') :-
        Resta is V2-V1,
```

```
        Resta > 0,
        Resta < 0.01.


compara(V1,V2,''noCambia') :-
        Resta is V2-V1,
        Resta =  0,
        Resta < 0.01.


compara(V1,V2,'aumenta') :-
        V1<V2.

compara(V1,V2,'disminuye') :-
        V1>V2.

tendApunto(A1punto,A2punto,R) :-
      compara(A1punto,A2punto,R).

tendA(A1,A2,R) :-
      compara(A1,A2,R).

tendXpunto(Xp1,Xp2,R) :-
      compara(Xp1,Xp2,R).

tendX(X1,X2,R) :-
      compara(X1,X2,R).

%%%% Se utilizaron 300 ejemplos

ej(dato1(-0.365005,-0.117936,-0.711937,-0.52007),dato2(-0.69245,-
0.125236,-0.515353,-0.534309),left).
ej(dato1(-0.69245,-0.125236,-0.515353,-0.534309),dato2(-1.11222,-
0.169227,-0.314874,-0.565123),inertia).
```

## 1) Definiciones de modo, conocimiento del dominio y ejemplos para aprender aceleración y dirección.

```
:- set(posonly)?
:- set(inflate,601)?
:- set(h,10000)?
:-
modeh(1,ej(dato1(+ap1,+a1,+xp1,+x1),dato2(+ap2,+a2,+xp2,+x2),dato3(+ap3,+
a3,+xp3,+x3),#ej))?
:- modeb(1,pendulo(+a2,+a3,#dirP))?
:- modeb(1,posicionCarro(+x2,+x3,#posiCarro))?
:- modeb(1,aceAngulo(+ap1,+ap2,+ap3,#ace))?
:- modeb(1,aceCarro(+xp1,+xp2,+xp3,#ace))?

% Types
ej(right).
ej(inertia).
ej(left).
```

```
ace(aumenta).
ace(disminuye).
ace(constante).
dirP(levantandoDerecha).
dirP(cayendoIzquierda).
dirP(levantandoIzquierda).
dirP(cayendoDerecha).
dirP(equilibrio).
posiCarro(noCambia).
posiCarro(alejaIzquierda).
posiCarro(alejaDerecha).
posiCarro(acercaDerecha).
posiCarro(acercaIzquierda).

ap1(Y1) :- ej(dato1(Y1,_,_,_),dato2(_,_,_,_),dato3(_,_,_,_),_).
a1(Z1):-ej(dato1(_,Z1,_,_),dato2(_,_,_,_),dato3(_,_,_,_),_).
xp1(W1):-ej(dato1(_,_,W1,_),dato2(_,_,_,_),dato3(_,_,_,_),_).
x1(E1):-ej(dato1(_,_,_,E1),dato2(_,_,_,_),dato3(_,_,_,_),_).
ap2(Y2):-ej(dato1(_,_,_,_),dato2(Y2,_,_,_),dato3(_,_,_,_),_).
a2(Z2):-ej(dato1(_,_,_,_),dato2(_,Z2,_,_),dato3(_,_,_,_),_).
xp2(W2):-ej(dato1(_,_,_,_),dato2(_,_,W2,_),dato3(_,_,_,_),_).
x2(E2):-ej(dato1(_,_,_,_),dato2(_,_,_,E2),dato3(_,_,_,_),_).
ap3(Y3):-ej(dato1(_,_,_,_),dato2(_,_,_,_),dato3(Y3,_,_,_),_).
a3(Z3):-ej(dato1(_,_,_,_),dato2(_,_,_,_),dato3(_,Z3,_,_),_).
xp3(W3):-ej(dato1(_,_,_,_),dato2(_,_,_,_),dato3(_,_,W3_,_),_).
x3(E3):-ej(dato1(_,_,_,_),dato2(_,_,_,_),dato3(_,_,_,E3),_).



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Aceleracion Angulo
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

obtenAcel(V1,V2,'constante') :-
        Resta is V2-V1,
        Resta < 0,
        AbsResta is Resta*(-1),
        AbsResta < 0.11.

obtenAcel(V1,V2,'constante') :-
        Resta is V2-V1,
        Resta > 0,
        Resta < 0.11.

obtenAcel(V1,V2,'constante') :-
        Resta is V2-V1,
        Resta =  0,
        Resta < 0.11.

obtenAcel(V1,V2,'aumenta') :-
        V1<V2.

obtenAcel(V1,V2,'disminuye') :-
        V1>V2.

aceAngulo(A1p,A2p,A3p,R) :-
```

```
        Difa1 is A2p-A1p,
        Difa2 is A3p-A2p,
        obtenAcel(Difa1,Difa2,R).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%     Aceleracion Carro
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

aceCarro(X1p,X2p,X3p,R) :-
        Difa1 is X2p-X1p,
        Difa2 is X3p-X2p,
        obtenAcel(Difa1,Difa2,R).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%  Posicion Carro
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

obtenPosCarro(V2,V3,'noCambia') :-
        Resta is V3-V2,
        Resta < 0,
        AbsResta is Resta*(-1),
        AbsResta < 0.1.

obtenPosCarro(V2,V3,'noCambia') :-
        Resta is V3-V2,
        Resta > 0,
        Resta < 0.1.

obtenPosCarro(V2,V3,'noCambia') :-
        Resta is V3-V2,
        Resta =  0,
        Resta < 0.1.

obtenPosCarro(V2,V3,'alejaIzquierda') :-
        V3<V2,
        V3<0,
        V2<0.

obtenPosCarro(V2,V3,'acercaIzquierda') :-
        V3>V2,
        V3<0,
        V2<0.

obtenPosCarro(V2,V3,'alejaDerecha') :-
        V3>V2,
        V3>0,
        V2>0.

obtenPosCarro(V2,V3,'acercaDerecha') :-
        V3<V2,
        V3>0,
        V2>0.

posicionCarro(X2,X3,R) :-
      obtenPosCarro(X2,X3,R).
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Dirección Péndulo
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

obtenDirP(V2,V3,'levantandoDerecha') :-
     V3 > V2,
     V2 < -0.1,
     V3 < 0.1.

obtenDirP(V2,V3,'cayendoIzquierda') :-
        V3 < V2,
        V2 < -0.1,
        V3 < -0.1.

obtenDirP(V2,V3,'levantandoIzquierda') :-
        V3<V2,
        V2>0.1,
        V3>0.1.

obtenDirP(V2,V3,'cayendoDerecha') :-
        V3>V2,
        V2>0.1,
        V3>0.1.

obtenDirP(V2,V3,'equilibrio') :-
        V3 > -0.1,
        V2 =< 0.1,
        V3 > -0.1,
       V3=<0.1.

pendulo(A2,A3,R) :-
     obtenDirP(A2,A3,R).


%%% 300 ejemplos
ej(dato1(0.365005,0.117936,0.711937,0.52007),dato2(0.48332,0.142107,0.706
658,0.562685),dato3(0.335375,0.16227,0.895191,0.612756),right).
ej(dato1(0.48332,0.142107,0.706658,0.562685),dato2(0.335375,0.16227,0.895
191,0.612756),dato3(0.38626,0.168978,0.892925,0.63066),right).
ej(dato1(0.335375,0.16227,0.895191,0.612756),dato2(0.38626,0.168978,0.892
925,0.63066),dato3(0.439212,0.176703,0.890575,0.648518),right).
ej(dato1(0.38626,0.168978,0.892925,0.63066),dato2(0.439212,0.176703,0.890
575,0.648518),dato3(0.494538,0.185487,0.88813,0.66633),right).
ej(dato1(0.439212,0.176703,0.890575,0.648518),dato2(0.494538,0.185487,0.8
8813,0.66633),dato3(0.265608,0.195378,1.08022,0.684093),right)
```

Apéndice V

# Ejemplos de Modelos obtenidos con PROGOL

## Modelo de Tendencias del Experto

```
ej(dato1(A,B,C,D),dato2(E,F,G,D),left) :-
        tendApunto(A,E,aumenta).

ej(dato1(A,B,C,D),dato2(E,F,G,H),left) :-
        tendApunto(A,E,aumenta),
        tendA(A,B,disminuye),
        tendXpunto(A,G,disminuye).

ej(dato1(A,B,C,D),dato2(E,F,G,H),inertia) :-
        tendA(B,F,noCambia),
        tendX(D,H,noCambia).

ej(dato1(A,B,C,D),dato2(E,F,G,H),right) :-
        tendApunto(F,E,aumenta),
        tendX(D,H,noCambia).

ej(dato1(A,B,C,D),dato2(E,F,G,H),right) :-
         tendApunto(A,E,disminuye),
        tendXpunto(C,G,aumenta),
        tendX(D,H,noCambia).

ej(dato1(A,B,C,D),dato2(E,B,F,D),right).

ej(dato1(A,B,C,D),dato2(E,B,F,G),left).

ej(dato1(A,B,C,D),dato2(E,F,G,H),left) :-
        tendA(B,F,noCambia),
         tendX(D,H,noCambia).

ej(dato1(A,B,C,D),dato2(E,B,F,G),inertia) :-
        tendX(D,G,noCambia).

ej(dato1(A,B,C,D),dato2(E,F,C,D),inertia).

ej(dato1(A,B,C,D),dato2(E,F,G,D),inertia) :-
        tendApunto(A,F,disminuye),
        tendApunto(B,C,noCambia).
```

## Modelo de Aceleración/Dirección del Experto

```
ej(dato1(A,B,C,D),dato2(E,F,G,H),dato3(I,J,G,K),inertia).

ej(dato1(A,B,C,D),dato2(E,B,F,G),dato3(H,I,J,G),left).
```

```
ej(dato1(A,B,C,D),dato2(E,F,G,D),dato3(H,I,J,K),left) :-
        pendulo(B, G,equilibrio).

ej(dato1(A,B,C,D),dato2(E,B,F,G),dato3(H,I,J,K),left) :-
        posicionCarro(B,D,alejaIzquierda).

ej(dato1(A,B,C,D),dato2(E,F,G,H),dato3(I,F,J,K),left).

ej(dato1(A,B,C,D),dato2(E,F,G,H),dato3(I,J,K,L),left) :-
        pendulo(A,E,cayendoDerecha),
        pendulo(A,I,cayendoDerecha),
        pendulo(C,B,levantandoDerecha).

ej(dato1(A,B,C,D),dato2(E,F,G,H),dato3(I,J,K,L),inertia) :-
        pendulo(A, E,cayendoDerecha),
        pendulo(A,I,cayendoDerecha),
        pendulo(D,G,levantandoDerecha).

ej(dato1(A,B,C,D),dato2(E,F,C,G),dato3(H,I,C,J),right).

ej(dato1(A,B,C,D),dato2(E,F,G,H),dato3(I,J,K,L),right) :-
        pendulo(A,I,levantandoIzquierda),
        pendulo(C,B,levantandoDerecha).

ej(dato1(A,B,C,D),dato2(E,F,G,H),dato3(I,J,K,L),right) :-
        pendulo(B, I,equilibrio),
        pendulo(D,C,levantandoDerecha),
        pendulo(D,G,levantandoDerecha).

ej(dato1(A,B,C,D),dato2(E,B,F,G),dato3(H,I,J,K),right) :-
        pendulo(A,A,equilibrio).

ej(dato1(A,B,C,D),dato2(E,F,G,H),dato3(I,J,K,L),right) :-
        pendulo(C, K,cayendoDerecha).

ej(dato1(A,B,C,D),dato2(E,F,G,H),dato3(I,J,C,K),left).

ej(dato1(A,B,C,D),dato2(E,F,G,H),dato3(I,F,J,K),inertia) :-
        pendulo(C,G,levantandoIzquierda),
          pendulo(C,J,levantandoIzquierda).
ej(dato1(A,B,C,D),dato2(E,B,C,F),dato3(G,H,I,J),inertia) :-
        pendulo(C, I,levantandoIzquierda).

ej(dato1(A,B,C,D),dato2(E,F,G,H),dato3(I,J,K,L),inertia) :-
        pendulo(C, K,cayendoDerecha).

ej(dato1(A,B,C,D),dato2(E,F,G,H),dato3(I,J,K,L),left) :-
        pendulo(A, I,levantandoDerecha),
        pendulo(C,K,levantandoIzquierda).

ej(dato1(A,B,C,D),dato2(E,F,C,D),dato3(G,H,I,J),inertia) :-
        pendulo(A,E,levantandoIzquierda).
```

## Modelo de Aceleración/Dirección del participante 6

```
ej(dato1(A,B,C,D),dato2(E,F,G,H),dato3(I,J,K,L),right) :-
        pendulo(F, J,cayendoDerecha),
        posicionCarro(H,L,noCambia),
        aceCarro(C,K,D,disminuye).

ej(dato1(A,B,C,D),dato2(E,F,G,H),dato3(I,J,K,L),right) :-
        pendulo(F,J,levantandoIzquierda),
        posicionCarro(H,L,noCambia),
        aceAngulo(A,I,I,aumenta),
        aceCarro(C,G,D,disminuye).

ej(dato1(A,B,C,D),dato2(E,F,G,H),dato3(I,J,K,L),left) :-
        pendulo(F, J,equilibrio),
        posicionCarro(H,L,noCambia),
        aceCarro(C, G,A,disminuye).

ej(dato1(A,B,C,D),dato2(E,F,G,H),dato3(I,J,K,L),left) :-
         posicionCarro(H,L,noCambia),
        aceAngulo(A,I,I,disminuye),
        aceAngulo(E,E,I,aumenta),
        aceCarro(C,G,C,aumenta),
        aceCarro(C,K,C,aumenta).

ej(dato1(A,B,C,D),dato2(E,F,G,H),dato3(I,J,K,L),inertia) :-
        posicionCarro(H,L,noCambia),
        aceAngulo(A,I,I,constante),
        aceCarro(C,G, A,disminuye),
        aceCarro(C,G,H,aumenta).

ej(dato1(A,B,C,D),dato2(E,F,G,H),dato3(I,J,K,L),left) :-
        pendulo(F, J,cayendoIzquierda),
        posicionCarro(H,L,noCambia),
        aceAngulo(A, E,I,constante),
        aceAngulo(A,I,I,aumenta),
        aceCarro(C, K,D,aumenta).

ej(dato1(A,B,C,D),dato2(E,F,G,H),dato3(I,J,K,L),inertia) :-
        posicionCarro(H, L,noCambia),
        aceAngulo(A,E,I,constante),
        aceCarro(C,G, K,constante).

ej(dato1(A,B,C,D),dato2(E,F,G,H),dato3(I,J,K,L),left) :-
        pendulo(F, J,cayendoIzquierda),
        aceAngulo(A,E,I,constante),
        aceAngulo(A,I,I,aumenta),
        aceCarro(C,G,K,constante).

ej(dato1(A,B,C,D),dato2(E,F,G,H),dato3(I,J,K,L),inertia) :-
        posicionCarro(H,L,alejaIzquierda),
        aceAngulo(A,I,I,aumenta),
        aceAngulo(E,E,I,disminuye),
        aceCarro(C,G,A,aumenta),
```

```
        aceCarro(G,K,B,aumenta).

ej(dato1(A,B,C,D),dato2(E,F,G,H),dato3(I,J,K,L),left) :-
        pendulo(F, J,cayendoIzquierda),
        aceAngulo(E,E,I,constante),
        aceCarro(C,G,C,aumenta).

ej(dato1(A,B,C,D),dato2(E,F,G,H),dato3(I,J,K,L),left) :-
        aceCarro(C, G,K,aumenta).

ej(dato1(A,B,C,D),dato2(E,F,G,H),dato3(I,J,K,L),inertia) :-
        aceCarro(C,G,K,aumenta).

ej(dato1(A,B,C,D),dato2(E,F,G,H),dato3(I,J,K,L),left) :-
        aceCarro(C,G,K,disminuye).

ej(dato1(A,B,C,D),dato2(E,F,G,H),dato3(I,J,K,L),right) :-
        aceCarro(C,G,K,constante).

ej(dato1(A,B,C,D),dato2(E,F,G,H),dato3(I,J,K,L),right) :-
        aceCarro(C, G,K,aumenta).

ej(dato1(A,B,C,D),dato2(E,F,G,H),dato3(I,J,K,L),inertia) :-
        aceCarro(C,G,K,disminuye).

ej(dato1(A,B,C,D),dato2(E,F,G,H),dato3(I,J,K,L),right) :-
        pendulo(F, J,equilibrio), posicionCarro(H,L,noCambia),
        aceCarro(C, G,K,disminuye).
```