Universidad Politécnica de Madrid
Escuela Universitaria de Informática

Trabajo fín de carrera

Lenguaje para implementar sistemas expertos con modelos algebraicos

autor
Roberto Maestre Martínez
tutor
Antonio Hernando Esteban
departamento
Sistemas Inteligentes Aplicados

Diciembre, 2010

MADRID

Agradecimientos

"The path from dreams to success does exist.
May you have the vision to find it,
the courage to get on to it,
and the perseverence to follow it.

Wishing you a great journey."

Kalpana Chawla

En primer lugar quería agradecer al profesor Antonio Hernando Esteban la confianza depositada en mí a la hora de abordar este proyecto. Desde el primer momento me dirigió de una manera clara y concisa, enseñándome a enfocar el esfuerzo y profundizar en los fundamentos teóricos, ajustando así el proyecto a mis inquietudes intelectuales. En este intenso periodo alimentó mi hambre investigadora con una mezcla de conocimiento, energía, positividad e ilusión.

A mi Alma máter, el Consejo Superior de Investigaciones Científicas (CSIC); institución que siempre me brindó una oportunidad y de la que he aprendido los valores del trabajo, la excelencia y la perseverancia.

A mis compañeros del proyecto Dynamic Complexity of Cooperation-Based Self-Organizing Networks de la European Science Foundation y de la Unidad de Sistemas de Información Geográfica del CSIC con los que he compartido los últimos años.

A mi familia, Epifanio Ángel, Mercedes, Daniel y Corina; por su apoyo incondicional en todo momento y su crítica positiva; por enseñarme con su ejemplo a pensar libremente a través de la razón. Por no fallar nunca. Por ser mi estrella polar.

A Sol por mostrarme con su ejemplo de lucha y valentía a no perder nunca de vista el camino

que lleva a la consecución de los sueños.

Al doctor Antonio Zarazaga Monzón por la información aportada en el diseño del sistema experto propuesto en este trabajo.

A José Gabriel Pérez Díez, Jesús López Sánchez y su asignatura Compiladores e Interpretes en donde aprendí los conceptos necesarios para abordar el diseño y construcción del traductor propuesto en este trabajo.

Abstract

According to previous mathematical results, expert systems can be quickly and easily implemented on specialized mathematical software (Computer Algebra Systems). These mathematical results provide two equivalent different approaches to implement algebraically expert systems: one based on propositional Boolean logic and another based on the 'concept-attribute-value' representation paradigm. Both approaches are equivalent in the sense that every expert system implemented through one algebraic approach can be also implemented on the another algebraic approach. However, the performance of these algebraic approaches depends on the specific expert system. Consequently, the most suitable algebraic approach can only be determined if the specific expert system is implemented in both algebraic approaches in order to compare their performance. In this document we have designed a natural language to implement algebraically expert systems which provides these three main advantages: we can implement directly any expert system in both algebraic approach since our language is independent of the algebraic approach; we can use any computer algebra system since our language is independent of the computer algebra system used; and we do not need to be acquainted with specialized abstract mathematical concepts like Gröbner basis or normal forms since our language is natural and does not make use of these concepts.

Resumen

De acuerdo con resultados matemáticos previos, se puede implementar sistemas expertos de una manera sencilla y rápida usando software matemático especializado (los sistemas de algebra computacional). Estos resultados matemáticos proporcionan dos aproximaciones algebraicas equivalentes aunque diferentes: una de ellas está basada en lógica proposicional Booleana, la otra está basada en el paradigma de representación 'Concepto-Atributo-Valor'. Ambas aproximaciones son equivalentes en el sentido de que cualquier sistema experto implementado en una aproximación algebraica puede ser implementado también en la otra aproximación algebraica. No obstante, la ejecución de ambas aproximaciones algebraicas depende del sistema experto específico y por ello, la aproximación algebraica más adecuada sólo puede ser determinada si se implementa el sistema experto en ambas aproximaciones con el fin de poder comparar su eficiencia. En este proyecto final de carrera se ha diseñado un lenguaje natural de implementación algebraica de sistemas experto que proporciona las siguientes ventajas: se puede implementar directamente cualquier sistema experto en cualquiera de las dos aproximaciones algebraicas ya que nuestro lenguaje es independiente de la aproximación algebraica; se puede usar cualquier sistema de algebra computacional puesto que nuestro lenguaje es independiente del sistema algebraico usado; y no es necesario estar familiarizado con conceptos abstractos como las bases de Gröbner o las formas normales ya que nuestro lenguaje no utiliza esos conceptos.

Obra derivada

Como fruto del presente trabajo de fín de carrera, se ha enviado un artículo a la revista 'Expert Systems with Applications', situada en el primer cuartil del JCR, con el siguiente título:

"A natural language for implementing algebraically expert systems"

Antonio Hernando, Roberto Maestre-Martínez, Eugenio Roanes-Lozano.

Índice

Índice de cuadros

Índice de figuras

1. Estado de la cuestión, motivación y objetivos

Los sistemas expertos son programas computacionales, que dentro de un dominio de conocimiento, tratan de simular las decisiones que expertos humanos en este dominio tomarían. En los últimos años, muchos investigadores se han centrado en el desarrollo de técnicas para la representación de conocimiento humano en una computadora y para el razonamiento automático.

Una manera interesante para representar conocimiento en un sistema experto se basa en describirlo a través de la lógica proposicional. Por medio de un resultado matemático [1] basado en trabajos previos [2, 3, 4, 5, 6], las cuestiones relativas a la inferencia del sistema experto pueden ser tratadas algebraicamente. Por esto, los sistemas expertos basados en lógica proposicional booleana pueden ser fácilmente implementados por medio de Sistemas de Computación Algebraica (CAS, en inglés) como CoCoA [7, 8], Singular [9] o PolyBoRi [10]. Haciendo uso de estos resultados, se han implementado diversos sistemas expertos en los últimos años [11, 12, 13, 14].

El paradigma 'Concepto-Atributo-Valor' [15] proporciona una forma más rápida y natural de representar conocimiento. Recientemente, se ha obtenido un resultado matemático similar[16] que permite implementar sistemas expertos basados en este paradigma con CAS.

Aunque el método algebraico dado en lógica proposicional booleana y el basado en el paradigma 'Concepto-Atributo-Valor' son diferentes maneras de implementar sistemas expertos matemáticamente, ambos son equivalentes en el sentido de que cualquier sistema experto implementado en un modelo puede ser implementado en el otro.

En este trabajo, mostramos una comparación entre el rendimiento de ambos enfoques algebraicos con el objetivo de afirmar que la cuestión sobre que enfoque algebraico es

mejor, depende directamente de las características del sistema experto. Debido a este motivo, solo podremos averiguar que enfoque debemos elegir para cada sistema experto si lo implementamos en ambos modelos, y esto, implica duplicar esfuerzos.

Otra cuestión relacionada con la implementación del sistema experto es la elección del CAS sobre el que se ejecutará. Al igual que en la elección del método algebraico, cambiar el CAS en el que será ejecutado, implica el arduo trabajo adaptar los comandos al CAS destino. Además, aunque ambos enfoques proporcionan una manera rápida y fácil de implementar sistemas expertos, los desarrolladores deben de estar familiarizados con abstractos y complicados conceptos matemáticos como son la 'forma normal' y la 'base de Gröbner'.

Con el fin de evitar estos problemas, en este trabajo final de carrera desarrollaremos un nuevo lenguaje para implementar algebraicamente sistemas expertos. El uso de este lenguaje proporcionará las siguientes ventajas:

i) El lenguaje es independiente del enfoque algebraico elegido. En este trabajo mostraremos como traducir un sistema experto descrito en nuestro lenguaje en ambos enfoques, de esta manera, podremos generar código fácilmente para ambos enfoques con el fin de experimentar con ellos y elegir el que mejor rendimiento muestre.

ii) El lenguaje es independiente del CAS elegido. De esta manera, cambiar el CAS sólo conllevará compilar el sistema experto especificado en nuestro lenguaje.

iii) El lenguaje no usa notación matemática. Por lo tanto, los desarrolladores no necesitan conocer los conceptos matemáticos que son comúnmente utilizados en estos enfoques algebraicos (Forma normal y base de Gröbner).

iv) Nuestro lenguaje proporciona una manera sencilla de implementar sistemas expertos. De hecho, el código del sistema es normalmente más reducido y comprensible que el código descrito en el CAS elegido. Nuestro lenguaje proporciona una manera muy fácil

e intuitiva de escribir reglas (las cuales, son la manera más natural de describir la base de conocimiento de un sistema experto).

En el apartado 3, se dará una visión general de los enfoques algebraicos obtenidos para implementar sistemas expertos en sistemas de computación algebraica. En el apartado 4 y 5, se presentan los enfoques algebraicos relacionados con la lógica Booleana proposicional y el modelo 'Concepto-Atributo-Valor'. En el apartado 6, se describe el nuevo lenguaje que se ha diseñado para implementar sistemas expertos. Haciendo uso de este lenguaje, en el apartado 7 se desarrolla un pequeño sistema experto médico real. En el apartado 8, se hace un estudio comparativo de los diferentes enfoques algebraicos, concluyendo que la elección del enfoque algebraico más adecuado depende del sistema experto en concreto. Finalmente, en el apartado 10, se presentan las conclusiones.

2. Sistemas de computación algebraica

2.1. PolyBoRi

PolyBoRi esta implementado como una librería C++ para polinomios sobre anillos Booleanos, y proporciona un alto nivel de abstracción de tipos de datos para polinomios y monomios Booleanos, vectores exponente, así como para los anillos de polinomios subyacentes. Las variables del anillo pueden ser identificadas por sus índices o por una cadena personalizada. Las estructuras polinomiales y los monomios usan diagramas de decisión binarios cero suprimido (ZDDs) como tipo de almacenaje interno, pero es transparente para el usuario. La implementación actual utiliza el diagrama de decisiones de gestión de Cudd.

Además, las operaciones básicas con polinomios - como, la adición y la multiplicación - han sido implementadas y asociadas a los operadores correspondientes. Con el objetivo de conseguir una implementación eficiente, los operadores fueron reformulados en términos de conjuntos de operaciones, los cuáles son compatibles con el enfoque ZDD.

Una completa interfaz para Python permite el análisis sintáctico de los sistemas polinómicos complejos, también estrategias para el cálculo de la base de Gröebner. Además, con la herramienta de ipython las estructuras de datos PolyBoRi y sus procedimientos se pueden utilizar de forma interactiva como una herramienta de línea de comandos.

Además de optimizar las estructuras de datos, PolyBoRi dispone de una implementación de referencia para el cálculo de la base Gröbner utilizando una variante mejorada del algoritmo de slimgb, que se aplicó por primera vez en el sistema de álgebra computacional *Singular*. El "framework" fue desarrollado y diseñado conjuntamente con el autor original de slimgb.

El rendimiento inicial de PolyBoRi parece ser muy prometedor. Se puede observar, que la

ventaja de PolyBoRi crece con el número de variables. Para muchas aplicaciones prácticas este tamaño cada vez será mayor. Tenemos mucha confianza en que será posible abordar algunos de estos problemas en el futuro a través de métodos mas especializados. Éste es un punto clave en el desarrollo de PolyBoRi para facilitar soluciones de problemas específicos y soluciones de alto rendimiento

2.1.1. Versión utilizada

0.5

2.1.2. Definiciones

1. Declaración del anillo.

   Hay que destacar que PolyBoRi sólo trabaja sobre anillos Booleanos, por lo que el anillo será definido por defecto en el cuerpo $\mathbb{Z}_2$ y sólo habrá que especificar las variables.

   La sintaxis para definir un anillo en PolyBoRi es la siguiente:

   ```
   declare_ring([Block("n",m)],globals());
   ```

   donde *n* será el identificador de un *array* de variables y *m* será el tamaño de ese *array*. Cada variable del *array* se nombrará de la siguiente manera:

   $$n(0), n(1), n(2), \ldots, n(m-1)$$

2. Definición de polinomios

   El código es el siguiente:

   ```
   p0=x(0)^2+x;
   ```

   que es equivalente a $p_0 = x_0^2 + x$.

3. Definición de ideales

   Para definir un ideal en PolyBoRi se hará de la siguiente manera:

   ```
   J=[p1,p2,...,pn];
   ```

   donde $p1, p2, ...pn$ son polinomios.

4. Definición de funciones

   El código es el siguiente:

   ```
   def nombrefunction(par1,par2,...,parn):
   return <expresion>
   ```

   donde:

   - 'def' y 'return' son palabras reservadas.
   - $nombrefunction$ es el nombre de la función y por la cuál será llamada desde el cualquier parte del código.
   - $par1, ..., parn$ son los parámetro pasados a la función.
   - 'Polinomial($< expresion >$)' hace referencia implícita al que el tipo de la expersión es un polinomio.

   Una $< expresion >$ puede ser por ejemplo: $1+1$, o $x(0)+1$, o $Polynomial(x(0)^2)*0$.

5. Asignación de variables

   El código es el siguiente:

   ```
   p0=x(0);
   ```

6. Cálculo de la base de Gröebner

   El código es el siguiente:

   ```
   groebner_basis([p0,p1,...pn],heuristic=False);
   ```

donde:

- $p1, p2, ...pn$ son polinomios.
- $heuristic = False$ desactivamos la heurística.

7. Cálculo de la forma normal

   Utilizamos la siguiente función de reducción:

   ```
   nf=eliminate(gb)[1];
   ```

   Ya que 'eliminate' devuelve una función, la llamada para reducir un polinomio dado es la siguiente:

   ```
   print nf(x(0));
   ```

   donde *gb* es un vector con la base de Gröebner previamente calculada.

2.2. Singular

Singular es un sistema de computación algebraica para cálculos con polinomios, con especial énfasis en el álgebra conmutativa y no-conmutativa, geometría algebraica y teoría de la singularidad. Es libre y de código abierto bajo licencia GNU.

Singular proporciona:

- Algoritmos altamente eficientes.
- Una gran cantidad de algoritmos avanzados en los ámbitos mencionados implementados, bien en el núcleo (escrito en C/C++), o como bibliotecas.
- Principales objetos de cómputo: polinomios, ideales y módulos sobre una gran variedad de anillos.
- Un lenguaje intuitivo, similar a C.

- Maneras fáciles de hacer que el usuario pueda realizar ampliaciones a través de bibliotecas.
- Un completo manual en línea y ayudas de funciones.

Su objeto principal de cómputo son ideales, módulos y matrices sobre un gran número de anillos base. Estos incluyen:

- Anillos de polinomios sobre varios cuerpos y algunos anillos (incluyendo los enteros).
- Localizaciones de las anteriores.
- Una clase general de algebras no conmutativas.
- Anillos cociente de lo anterior.

Los algoritmos básicos de *Singular* manejan:

- Gröebner respecto bases estándar y resoluciones libres.
- Factorización polinomial.

2.2.1. Versión utilizada

3.0.4

2.2.2. Definiciones

1. Declaración del anillo:

   El código es el siguiente:

```
ring r=p,(n(0..m)),lp;
```

donde:

- p es el número entero que define el cuerpo.
- n es el identificador de un $array$ de variables.
- m el número de variables que contiene el $array$ n. Cada variable de ese $array$ se especifica de la siguiente manera:

$$n(0), n(1), n(2), ..., n(m-1)$$

2. Definición de polinomios

   El código es el siguiente:

```
poly p0=x(0)*x(1);
```

3. Definición de ideales

   El código es el siguiente:

```
ideal J=p0,p1,...,pn;
```

   donde $p1, p2, ...pn$ son polinomios.

4. Definición de funciones:

   El código es el siguiente:

```
proc nombrefunction(t1 par1,t2 par2,,...,tn parn) {
return(<expresion>);
}
```

   donde:

   - 'proc' y 'return' son palabras reservadas.

- *nombrefunction* es el nombre de la función y por la cual será llamada desde el cualquier parte del código.
- *tn* es el tipo de datos de la variable *vn*.
- $<$ *expresion* $>$ es la expresión cuyo resultado de ejecución devuelve la función.

5. Asignación de variables:

   El código es el siguiente:

   ```
   poly p0=1;
   ```

6. Cálculo de la base de Gröebner:

   El código es el siguiente:

   ```
   ideal BASE=slimgb([p0,p1,...,pn]);
   ```

   donde $p1, p2, ...pn$ son polinomios.

7. Cálculo de la forma normal:

   El código es el siguiente:

   ```
   reduce(pi,[p0,p1,...pn]);
   ```

   donde $p0, p2, ...pn$ y pi son polinomios.

2.3. CoCoA

- *CoCoA* es un programa para realizar cálculos con números y polinomios.
- Funciona en varios sistemas operativos.
- Realiza operaciones con enteros muy grandes y números racionales, polinomios, bases de Gröebner, sistemas de ecuaciones lineales,...

2.3.1. Versión utilizada

4.7.5

2.3.2. Definiciones

1. Declaración del anillo:

   El código es el siguiente:

   ```
   USE Z/(p)[n[0.m]];
   ```

   donde:

   - p es el número entero que define el cuerpo.
   - n es el identificador del $array$ de variables.
   - m es el número de variables del $array$ n. Cada variable de ese $array$ se especifica de la siguiente manera:

   $$n(0), n(1), n(2), ..., n(m-1)$$

2. Definición de polinomios:

   El código es el siguiente:

   ```
   V0:=x[0];
   ```

3. Definición de ideales:

   El código es el siguiente:

   ```
   MEMORY.J:=Ideal(p0,p1,...,pn);
   ```

   donde $p1, p2, ...pn$ son polinomios.

4. Definición de funciones:

   El código es el siguiente:

```
Define nombrefunction(par0,par1,...,parn)
 RETURN (1+M);
EndDefine;
```

donde:

- 'Define', 'RETURN' y 'EndDefine' son palabras reservadas.
- *nombrefunction* es el nombre de la función y por la cuál será llamada desde el cualquier parte del código.
- $par0, par1, ..., parn$ son los parámetros pasados a la función.
- $<$ *expresion* $>$ es la expresión cuyo resultado de ejecución devuelve la función.

5. Asignación de variables:

   El código es el siguiente:

   ```
   IC0:=nombrefunction(x[1]);
   ```

6. Cálculo de la base de Gröebner:

   El código es el siguiente:

   ```
   GBasis([p0,p1,...,pn]);
   ```

   donde $p1, p2, ...pn$ son polinomios.

7. Cálculo de la forma normal:

   El código es el siguiente:

   ```
   NF(pi,[p0,p1,...pn]);
   ```

   donde $p0, p1, ...pn$ y pi son polinomios.

3. Visión general de sistemas expertos implementados con álgebra computacional

3.1. Formalismo

Los sistemas expertos son programas informáticos compuestos por estos tres elementos:

Entrada La entrada de un sistema experto se refiere a la información relacionada con el entorno del sistema experto. Ya que el entorno puede cambiar, esta información también está sujeta a cambios a medida que pasa el tiempo. La información relativa a la entrada será denotada por $\mathcal{I}$.

Salida La salida de un sistema experto se refiere a la información relacionada con la información deducida por el sistema experto que será útil para llevar a cabo acciones en el entorno. La información relativa a la salida será denotada por $\mathcal{O}$.

Base de conocimiento La base de conocimiento del sistema experto se refiere a la información contenida en el sistema, la cuál se utiliza junto con la entrada del sistema experto para inferir la salida del sistema. Este conocimiento se suele describir por medio de reglas. Es decir, el conocimiento con la forma "si algunas condiciones relacionadas con la entrada se cumplen, entonces es necesario tomar una acción determinada". En esta parte, es posible encontrar conocimiento relativo a restricciones que debe cumplir la entrada del sistema experto. Este tipo de conocimiento normalmente es llamado "Restricciones de Integridad". La información relacionada con la base de conocimientos será denotada por $\mathcal{K}$.

Con el objetivo de diseñar programas que permitan simular el conocimiento de los humanos expertos, los investigadores en Inteligencia Artificial han logrado desarrollar diferentes modelos de representación capaces de especificar formalmente el conocimiento de los humanos expertos. En este capítulo, los sistemas expertos usan difentes variables $X_1, \ldots, X_m$ para

especificar un determinado estado de la entrada y la salida del sistema experto. Dependiendo de la representación de cada modelo, las variables pueden tomar diferentes posibles valores. Vamos a considerar diferentes tipos de variables:

Variables de entrada Las variables que describen la entrada de un sistema experto. Es decir, las variables que describe el entorno del sistema experto.

Variables de salida Las variables que describen posibles acciones o consecuencias que el sistema experto deduce.

Variables auxiliares Las variables que son necesarias para especificar la base de conocimiento del sistema experto.

Un estado S, es una posible situación en la cuál cada variable toma un valor. Dado un estado S y una variable X, la expresión $S(X)$ denota el valor que la variable X toma en el estado S.

Como veremos en los siguientes apartados, la información relacionada con la entrada, la salida y la base de conocimiento es definida a través de fórmulas. Para cada modelo de representación, se define formalmente el concepto de fórmula.

El conjunto de fórmulas en un modelo de representación particular se denota por $\mathcal{C}$. La entrada y la salida son por lo general restringidas a un tipo específico de fórmulas que limitan el posible valor de las variables de entrada y salida. Mientras que la entrada y la salida cambian dependiendo del entorno del sistema experto, las fórmulas que describen la base de conocimientos se mantienen invariables.

Las fórmulas se pueden cumplir o no en un estado en particular. Para cada modelo de representación, definimos formalmente cuando una fórmula A tiene un estado S. Las formulas de salida son deducidas de las fórmulas de entrada y de la base de conocimiento.

El problema de determinar que formulas son deducidas es formalmente definido de la siguiente manera:

Definición Una fórmula B es deducida por un sistema experto si y sólo si para cada estado S en que las fórmulas $A \in \mathcal{K} \cup \mathcal{I}$ cumplen el estado S, la fórmula B también cumple el estado S.

3.2. Modelo algebraico

Gracias a distintos resultados matemáticos, en este capítulo veremos cómo algunos sistemas expertos pueden ser implementados en sistemas de computación algebraica. Estos resultados matemáticos estan basados en traducir cada posible fórmula que describa conocimiento, $\mathcal{C}$, en polinomios de $\mathbb{Z}_q[x_1, \ldots, x_m]$ a través de una función $\varphi : \mathcal{C} \to \mathbb{Z}_q[x_1, \ldots, x_m]$. Como veremos en los siguientes apartados, diferentes anillos de polinomios y funciones φ son definidas dependiendo del modelo de representación utilizado para especificar el conocimiento del sistema experto.

Un concepto importante relacionado con los anillos de polinomios es el concepto de ideal. El ideal generado por un conjunto de polinomios $e_1, \ldots, e_n$ (denotado por la expresión $\langle e_1, \ldots, e_n \rangle$), es un subconjunto de polinomios de $\mathbb{Z}_q[x_1, \ldots, x_m]$ los cuales incluyen los elementos $e_1, \ldots, e_n$ y cumplen algunas propiedades. Dos ideales pueden ser sumados dando como resultado otro ideal nuevo. Como veremos en siguientes apartados, consideraremos los siguientes ideales cuya definición formal depende del modelo de representación utilizado.

Ideal J Ideal definido en el anillo de polinomios $\mathbb{Z}_q[x_1, \ldots, x_m]$ (donde q depende de la representación del modelo) requerido para definir la función φ que transforma fórmulas en polinomios.

Ideal I Ideal vinculado a la entrada del sistema experto.

$$I = \langle \varphi(A) | A \in \mathcal{I} \rangle$$

Ideal K Ideal relacionado con la base de conocimiento del sistema experto.

$$K = \langle \varphi(A) | A \in \mathcal{K} \rangle$$

Los ideales desempeñan un papel importante ya que el problema de determinar si un

sistema experto deduce una fórmula puede ser traducido al problema algebraico de determinar si un polinomio pertenece a un ideal. De hecho, como veremos en los próximos apartados, una fórmula A puede ser deducida por un sistema experto, si el polinomio $\varphi(A)$, pertenece al ideal $I + J + K$. Es decir:

$$A \in \mathcal{O} \Leftrightarrow \varphi(A) \in I + J + K$$

Este problema algebraico se resuelve por medio del concepto de bases de Gröebner y la forma normal [17]. Aunque estos conceptos son lo suficientemente complejos para ser definido formalmente aquí, (ver [18] para una introducción al algebra computacional) es suficiente saber que la forma normal de un polinomio, p, módulo un ideal $I + J + K$ (el cuál es denotado por $\mathrm{NF}(p, I + J + K)$) es otro polinomio que en entre otras condiciones cumple lo siguiente:

$$p \in I + J + K \Leftrightarrow \mathrm{NF}(p, I + J + K) = 0$$

3.3. Ejemplo

Consideremos un sistema experto el cuál aconseja sobre si se debe salir de casa en función de unas condiciones atmosféricas.

La **entrada** de este sistema experto muestra condiciones meteorológicas como pueden ser relativas al tiempo en general o relativo a la lluvia en particular.
La **salida** del sistema experto muestra si se debe salir de casa o no. Una de las opciones indica que uno debe esperar en casa a que el tiempo mejore.
La **base de conocimiento** del sistema experto describe cómo la entrada y la salida están relacionadas. Este conocimiento, $\mathcal{K}$ puede ser representado por reglas como la siguiente:

'Si el tiempo es muy bueno y no hay nada de lluvia, entonces se puede salir de casa'.

'Si el tiempo es bueno, entonces se puede salir de casa'.

'Si el tiempo es muy malo, entonces no se puede salir de casa'.

'Si no llueve, entonces el tiempo es bueno'.

La siguiente definición nos proporcionará el ejemplo estándar utilizado en las definiciones de los modelos.

Si consideramos que la entrada, $\mathcal{I}$, es que no hay lluvia y que el tiempo es muy bueno, entonces el sistema experto deducirá que se puede salir de casa.

4. Modelo lógico proposicional Booleano

4.1. Formalismo de representación

Un sistema experto basado en lógica proposicional Booleana utiliza variables Booleanas $X_1, ..., X_m$ para especificar un estado particular de la entrada y salida del sistema experto. Cada variable debe tomar el valor $verdadero$ o $falso$.
La base de conocimiento de este tipo de sistemas expertos es descrita por medio de fórmulas Booleanas proposicionales. Las fórmulas proposicionales son descritas por medio de las conocidas conectivas Booleanas lógicas $\neg$, $\wedge$, $\vee$, $\rightarrow$ acordes con la siguiente definición.

Definición: Una fórmula A es definida recursivamente de la siguiente manera:

- X_i donde X_i es una variable.
- $\neg B$ donde B es una fórmula.
- $B \vee C$ donde B y C son fórmulas.
- $B \wedge C$ donde B y C son fórmulas.
- $B \rightarrow C$ donde B y C son fórmulas.

Usaremos $\mathcal{C}$ para indicar el conjunto de fórmulas. Las reglas Booleanas proposicionales son las fórmulas más naturales de enunciar conocimiento. Las reglas son fórmulas con la siguiente forma

$$(A_1 \wedge ... \wedge A_r) \rightarrow (B_1 \vee ... \vee B_s)$$

donde cada fórmula $A_1, ..., A_r, B_1, ..., B_s$ es una variable o la negación de una variable.

Un estado S, es una posible situación en la cuál cada variable toma o bien el valor $verdadero$ o el valor $falso$. Dado un estado S y una variable X, la expresión $S(X)$ denota el valor de la variable X tomada en el instante S. De acuerdo con la siguiente definición, determinaremos

si una fórmula A se cumple o no en un estado particular S

Definición: Sea A una fórmula. Sea S un estado.

La fórmula A se cumple en el estado S si y sólo si:

Caso A es X_i donde X_i es una variable.

A se cumple en $S \Leftrightarrow S(X_i)$ es $verdadera$

Caso A es $\neg B$ donde B es una fórmula.

A se cumple en $S \Leftrightarrow$ B no se cumple en S

Caso A es $B \vee C$ donde B y C son fórmulas.

A se cumple en $S \Leftrightarrow B$ se cumple en S o C se cumple en S

Caso A es $B \wedge C$ donde B y C son fórmulas.

A se cumple en $S \Leftrightarrow B$ se cumple en S y C se cumple en S

Caso A es $B \rightarrow C$ donde B y C son fórmulas.

A se cumple en $S \Leftrightarrow$ la fórmula $\neg B \vee C$ se cumple en S.

Por medio de las fórmulas describimos la entrada, salida y la base de conocimiento de un sistema experto.

Entrada Le entrada es un conjunto de fórmulas con la forma X o $\neg X$ donde X es una variable de entrada que describe el entorno del sistema experto. Mientras que la fórmula X en la entrada representa que la variable de entrada X toma el valor $verdadero$, la fórmula $\neg X$ representa que la variable de salida X toma el valor $falso$. Naturalmente, el conjunto de fórmulas que forman la entrada de los sistemas expertos cambian dependiendo del entorno del sistema experto (el cuál describe la entrada del sistema experto). El símbolo $\mathcal{I}$ denota el conjunto de fórmulas que representan la entrada del sistema experto.

Base de conocimiento La base de conocimiento es un conjunto finito de fórmulas. A diferencia de la entrada, esas fórmulas son estáticas y no cambian a lo largo del tiempo. El símbolo

$\mathcal{K}$ denota el conjunto de fórmulas que representan la base de conocimiento del sistema experto.

Salida La salida es un conjunto de fórmulas con la forma X o $\neg X$ donde X es una variable de salida. Estas fórmulas son deducidas por el sistema experto por medio de las fórmulas relacionadas con la entrada y la base de conocimiento. Como en la entrada, una fórmula X representa que la variable de salida X toma el valor $verdadero$ y la fórmula $\neg X$ representa que la fórmula X toma el valor $falso$. Obviamente, este conjunto de fórmulas cambian dependiendo de la entrada del sistema experto. El símbolo $\mathcal{O}$ denota el conjunto de fórmulas que describe la salida del sistema experto

En esta apartado desarrollaremos el ejemplo de sistema experto descrito en el apartado 3.3 en lógica proposicional.

Las variables proposicionales que vamos a utilizar son:

Variables de Entrada Para el tiempo, utilizaremos las siguientes seis variables proposicionales:

x1(0): el tiempo es muy bueno

x1(1): el tiempo es bueno

x1(2): el tiempo es normal

x1(3): el tiempo es regular

x1(4): el tiempo es malo

x1(5): el tiempo es muy malo

Para comprobar lo mucho o poco que llueve, utilizaremos las siguientes cuatro variables proposicionales:

x2(0): llueve mucho

x2(1): llueve normal

x2(2): llueve poco

x2(3): no llueve nada

Variables de Salida Para salir de casa, utilizaremos las dos siguientes variables proposicionales

x0(0): salimos de casa

Ahora pasaremos a describir el conjunto de fórmulas, $\mathcal{K}$ que constituyen la base de conocimiento: En primer lugar hablaremos de las restricciones de integridad:

1. En relación con el tiempo: no puede ocurrir que el tiempo sea a la vez muy bueno, bueno, normal, regular, malo y muy malo. Con lo cual esta idea se describe con las siguientes restricciones de integridad:

$$IC1 : \neg(x1(0) \wedge x1(1))$$

$$IC2 : \neg(x1(0) \wedge x1(2))$$

$$IC3 : \neg(x1(0) \wedge x1(3))$$

$$IC4 : \neg(x1(0) \wedge x1(4))$$

$$IC5 : \neg(x1(0) \wedge x1(5))$$

$$IC6 : \neg(x1(1) \wedge x1(2))$$

$$IC7 : \neg(x1(1) \wedge x1(3))$$

$$IC8 : \neg(x1(1) \wedge x1(4))$$

$$IC9 : \neg(x1(1) \wedge x1(5))$$

$$IC10 : \neg(x1(2) \wedge x1(3))$$

$$IC11 : \neg(x1(2) \wedge x1(4))$$

$$IC12 : \neg(x1(2) \wedge x1(5))$$

$$IC13 : \neg(x1(3) \wedge x1(4))$$

$$IC14 : \neg(x1(3) \wedge x1(5))$$

$$IC15 : \neg(x1(4) \wedge x1(5))$$

y también esta la restricción que dice el tiempo tiene que ser o muy bueno, o bueno, o normal, o regular, o malo, o muy malo:

$$IC16 : (x1(0) \vee x1(1) \vee x1(2) \vee x1(3) \vee x1(4) \vee x1(5))$$

2. En relación con la lluvia: no puede ocurrir que llueva mucho, normal, poco o no llueva al mismo tiempo:

$$IC17 : \neg(x2(0) \wedge x2(1))$$

$$IC18 : \neg(x2(0) \wedge x2(2))$$

$$IC19 : \neg(x2(0) \wedge x2(3))$$

$$IC20 : \neg(x2(1) \wedge x2(2))$$

$$IC21 : \neg(x2(1) \wedge x2(3))$$

$$IC22 : \neg(x2(2) \wedge x2(3))$$

y también esta la restricción que dice que llueve mucho, llueve normal, poco o no llueve nada:

$$IC23 : (x2(0) \vee x2(1) \vee x2(2) \vee x2(3))$$

Por otro lado, la reglas que relacionan la entrada con la salida son las mostradas a continuación:

1. Si el tiempo es muy bueno y no hay nada de lluvia, entonces se puede salir de casa

$$R1 : (x1(0) \wedge x2(3)) \rightarrow x0(0)$$

2. Si el tiempo es bueno, entonces se puede salir de casa

$$R2 : x1(1) \rightarrow x0(0)$$

3. Si el tiempo es muy malo, entonces no se puede salir de casa

$$R3 : x1(5) \rightarrow \neg x0(0)$$

4. Si no llueve, entonces el tiempo es bueno

$$R4 : x2(3) \rightarrow x1(1)$$

La base de conocimiento sería el conjunto de fórmulas

$$\mathcal{K} = \{IC1, ..., IC23, R1, ...R4\}$$

Si se quisiera describir como entrada 'que no hay lluvia' entonces la entrada sería:

$$\mathcal{I} = \{x2(3)\}$$

Con esta entrada se deduciría la proposición $x0(0)$, y esto es que se puede salir de casa.

4.2. Modelo algebraico

El problema de determinar si una fórmula puede ser deducida desde otras puede ser tratados algebraicamente. De hecho, representando cada fórmula en $\mathcal{C}$ como un polinomio Booleano, nosotros podemos transformar el problema de determinar si una fórmula se puede deducir de las otras en un problema algebraico. En primer lugar, vamos a considerar cómo las fórmulas son traducidas a polinomios Booleanos. Necesitamos definir el ideal J en $\mathbb{Z}_2[x_1, \ldots, x_m]$:

$$J = \langle x_1^2 + x_1, \ldots, x_m^2 + x_m \rangle$$

Una vez definido el ideal J, las fórmulas son transformadas en un polinomio del $\mathbb{Z}_2[x_1, \ldots, x_m]$ por medio de la siguiente función:

Definición Definimos recursivamente la función $\varphi : \mathcal{C} \longrightarrow \mathbb{Z}_2[x_1, \ldots, x_m]$ como sigue:

Caso A es X_i donde X_i es una variable.

$$\varphi(A) = \varphi(X_i) = x_i$$

Caso A es $\neg B$ donde B es una fórmula.

$$\varphi(A) = \varphi(\neg B) = \mathrm{NF}(1 + \varphi(B), J)$$

Caso A es $B \vee C$ donde B y C son fórmulas.

$$\varphi(A) = \varphi(B \vee C) = \mathrm{NF}(\varphi(B) \cdot \varphi(C), J)$$

Caso A es $B \vee C$ donde B y C son fórmulas.

$$\varphi(A) = \varphi(B \wedge C) = \mathrm{NF}(\varphi(B) + \varphi(C) + \varphi(B) \cdot \varphi(C), J)$$

Caso A es $B \to C$ donde B y C son fórmulas.

$$\varphi(A) = \varphi(B \to C) = \mathrm{NF}(1 + \varphi(B) + \varphi(B) \cdot \varphi(C), J)$$

donde NF especifica la forma nomal de un polinomio módulo un ideal.

En relación con los sistemas expertos basados en lógica Booleana proposicional, nosotros consideraremos los siguientes ideales:

Ideal J Ideal necesario para definir la función φ .

$$J = \langle x_1^2 + x_1, \ldots, x_m^2 + x_m \rangle$$

Ideal I Ideal relacionado con la entrada del sistema experto.

$$I = \langle \varphi(A) | A \in \mathcal{I} \rangle$$

Ideal K Ideal relacionado con la base de conocimiento del sistema experto.

$$K = \langle \varphi(A) | A \in \mathcal{K} \rangle$$

Una vez transformadas las fórmulas en polinomios, nosotros podemos tratar algebraicamente la cuestión de determinar si una fórmula puede ser deducida por el sistema experto.

Teorema 4.1 *Una fórmula B puede ser deducida por el sistema experto si y solo si:*

$$\varphi(B) \in I + J + K$$

Este teorema puede ser reescrito de la siguiente manera:

Teorema 4.2 *Sea X_i una variable de salida.*

- $S(X_i)$ *es* $verdadero \Leftrightarrow NF(x_i, I + J + K) = 0$
- $S(X_i)$ *es* $falso \Leftrightarrow NF(x_i, I + J + K) = 1$

Este último teorema es especialmente útil ya que podemos encontrar el valor de salida calculando la forma normal de un polinomio modulo un ideal (el cual es calculado por un sistema de computación algebraica).

4.3. Implementación en el sistema de computación algebraica Singular

En este apartado usaremos un sistema computación algebraica llamado Singular con el objetivo de mostrar como un sistema experto puede ser implementado. Primero de todo, necesitamos especificar el anillo en el que serán traducidas las fórmulas. En este ejemplo considerado en el apartado 3.3, el anillo será definido en Singular de la siguientes manera:

```
ring r=2,(x0(0..1),x1(0..5),x2(0..3)),lp;


poly s1=x0(0)^2+x0(0);

poly s2=x1(0)^2+x1(0);

poly s3=x1(1)^2+x1(1);

poly s4=x1(2)^2+x1(2);

poly s5=x1(3)^2+x1(3);

poly s6=x1(4)^2+x1(4);

poly s7=x1(5)^2+x1(5);

poly s8=x2(0)^2+x2(0);

poly s9=x2(1)^2+x2(1);

poly s10=x2(2)^2+x2(2);

poly s11=x2(3)^2+x2(3);


ideal J=s1,s2,s3,s4,s5,s6,s7,s8,s9,s10,s11;

J = std(J);
```

También definimos las conectivas lógicas de acuerdo con la definición de φ, en la sintaxis de Singular:

```
proc NEG(poly M) {return(reduce(1+M,J));}

proc O(poly M,poly N) {return(reduce(M*N,J));}

proc IMP(poly M,poly N) {return(O(NEG(M),N));}

proc Y(poly M,poly N) {return(NEG(O(NEG(M),NEG(N))));}
```

Ahora nosotros definiremos la base de conocimiento del sistema experto, los comandos necesarios son los siguientes:

```
poly ic1=O(O(O(O(O(x1(5),x1(4)),x1(3)),x1(2)),x1(1)),x1(0));

poly ic2=NEG(Y(x1(0),x1(1)));

poly ic3=NEG(Y(x1(0),x1(2)));

poly ic4=NEG(Y(x1(0),x1(3)));

poly ic5=NEG(Y(x1(0),x1(4)));

poly ic6=NEG(Y(x1(0),x1(5)));

poly ic7=NEG(Y(x1(1),x1(2)));

poly ic8=NEG(Y(x1(1),x1(3)));

poly ic9=NEG(Y(x1(1),x1(4)));

poly ic10=NEG(Y(x1(1),x1(5)));

poly ic11=NEG(Y(x1(2),x1(3)));

poly ic12=NEG(Y(x1(2),x1(4)));

poly ic13=NEG(Y(x1(2),x1(5)));

poly ic14=NEG(Y(x1(3),x1(4)));

poly ic15=NEG(Y(x1(3),x1(5)));

poly ic16=NEG(Y(x1(4),x1(5)));
```

```
poly ic17=O(O(O(x2(3),x2(2)),x2(1)),x2(0));

poly ic18=NEG(Y(x2(0),x2(1)));

poly ic19=NEG(Y(x2(0),x2(2)));

poly ic20=NEG(Y(x2(0),x2(3)));

poly ic21=NEG(Y(x2(1),x2(2)));

poly ic22=NEG(Y(x2(1),x2(3)));

poly ic23=NEG(Y(x2(2),x2(3)));


poly r1=IMP(Y((x1(0)),x2(3)),(x0(0)));

poly r2=IMP((x1(1)),(x0(0)));

poly r3=IMP(x1(5),NEG(x0(0)));

poly r4=IMP(x2(3),(x1(1)));


ideal K=ic1,ic2,ic3,ic4,ic5,ic6,ic7,ic8,ic10,ic11,ic12,ic13,ic14,

ic15,ic16,ic17,ic18,ic19,ic20,ic21,ic22,ic23,r1,r2,r3,r4;

K = std(K);
```

Ahora consideraremos la entrada del sistema experto 'no hay nada de lluvia' ($\mathcal{I} = \{X2(3)\}$) En este caso es:

```
poly i0=x2(3);

ideal I=i0;

I = std(I);
```

Ahora nosotros podremos encontrar cual es el valor para la variable de salida S inferida por el sistema experto por medio de lo siguiente:

```
ideal BASE=slimgb(I+J+K);

ideal BASESTD=std(BASE);

print(reduce(x0(0),BASESTD));
```

Singular responde lo siguiente:

```
0

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

4.4. Implementación en el sistema de computación algebraica PolyBoRi

En este apartado usaremos un sistema de computación algebraica llamado PolyBoRi con el objetivo de mostrar cómo un sistema experto se puede implementar. En concreto utilizaremos el sistema experto descrito en el apartado anterior. Primero de todo, necesitamos realizar los 'imports' de algunas librerias:

```
from polybori.PyPolyBoRi import *
from polybori.gbcore import *
from polybori import *
```

A cotinuación se especifica el anillo en el que serán traducidas las fórmulas y las conectivas lógicas de acuerdo con la definición de φ. En este ejemplo considerado en el apartado anterior, el anillo estará definido en Polybori de la siguientes manera:

```
declare_ring([Block("x0",2),Block("x1",6),Block("x2",4)]
             ,globals());


def O(M,N): return Polynomial(M)*Polynomial(N)
def NEG(M): return 1+Polynomial(M)
def IMP(M,N): return O(NEG(M),N)
def Y(M,N): return NEG(O(NEG(M),NEG(N)))


ic1=O(O(O(O(O(x1(5),x1(4)),x1(3)),x1(2)),x1(1)),x1(0));
ic2=NEG(Y(x1(0),x1(1)));
ic3=NEG(Y(x1(0),x1(2)));
ic4=NEG(Y(x1(0),x1(3)));
```

```
ic5=NEG(Y(x1(0),x1(4)));

ic6=NEG(Y(x1(0),x1(5)));

ic7=NEG(Y(x1(1),x1(2)));

ic8=NEG(Y(x1(1),x1(3)));

ic9=NEG(Y(x1(1),x1(4)));

ic10=NEG(Y(x1(1),x1(5)));

ic11=NEG(Y(x1(2),x1(3)));

ic12=NEG(Y(x1(2),x1(4)));

ic13=NEG(Y(x1(2),x1(5)));

ic14=NEG(Y(x1(3),x1(4)));

ic15=NEG(Y(x1(3),x1(5)));

ic16=NEG(Y(x1(4),x1(5)));

ic17=O(O(O(x2(3),x2(2)),x2(1)),x2(0));

ic18=NEG(Y(x2(0),x2(1)));

ic19=NEG(Y(x2(0),x2(2)));

ic20=NEG(Y(x2(0),x2(3)));

ic21=NEG(Y(x2(1),x2(2)));

ic22=NEG(Y(x2(1),x2(3)));

ic23=NEG(Y(x2(2),x2(3)));
```

Ahora nosotros definiremos la base de conocimiento del sistema experto, los comandos necesarios son los siguientes:

```
r1=IMP(Y((x1(0)),x2(3)),(x0(0)));
```

```
r2=IMP((x1(1)),(x0(0)));

r3=IMP(x1(5),x0(1));

r4=IMP(x2(3),(x1(1)));


K=[r1,r2,r3,r4];
```

Ahora consideraremos la entrada del sistema experto 'no hay nada de lluvia' ($\mathcal{I} = \{X2(3)\}$) En este caso es:

```
i0=x2(3);

I=[i0];
```

Ahora nosotros podremos encontrar cual es el valor para la variable de salida S inferida por el sistema experto por medio de lo siguiente:

```
gb=groebner_basis(I+K);

nf=eliminate(gb)[1];

print nf(x0(0));
```

Polybori responde lo siguiente:

```
0


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

5. Modelo 'Concepto-Atributo-Valor'

5.1. Formalismo de representación

Un sistema experto basado en el modelo 'Concepto-Atributo-Valor' usa diferentes variables $X_1, ..., X_m$ con el objetivo de especificar un estado particular de la entrada y la salida del sistema experto. Cada variable puede tomar un subconjunto de posibles valores $V = \{v_1, ..., v_n\}$.

Las fórmulas en este modelo de representación son definidas como sigue a continuación: Definimos primero el concepto de fórmula atómica positiva:

Definición Una fórmula atómica positiva A es una fórmula de la forma:

- $X_i = v$ donde X_i es una variable y v es un posible valor de X_i.
- $X_i = X_j$ donde X_i and X_j son variables.

Definición Una fórmula A es definida recursivamente como sigue a continuación:

- A es una fórmula atómica positiva.
- $\neg A$ donde A es una fórmula.
- $A_1 \vee A_2$ donde A_1, A_2 son fórmulas.
- $A_1 \wedge A_2$ donde A_1, A_2 son fórmulas.
- $A_1 \rightarrow A_2$ donde A_1, A_2 son fórmulas.

Usaremos $\mathcal{C}$ para denotar el conjunto de fórmulas. De todas las fórmulas destacaremos dos tipos de fórmulas que son las fórmulas atómicas y las reglas.

Definición Una fórmula A es atómica si es una fórmula atómica positiva o tiene la forma $\neg A$ donde A es una fórmula atómica positiva.

Definición Una fórmula A es una regla si es de la forma:

$$(A_1 \wedge \ldots \wedge A_r) \rightarrow (B_1 \vee \ldots \vee B_s)$$

donde $A_1, \ldots, A_r, B_1, \ldots B_s$ son fórmulas atómicas.

La importancia de estos dos tipos de fórmulas se debe a que cualquier conjunto de fórmulas puede expresarse únicamente con fórmulas atómicas y reglas de acuerdo con la siguiente propiedad:

Proposición 5.1 *Dado un conjunto de fórmulas $\mathcal{K}$ hay un conjunto de fórmulas $\mathcal{K}'$ integrado únicamente por reglas y fórmulas atómicas que es equivalente a $\mathcal{K}$.*

De acuerdo con la siguiente proposición, una regla puede ser descrita como una disyunción de fórmulas atómicas. Esta proposición será útil en un siguiente apartado.

Proposición 5.2 *La regla $(A_1 \wedge \ldots \wedge A_r) \rightarrow (B_1 \vee \ldots \vee B_s)$ es equivalente a la fórmula:*

$$\neg A_1 \vee \ldots \vee \neg A_r \vee B_1 \vee \ldots \vee B_s$$

De acuerdo con la siguiente definición, podemos determinar si una fórmula A se cumple en un particular estado S.

Definición Sea A una fórmula. Sea S un estado.
La fórmula A se cumple en el estado S si y solo si:

Caso A es $X_i = v$ donde X_i es una variable y v un posible valor de X_i.

A se cumple en $S \Leftrightarrow S(X_i) = v$

Caso A es $X_i = X_j$ donde X_i y X_j son variables.

A se cumple en $S \Leftrightarrow S(X_i) = S(X_j)$

Caso A es $\neg B$ donde B es una fórmula.

A se cumple en $S \Leftrightarrow$ B no se cumple en S

Caso $A \equiv B \vee C$ donde B y C son fórmulas.

A se cumple en $S \Leftrightarrow$ o bien B se cumple en S o C se cumple en S

Caso $A \equiv B \wedge C$ donde B y C son fórmulas.

A se cumple en $S \Leftrightarrow B$ se cumple en S y C se cumple en S

Caso $A \equiv B \rightarrow C$ donde B y C son fórmulas.

A se cumple en $S \Leftrightarrow$ la fórmula $\neg B \vee C$ se cumple en S.

Por medio de las fórmulas describimos la entrada, la salida y la base de conocimiento de un sistema experto.

Entrada La entrada es un conjunto de fórmulas atómicas positivas. El símbolo $\mathcal{I}$ denota el conjunto de fórmulas que representan la entrada de un sistema experto.

Base de conocimiento La base de conocimiento es un conjunto finito de fórmulas. A diferencia de la entrada, estas fórmulas permanecen constantes e invariables a lo largo del tiempo. Como de costumbre, el símbolo $\mathcal{K}$ denota un conjunto de fórmulas que representan la base de conocimiento del sistema experto.

Salida La salida es un conjunto de fórmulas atómicas positivas. El sistema experto deduce estas fórmulas por medio de las fórmulas relacionadas con la entrada y la base de conocimiento. El símbolo $\mathcal{O}$ denota el conjunto de fórmulas que representan la salida del sistema experto.

En este apartado desarrollaremos el ejemplo de sistema experto descrito en el apartado 3.3 de acuerdo con el modelo 'Concepto-Atributo-Valor'.

Las variables que vamos a utilizar son:

Variables de entrada Dispondremos de las siguientes variables de entrada:

Variable relativa al tiempo, x(1).

Esta variable puede tomar los siguientes posibles valores $\{muybueno, bueno, normal, regular, malo, muymalo\}$ indicando el tiempo que hace.

Variable relativa a lo mucho o poco que llueve, x(2).

Esta variable puede tomar los siguientes posibles valores $\{mucha, bastante, poca, nada\}$.

Variables de salida Variable que informa si se sale de casa o no. Esta variable puede tomar los siguientes posibles valores $\{Verdadero, Falso\}$ indicando si se puede salir de casa o no.

Ahora pasaremos a describir el conjunto de fórmulas, $\mathcal{K}$ que constituyen la base de conocimiento. En primer lugar hay que destacar que en este modelo de representación, no es necesario definir restricciones de integridad ya que se tienen en cuenta implícitamente.

Por otro lado, la reglas que relacionan la entrada con la salida son las mostradas a continuación:

1. Si el tiempo es muy bueno y no hay nada de lluvia, entonces se puede salir de casa

$$R1 : (x(1) = muybueno) \wedge (x(2) = nada) \rightarrow (x(0) = verdadero)$$

2. Si el tiempo es bueno, entonces se puede salir de casa

$$R2 : (x(1) = bueno) \rightarrow (x(0) = verdadero)$$

3. Si el tiempo es muy malo, entonces no se puede salir de casa

$$R3 : (x(1) = muymalo) \rightarrow (x(0) = falso)$$

4. Si no llueve, entonces el tiempo es bueno

$$R4 : (x(2) = nada) \rightarrow (x(1) = bueno)$$

La base de conocimiento sería el conjunto de fórmulas

$$\mathcal{K} = \{R1, R2, R3, R4\}$$

Si se quisiera describir como entrada 'que no hay lluvia' entonces la entrada sería:

$$\mathcal{I} = \{x(2) = nada\}$$

Con esta entrada se deduciría la fórmula $(x(0) = verdadero)$, es decir, se puede salir de casa.

5.2. Modelo algebraico

El problema de determinar si una fórmula puede ser deducida de otras puede ser tratado algebraicamente de acuerdo con [16].

Sea p un número primo entero mayor o igual que el número de valores posibles que puede tomar cualquier variable. Primero, necesitamos codificar todos los posibles valores de las variables. Es decir, tenemos que definir una función, ϕ, entre los posibles valores de las variables y $\{0, ...p-1\}$:

$$\phi : V \longrightarrow \{0, ..., p-1\}$$

La función φ debe cumplir la siguiente propiedad: dado dos valores posibles de una misma variable, $v, w \in V$, $\phi(v) \neq \phi(w)$.

Dada una variable X_i, que podrá tomar un posible valor $\{v_1, ..., v_n\}$, definimos el polinomio s_i asociado a la misma:

$$s_i = (x - \phi(v_1)) \cdot (x - \phi(v_2)) \cdot ... \cdot (x - \phi(v_n))$$

El ideal J sobre $\mathbb{Z}_p[x_1, ..., x_m]$ es definido de la siguiente manera:

$$J = \langle s_1, ..., s_m \rangle$$

Una vez definido el ideal J, las fórmulas son transformadas en un polinonimo de $\mathbb{Z}_p[x_1, ..., x_m]$ por medio de la siguiente función:

Definición Definimos recursivamente la función $\varphi : \mathcal{C} \longrightarrow \mathbb{Z}_p[x_1, ..., x_m]$ de la siguiente manera:

Caso A es $X_i = v$ donde X_i es una variable y v un posible valor de X_i.

$$\varphi(A) = \varphi(X_i = v) = x_i - v$$

Caso A es $X_i = X_j$ donde X_i y X_j son variables.

$$\varphi(A) = \varphi(X_i = X_j) = x_i - x_j$$

Caso A es $\neg B$ donde B es una fórmula.

$$\varphi(A) = \varphi(\neg B) = \mathrm{NF}(\varphi(B)^{p-1} - 1, J)$$

Caso A es $B \vee C$ donde B y C son fórmulas.

$$\varphi(A) = \varphi(B \vee C) = \mathrm{NF}(\varphi(B) \cdot \varphi(C), J)$$

Caso A es $B \wedge C$ donde B y C son fórmulas.

$$\varphi(A) = \varphi(B \wedge C) = \varphi(\neg(\neg B \vee \neg C)))$$

Caso A es $B \to C$ donde B y C son fórmulas.

$$\varphi(A) = \varphi(B \to C) = \varphi(\neg B \vee C)$$

donde NF denota la forma normal de un polinomio modulo un ideal.

La traducción de una regla, $(A_1 \wedge ... \wedge A_r) \to (B_1 \vee ... \vee B_s)$, es especialmente simple:

$$\varphi((A_1 \wedge ... \wedge A_r) \to (B_1 \vee ... \vee B_s)) = \mathrm{NF}(\varphi(\neg A_1) \cdot ... \cdot \varphi(\neg A_r) \cdot \varphi(B_1) ... \cdot \varphi(B_s), J)$$

Una vez traducidas las fórmulas en polinomios, podemos tratar algebraicamente la cuestión

de determinar si una fórmula puede ser deducida por el sistema experto. Consideremos los siguientes ideales:

Ideal J Ideal para definir la función $\mathbb{Z}_p[x_1, \ldots, x_m]$ que tranforma fórmulas en polinomios.

Ideal I Ideal relacionado con la entrada del sistema experto.

$$I = \langle \varphi(A_i) | A_i \in \mathcal{I} \rangle$$

Ideal K Ideal relacionados con la base de conocimiento del sistema experto.

$$K = \langle \varphi(A_i) | A_i \in \mathcal{K} \rangle$$

Una vez traducidas las fórmulas en polinomios, podemos tratar algebraicamente la cuestion de determinar si el sistema experto deduce una fórmula.

Teorema 5.3 *Una fórmula B puede ser deducida por el sistema experto, si y solo si:*

$$\varphi(B) \in I + J + K$$

Este teorema puede ser también reescrito de la siguiente manera:

Teorema 5.4 *Sea X_i una variable de salida y v un posible valor de X_i.*

$$S(X_i) = v \Leftrightarrow NF(x_i, I + J + K) = \phi(v)$$

Este último teorema es especialmente útil ya que podemos encontrar el valor de salida calculando la forma normal de un polinomio modulo un ideal (el cual es calculado por un sistema de computación algebraica).

5.3. Implementación en el sistema de computación algebraica Singular

En este apartado usaremos Singular con el objetivo de mostrar como un sistema experto puede ser implementado en un sistema de computación algebraica. Utilizaremos el ejemplo del apartado anterior. En este ejemplo $p = 7$. Ahora podemos especificar el anillo sobre el que las fórmulas serán traducidas.

```
ring r=7,(s,x(1..2)),lp;
poly S1=(s-1)*s;
poly S2=(x(1)-5)*(x(1)-4)*(x(1)-3)*(x(1)-2)*(x(1)-1)*x(1);
poly S3=(x(2)-3)*(x(2)-2)*(x(2)-1)*x(2);
ideal J =S1,S2,S3;
J = std(J);
```

También definiremos la conectivas de acuerdo con la definición de φ. En sintaxis de Singular:

```
proc NEG(poly M) {return(reduce(M^6-1,J));}
proc O(poly M,poly N) {return(reduce(M*N,J));}
proc IMP(poly M,poly N) {return(O(NEG(M),N));}
proc Y(poly M,poly N) {return(NEG(O(NEG(M),NEG(N))));}
```

A continuación, definiremos la base de conocimiento del sistema experto. Aquí mostraremos los comandos necesarios para definir la base de conocimiento del sistema experto.

```
poly R1=IMP(Y(x(1),x(2)-3),s);
poly R2=IMP(x(1)-1,s);
poly R3=IMP(x(1)-5,s-1);
```

```
poly R4=IMP(x(2)-3,x(1)-1);

ideal K=R1,R2,R3,R4;

K = std(K);
```

Vamos a considerara como entrada que no hay lluvia ($\mathcal{I} = \{lluvia = nada\}$). En sintaxis de Singular:

```
ideal I=x(2)-3;

I = std(I);
```

Ahora podemos obtener cuál es el valor de la salida para la variable S inferida por el sistema experto por medio de:

```
ideal BASE=slimgb(J+I+K);

print(reduce(s,BASE));
```

Singular responde lo siguiente:

```
0

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

6. Lenguaje de implementación algebraico de sistemas expertos

En apartados previos de este trabajo hemos estudiado dos enfoques algebraicos para desarrollar sistemas expertos. Tal como veremos en el siguiente aparatado, la elección del enfoque más adecuado depende de la especificación del sistema experto. En este apartado, desarrollaremos un lenguaje para crear sistemas expertos, los cuales pueden ser traducidos al modelo lógico proposicional o al modelo 'Concepto-Atributo-Valor'

Dado que el modelo 'Concepto-Atributo-Valor' proporciona una manera más natural de implementar sistemas expertos, nuestro lenguaje está basado en este modelo. El siguiente código es un ejemplo de sistema experto descrito en nuestro lenguaje

6.1. Descripción del lenguaje

En este apartado describiremos la sintaxis de nuestro lenguaje. Lo siguiente muestra un ejemplo de sistema experto especificado en él.

```
BEGIN
  BEGIN_VARS
    A:    v1, v2, v3;
    B[2]: v1, v2;
    C[3]: true, false;
    D[2],E: w1, w2, w3;
  END_VARS


  BEGIN_KB
    (C[1]<>C[3]);
    (A=v1) AND (B[1]<>v2) AND (B[1]=B[2])-> (C[1]=C[2])
```

```
    OR (C[3]=true);
    (A<>v2) AND (B[1]=B[2]) -> (C[1]=true);
  END_KB
END.
```

Como puede verse, un sistema experto descrito en nuestro lenguaje contiene los dos siguientes apartados:

Definición de variables: Se refiere a la parte del código que está entre las partes 'BEGIN_VARS' y 'END_VARS'. Se define aquí cada variable, junto con los valores que esta variable puede tomar. El lenguaje proporciona también la posibilidad de definir *arrays* de variables. En el ejemplo, se han definido las siguientes variables:

- La variable, `A`. Puede tomar un valor de el conjunto de valores posibles $\{v1, v2, v3\}$
- Un *array* de dos variables: `B[1]` y `B[2]`. Cada una puede tomar un valor del conjunto de valores posibles $\{$`v1`, `v2`$\}$. Se definen como un *array* de dos variables.
- Un *array* de tres variables: `C[1]`, `C[2]` y `C[3]`. Cada una puede tomar un valor del conjunto $\{$`true`, `false`$\}$. Se definen como un *array* de tres variables.
- Un *array* de dos variables, `D[1]`y `D[2]`, y una variable `E`. Cada una puede tomar un valor en el conjunto $\{$`w1`, `w2`, `w3`$\}$.

Definición de la base de conocimiento . Se refiere a la parte de código entre las palabras 'BEGIN_KB' y 'END_KB'. Definiremos aquí la base de conocimiento del sistema experto por medio de reglas y fórmulas atómicas.[1]

Fórmulas atómicas . En nuestro lenguaje las fórmulas atómicas (ver Definición 5.1) son descritas por medio de la expresión es con la forma $var1 = value$, '$var1 = var2$', '$var1 <> value$', '$var1 <> var2$', donde '$var1$' y '$var2$' son variables, '$value$'

[1] De acuerdo con la Proposición 5.1, se puede describir la base de conocimiento de un sistema experto representado por el modelo 'Concepto-Atributo-Valor' a través de reglas y fórmulas atómicas

es un posible valor de la variable y el símbolo '<>' denota el símbolo '$\neq$' el cual puede ser usado en las fórmulas atómica. En el ejemplo anterior, nosotros hemos considerado las fórmulas atómicas:

$$\texttt{C[1]} \neq \texttt{C[3]}$$

Sólo consideramos fórmulas atómicas del tipo '$var1 = var2$' o '$var1 \neq var2$' si las variables '$var1$' y '$var2$' están declaradas en la misma línea. De esta manera, no consideramos como correcta la fórmulas atómicas 'A= `B[1]`' ya que están declaradas en distintas líneas. De la misma manera, consideraríamos como correcta la formula atómica '`D[1]=E`'.

Reglas . En nuestro lenguaje, una regla $A_1 \wedge ... \wedge A_r \rightarrow B_1 \vee ... \vee B_s$ (donde $A_1, ..., A_r, B_1, ..., B_s$ son fórmulas atómicas) se declara a través del siguiente código:

$$A_1 \texttt{ AND } A_2 \texttt{ AND} \ldots \texttt{AND } A_r \texttt{ -> } B_1 \texttt{ OR } B_2 \texttt{ OR} \ldots \texttt{OR } B_s$$

donde A_1,...A_r, B_1,..., B_s son fórmulas atómicas especificadas a través de nuestro lenguaje.

En el ejemplo anterior, hemos considerado las siguientes reglas:

$$(\texttt{A} = \texttt{v1}) \wedge (\texttt{B[1]} \neq \texttt{v2}) \wedge (\texttt{B[1]} = \texttt{B[2]}) \rightarrow (\texttt{C[1]} = \texttt{C[2]}) \vee (\texttt{C[3]} = \texttt{true})$$

$$(\texttt{A} \neq \texttt{v2}) \wedge (\texttt{B[1]} = \texttt{B[2]}) \rightarrow (\texttt{C[1]} = \texttt{true})$$

6.2. Traducción al modelo lógico proposicional Booleano

6.2.1. Traducción en CAS

Un sistema experto especificado en nuestro lenguaje (llamado sistema experto original en este apartado) puede ser traducido a un sistema experto equivalente especificado en lógica proposicional Booleana (llamado sistema experto traducido en este apartado) la cual esta descrita en el apartado 4.

A continuación, se tendrán en cuenta:

Variables El sistema experto traducido contiene sólo variables Booleanas. Cada variable Booleana en el sistema experto traducido indica si una variable especificada en el sistema experto original toma un valor determinado.

En el ejemplo anterior, definimos las variables Booleanas X_1, X_2 en el sistema experto traducido indicando si la variable `A` en el sistema experto original toma respectivamente el valor `v1` o el valor `v2`. Definir otro valor Booleano para el caso de que la variable `A` tome el valor `v3` no es necesario debido a que la información puede ser deducida de las variables X_1 y X_2. De hecho, cuando una fórmula Booleana $(\neg X_1 \wedge \neg X_2)$ se cumple, la variable `A` ni toma el valor `v1` ni el valor `v2`, y consecuentemente `A` toma el valor `v3`. En resumen, tenemos que:

X_1: La variable `A` toma el valor `v1`.

X_2: La variable `A` toma el valor `v2`.

$(\neg X_1 \wedge \neg X_2)$: La variable `A` toma el valor `v3`.

Del mismo modo, definiremos diferentes variables Booleanas para el resto de variables y el resto de valores del sistema experto original.

Con el fin de definir estas variables Booleanas, consideraremos las siguientes funciones:

- $n(Z)$ donde Z es una variable. Indica el número de valores que puede tomar esa variable.

- $\psi(Z, v)$, donde Z y v son respectivamente una variable y un valor definidos en el sistema experto original. Es un número natural entre 0 y $m-1$ donde m es el número de variables Booleanas definidas en el sistema experto traducido. Un valor $a = \psi(Z, v) \geq 0$ informa de que la variable Booleana X_a indica si la variable Z toma el valor v. Un valor $a = \psi(Z, v) = -1$ informa de que no hay variables Booleanas que indiquen si la variable Z toma el valor v. La función ψ se define de tal forma que para cualquier *array* de variables, W, se cumple lo siguiente:

$$\psi(W[i], v) = \psi(W[1], v) + (i-1) \cdot (n(W[1]) - 1)$$

.

En el ejemplo anterior, necesitamos definir en el sistema experto traducido las variables Booleanas $\{X_1, ..., X_7\}$. La función ψ se define como sigue:

$\psi(\texttt{A}, \texttt{v1}) = 0;$	$\psi(\texttt{A}, \texttt{v2}) = 1;$	$\psi(\texttt{A}, \texttt{v3}) = -1;$
$\psi(\texttt{B[1]}, \texttt{v1}) = 2;$	$\psi(\texttt{B[1]}, \texttt{v2}) = -1;$	
$\psi(\texttt{B[2]}, \texttt{v1}) = 3;$	$\psi(\texttt{B[2]}, \texttt{v2}) = -1;$	
$\psi(\texttt{C[1]}, \texttt{true}) = 4;$	$\psi(\texttt{C[1]}, \texttt{false}) = -1;$	
$\psi(\texttt{C[2]}, \texttt{true}) = 5;$	$\psi(\texttt{C[2]}, \texttt{false}) = -1;$	
$\psi(\texttt{C[3]}, \texttt{true}) = 6;$	$\psi(\texttt{C[3]}, \texttt{false}) = -1;$	
$\psi(\texttt{D[1]}, \texttt{w1}) = 7;$	$\psi(\texttt{D[1]}, \texttt{w2}) = 8;$	$\psi(\texttt{D[1]}, \texttt{w3}) = -1;$
$\psi(\texttt{D[2]}, \texttt{w1}) = 9;$	$\psi(\texttt{D[2]}, \texttt{w2}) = 10;$	$\psi(\texttt{D[2]}, \texttt{w3}) = -1;$
$\psi(\texttt{E}, \texttt{w1}) = 11$	$\psi(\texttt{E}, \texttt{w2}) = 12;$	$\psi(\texttt{E}, \texttt{w3}) = -1;$

- $\tau_1(Z, v)$ y $\tau_2(Z, v)$, donde Z y v son respectivamente una variable y un valor definidos en el sistema experto original. Son números naturales entre 0 y el número de variables Booleanas menos 1. Cuando $a = \psi(Z, v) = -1$, los valores $b_1 = \tau_1(Z, v)$ y $b_2 = \tau_2(Z, v)$ indican que se puede detectar si la variable Z toma

el valor v a través de la fórmula Booleana:

$$\neg X_{b_1} \wedge \neg X_{b_1+1} \wedge ... \wedge \neg X_{b_2-1} \wedge \neg X_{b_2}$$

En el ejemplo anterior, tenemos que se definen las funciones τ_1 and τ_2 de la siguiente manera:

$\tau_1(\texttt{A}, \texttt{v3}) = 0;$	$\tau_2(\texttt{A}, \texttt{v3}) = 1;$
$\tau_1(\texttt{B[1]}, \texttt{v2}) = 2;$	$\tau_2(\texttt{B[1]}, \texttt{v2}) = 2;$
$\tau_1(\texttt{B[2]}, \texttt{v2}) = 3;$	$\tau_2(\texttt{B[2]}, \texttt{v2}) = 3;$
$\tau_1(\texttt{C[1]}, \texttt{false}) = 4;$	$\tau_2(\texttt{C[1]}, \texttt{false}) = 4;$
$\tau_1(\texttt{C[2]}, \texttt{false}) = 5;$	$\tau_2(\texttt{C[2]}, \texttt{false}) = 5;$
$\tau_1(\texttt{C[3]}, \texttt{false}) = 6;$	$\tau_2(\texttt{C[3]}, \texttt{false}) = 6;$
$\tau_1(\texttt{D[1]}, \texttt{false}) = 7;$	$\tau_2(\texttt{D[1]}, \texttt{false}) = 8;$
$\tau_1(\texttt{D[2]}, \texttt{false}) = 9;$	$\tau_2(\texttt{D[2]}, \texttt{false}) = 10;$
$\tau_1(\texttt{E}, \texttt{false}) = 11;$	$\tau_2(\texttt{E}, \texttt{false}) = 12;$

Fórmulas Se debe especificar cada fórmula en el sistema experto original a través de las variables Booleanas que acabamos de definir. Nosotros definimos una función Φ que lo efectúa.

Caso A es la fórmula atómica positiva $Z = v$.

Dado $a = \psi(Z, v)$. Dado $b_1 = \tau_1(Z, v)$ y $b_2 = \tau_2(Z, v)$

$$\Phi(A) = \begin{cases} X_a & \text{si } a > 0 \\ \neg X_{b_1} \wedge \neg X_{b_1+1} ... \wedge \neg X_{b_2-1} \wedge \neg X_{b_2} & \text{si } a = 0 \end{cases}$$

Caso A es la fórmula atómica positiva $Z = W$.

Sea $\{v_1, ..., v_n\}$ el conjunto de valores que pueden ser tomados por la variable Z

y la variable W. Tenemos que:

$$\Phi(A) = (\Phi(Z = v_1) \wedge \Phi(W = v_1)) \vee \ldots \vee (\Phi(Z = v_n) \wedge \Phi(W = v_n))$$

Caso A **es** $\neg B$ donde B es una fórmula.

$$\Phi(A) = \Phi(\neg B) = \neg\Phi(B)$$

Caso A **es** $B \wedge C$ donde B y C son fórmulas.

$$\Phi(A) = \Phi(B \wedge C) = \Phi(B) \wedge \Phi(C)$$

Caso A **es** $B \vee C$ donde B y C son fórmulas.

$$\Phi(A) = \Phi(B \vee C) = \Phi(B) \vee \Phi(C)$$

Caso A **es** $B \rightarrow C$ donde B y C son fórmulas.

$$\Phi(A) = \Phi(B \rightarrow C) = \Phi(B) \rightarrow \Phi(C)$$

Restricciones de integridad A parte de la traducción de las fórmulas especificadas en el sistema experto original en términos Booleanos, necesitamos definir restricciones de integridad para que el modelo proposicional Booleano pueda ser equivalente al descrito en nuestro lenguaje. En el ejemplo anterior, necesitamos especificar que la variable A no puede tomar los valores `v1` y `v2` al mismo tiempo. Es decir, necesitamos especificar la restricción de integridad $\neg(X_1 \wedge X_2)$.

De esta manera, por cada variable Z y cada dos variables v y w (especificados en

el sistema experto original) de tal manera que $a = \psi(Z, v) \geq 0$ y $b = \psi(Z, w) \geq 0$ necesitamos definir la siguiente restricción de integridad, la cual especifica que la variable Z no puede tomar los valores a y b al mismo tiempo:

$$\neg(X_a \wedge X_b)$$

Para el ejemplo anterior, sólo tenemos la siguiente restricción de integridad:

$$IC_1 \equiv \neg(X_1 \wedge X_2)$$

A continuación, especificaremos los pasos necesarios para implementar un sistema experto basado en lógica proposicional Booleana en un sistema de computación algebraica.

i) Definir las funciones ψ, n, τ_1 y τ_2. Sea m el número de variables Booleanas necesarias.

ii) Definir el anillo de polinomios $\mathbb{Z}_2[x_1, \ldots, x_m]$ y el ideal J (mirar apartado 5.2). Para el ejemplo anterior en sintaxis de Singular:

```
ring r=2,(x(0..12)),lp;


poly s0=(x(0)^2) + x(0);

poly s1=(x(1)^2) + x(1);

poly s2=(x(2)^2) + x(2);

poly s3=(x(3)^2) + x(3);

poly s4=(x(4)^2) + x(4);

poly s5=(x(5)^2) + x(5);

poly s6=(x(6)^2) + x(6);
```

```
poly s7=(x(7)^2) + x(7);

poly s8=(x(8)^2) + x(8);

poly s9=(x(9)^2) + x(9);

poly s10=(x(10)^2) + x(10);

poly s11=(x(11)^2) + x(11);

poly s11=(x(12)^2) + x(12);


ideal J=s0,s1,s2,s3,s4,s5,s6,s7,s8,s9,s10,s11,s12;

J = std(J);
```

iii) Definir las conectivas $\neg$ y $\wedge$, $\vee$, $\rightarrow$ de acuerdo a la definición 4.2. No es necesario definir las conectivas $\rightarrow$ y $\wedge$. En sintaxis de CoCoA:

```
proc NEG(poly M) {return(reduce(1+M,J));}

proc O(poly M,poly N) {return(reduce(M*N,J));}

proc Y(poly M,poly N) {return(reduce(M*N+M+N,J));}

proc IMP(poly M,poly N) {return(reduce(M*N+N,J));}
```

iv) Definir las restricciones de integridad necesarias para que el modelo proposicional Booleano pueda ser equivalente al especificado en nuestro lenguaje. Para el ejemplo anterior en sintaxis de CoCoA:

```
poly ic1=NEG(Y(x(0),x(1)));
```

v) Traducir las fórmulas especificadas en nuestro lenguaje en fórmulas Booleanas tal como hemos visto anteriormente. En el ejemplo anterior, en sintaxis de CoCoA:

```
poly r1:= NEG(O(Y(x(4),x(6)),Y(NEG(x(4)),NEG(x(6)))));

poly r2:= IMP(Y(Y(x(0),NEG(NEG(x(2)))),

            O(Y(x(2),x(3)),Y(NEG(x(2)),NEG(x(3))))),

          O(O(Y(x(4),x(5)),Y(NEG(x(4)),NEG(x(5)))),x(6)));

poly r3:= IMP(Y(NEG(x(1)),O(Y(x(2),x(3)),Y(NEG(x(2))

    ,NEG(x(3))))),x(4));
```

vi) Definir el ideal K como los polinomios asociados a las fórmulas traducido en términos Booleanos junto con los polinomios asociados a las restricciones de integridad. Para el ejemplo anterior en sintaxis de Singular:

```
ideal K=slimgb(IC1,R1,R2,R3);
```

6.2.2. Tabla de símbolos

Cuando el traductor genera un código para un sistema experto dado en el modelo lógico proposicional Booleano, genera un fichero que forma la tabla de símbolos de variables y valores. A partir de este fichero podemos traducir una variable y un valor desde el sistema experto original al sistema experto traducido.

A continuación se muestra el fichero correspondiente a la tabla anterior. Las columnas se distinguen por tabuladores y las filas por saltos de línea.

Cuadro 1: Tabla de variables. Fichero: ts_mlp_1.txt

Variable	Valor	$\psi(Variable, Valor)$	n	τ_1	τ_2
A	v1	0	3	-	-
A	v2	1	-	-	-
A	v3	-1	-	0	1
B	v1	2	2	-	-
B	v2	-1	-	2	2
C	true	4	2	-	-
C	false	-1	-	4	4
D	w1	7	3	-	-
D	w2	8	-	-	-
D	w3	-1	-	7	8
E	w1	11	3	-	-
E	w2	12	-	-	-
E	w3	-1	-	11	12

```
A       v1       0        3
A       v2       1
A       v3      -1                0        1
B       v1       2        2
B       v2      -1                2        2
C       true     4        2
C       false   -1                4        4
D       w1       7        3
D       w2       8
D       w3      -1                7        8
E       w1      11        3
E       w2      12
E       w3      -1               11       12
```

6.3. Traducción del modelo 'Concepto-Atributo-Valor'

6.3.1. Traducción en CAS

En este apartado, trataremos de generar las sentencias para implementar el sistema experto descrito en nuestro lenguaje en un sistema de computación algebraica de acuerdo con el enfoque algebraico del apartado 5.

Dado que nuestro lenguaje se basa en el formalismo de representación 'Concepto-Atributo-Valor', estas sentencias son fáciles de generar. Conllevan los siguientes pasos:

i) Calcular un primo p mayor o igual que el número de posibles valores que puede tomar cualquier variable. En el ejemplo previo, el número de posibles valores de cualquier variable es como máximo tres. En consecuencia, tenemos que $p = 3$.

ii) Codificaremos cada variable y cada valor en nuestro lenguaje con un número natural. En consecuencia, definiremos las siguientes funciones:

- $\psi(X)$ indica el código de la variable X. Dado un $array$ de variables, X, se cumple lo siguiente:

$$\psi(X[i]) = \psi(X[1]) + i - 1$$

- $\phi(X, v)$ indica el código del valor v que puede tomar la variable X. El código de cualquier valor es un número natural entre 0 y $p - 1$. Dados dos valores v, w de una misma variable X, tenemos que

$$\phi(X, v) \neq \phi(X, w)$$

En el ejemplo anterior, codificamos las variables de la siguiente manera:

$\psi(\texttt{A}) = 1$; $\psi(\texttt{B[1]}) = 2$; $\psi(\texttt{B[2]}) = 3$;

$\psi(\texttt{C[1]}) = 4$; $\psi(\texttt{C[2]}) = 5$; $\psi(\texttt{C[3]}) = 6$;

$\psi(\texttt{D[1]}) = 7$; $\psi(\texttt{D[2]}) = 8$;

$\psi(\texttt{E}) = 9$;

y codificamos los valores de la siguiente manera:

$\phi(\texttt{A,v1}) = 0$; $\phi(\texttt{A,v2}) = 1$; $\phi(\texttt{A,v3}) = 2$;

$\phi(\texttt{B,v1}) = 0$; $\phi(\texttt{B,v2}) = 1$;

$\phi(\texttt{C,true}) = 0$; $\phi(\texttt{C,false}) = 1$;

$\phi(\texttt{D,w1}) = 0$; $\phi(\texttt{D,w2}) = 1$; $\phi(\texttt{D,w3}) = 2$;

$\phi(\texttt{E,w1}) = 0$; $\phi(\texttt{E,w2}) = 1$; $\phi(\texttt{E,w3}) = 2$;

iii) Definir el anillo de polinomios $\mathbb{Z}_p[x_1, \ldots, x_m]$ (ver apartado 5.2) donde m es el número de variables especificadas en nuestro lenguaje. Para el ejemplo anterior en sintaxis de CoCoA:

```
ring r=3,(x(0..9)),lp;
```

iv) Definir cada polinomio s_i asociado a cada variable y al ideal J (ver apartado 5.2). Para el ejemplo anterior en sintaxis de Singular:

```
poly s0=(x(0)-2) * (x(0)-1) * (x(0)-0);
poly s1=(x(1)-1) * (x(1)-0);
poly s2=(x(2)-1) * (x(2)-0);
poly s3=(x(3)-1) * (x(3)-0);
poly s4=(x(4)-1) * (x(4)-0);
poly s5=(x(5)-1) * (x(5)-0);
poly s6=(x(6)-2) * (x(6)-1) * (x(6)-0);
```

```
poly s7=(x(7)-2) * (x(7)-1) * (x(7)-0);

poly s8=(x(8)-2) * (x(8)-1) * (x(8)-0);

poly s9=(x(9)-2) * (x(9)-1) * (x(9)-0);


ideal J=s0,s1,s2,s3,s4,s5,s6,s7,s8,s9;

J = std(J);
```

v) Necesitamos definir las conectivas $\neg$ y $\vee$ de acuerdo con la Definición 5.2. Las conectivas $\rightarrow$ o $\wedge$no son necesarias de ser definidas. En sintaxys de Singular:

```
proc NEG(poly M) {return(reduce(M^2-1,J));}

proc O(poly M,poly N) {return(reduce(M*N,J));}
```

vi) Necesitamos definir el ideal asociado a la base de conocimiento del sistema experto. De acuerdo con la Proposición 5.2, las reglas son definidas usando solo las conectivas $\neg$ y $\vee$.Para el ejemplo anterior en sintaxis de Singular:

```
poly r0=NEG(x(2)-x(4));

poly r1=O(NEG((x(0)-0)),O(O((x(1)-1),NEG(x(1)-x(2)))
      ,O(x(3)-x(4),(x(5)-0))));

poly r2=O((x(0)-1),O(NEG(x(1)-x(2)),(x(3)-0)));


ideal K=r0,r1,r2;

ideal BASE=slimgb(J+K);
```

6.3.2. Tabla de símbolos

Cuando el traductor genera un código para un sistema experto dado en el modelo 'Concepto-Atributo-Valor', genera dos ficheros que forman la tabla de símbolos de variables y valores. A partir de estos dos ficheros podemos traducir una variable y un valor desde el sistema experto original al sistema experto traducido.

En la tabla de variables definimos el código que tiene asociado cada variable.

Cuadro 2: Tabla de variables. Fichero: ts_mcav_1.txt

Nombre variable	Código Variable
a	0
b	1
c	3

En la tabla de valores definimos el código que tiene asociado cada valor de cada variable.

Cuadro 3: Tabla de valores. Fichero: ts_mcav_2.txt

Nombre valor	Código Variable	Indice
v1	0	0
v2	0	1
v3	0	2
v1	1	0
v2	1	1
true	3	0
false	3	1

7. Ejemplo de Sistema Experto

En este apartado desarrollaremos un ejemplo de sistema experto real. Se trata de un sistema experto que recomienda el mejor tratamiento que un paciente con cáncer de colon en función de los resultados de unas pruebas clínicas al que el paciente previamente se ha sometido. Este sistema experto se va a desarrollar en el lenguaje descrito en este trabajo final de carrera. A continuación, detallamos el sistema experto.

7.1. Especificación

Algunos términos utilizados en el lenguaje del sistema experto.

- Tratamiento : Tratamiento que se debe aplicar al paciente. Consideramos los siguientes tratamientos definidos en [19].
    - Quimioterapia: Quimioterapia para la enfermedad avanzada o metastásica (ver pág. 25 en [19])
    - Tratamiento 3: Tratamiento y terapia adyuvante definida en (ver pág. 9 en [19])
    - Tratamiento 6: Tratamiento y terapia adyuvante definida en (ver pág. 12 en [19])
    - Tratamiento 7: Tratamiento y terapia adyuvante definida en (ver pág. 13 en [19])]
    - Tratamiento 8: Tratamiento primario definido en (ver pág. 14 en [19])
- Cirugia : En caso de que haya que someter con cirugía al paciente, consideraremos los siguientes tipos:
    - Supervision: No se practica cirugía al paciente, se recomienda la observación y seguimiento del mismo
    - Bloc removal.
    - One stage bloc removal.
    - Reserction with diversion.

- Stent.
- Diversion.

- Pruebas. En ocasiones es recomendable que el paciente se someta a nuevas pruebas. Consideraremos las siguientes:
  - Prueba 7.1 (ver pág. 7 en [19]). Consiste en efectuar lo siguiente:
    - Pathology review
    - Colonoscopy
    - Marking of cancerous polyp site

    El resultado de efectuar la prueba 7.1 es una de las siguientes posibilidades:
    - single specimen
    - fragmented specimen
    - negativo
    - desconocido
  - Prueba 7.2 (ver pág. 7 en [19]). Consiste en efectuar lo siguiente:
    - Pathology review
    - Colonoscopy
    - Marking of cancerous polyp site

    El resultado de efectuar la prueba 7.2 es una de las siguientes posibilidades:
    - single specimen
    - fragmented specimen
    - negativo
    - desconocido
  - Prueba 8 (ver pág. 8 en [19]). Consiste en efectuar lo siguiente:
    - Pathology review

- Colonoscopy
- CBC,platelets, chemistry profile
- CEA
- Chest/abdominal/pelvit CT
- PET-CT scan is not routinely indicated.

El resultado de efectuar la prueba 8 es una de las siguientes posibilidades:

- resecable obstruyendo
- resecable no obstruyendo
- localmente no resecable
- medicamente inoperable
- desconocido

- Prueba 15.1 (ver pág. 15 en [19]). Consiste en efectuar lo siguiente:
  - Physical exam
  - Colonoscopy
  - Chest/abdominal/pelvit CT
  - consider PET-CT scan

  El resultado de efectuar la prueba 15.1 es una de las siguientes posibilidades:
  - si
  - no
  - desconocido
- Prueba 15.2 (ver pág. 15 en [19]). Consiste en efectuar lo siguiente:
  - Consider PET-CT scan
  - Rreevaluate chest/abdominal/pelvit CT in 3 mo

  El resultado de efectuar la prueba 15.2 es una de las siguientes posibilidades:
  - si

- no
- desconocido

- Prueba 11 (ver pág. 11 en [19]). Consiste en efectuar lo siguiente:
  - Colonoscopy
  - Chest/addominal/pelvic CT
  - CBC,plateltes,chemistry profile
  - CEA
  - Determination of tumor KRAS genestatus
  - Needle biopsy, if clinically indicated
  - PET-CT scan only if potentially surgically curable M1 disease
  - Multidisciplinary team evaluation, including a surgeon experienced in the resection of hepatobiliary and lung metastases.

  El resultado de efectuar la prueba 11 es una de las siguientes posibilidades:
  - sincronos liver resecable
  - sincronos liver no resecable
  - sincronos abdominal
  - desconocido
- Prueba 13 (ver pág. 15 en [19]). Consiste en efectuar lo siguiente:
  - Reevaluate for conversion to resectable every 2 mo if conversion to resectability is a reasonable goal

  El resultado de efectuar la prueba 13 es una de las siguientes posibilidades:
  - convertida a resecable
  - restos no resecables
  - desconocido

- Presentacion Clinica (ver pág. 3 en [19]). El paciente puede presentar los siguientes síntomas:
  - Colon cancer: Colon cancer appropiate for resection.
  - Suspect: Suspected or proven metastatic adenocarcicoma from the large bowel.
  - Pendunculated poly: Pendunculated polyp (adenoma [tubular, tubulovillous or villous]) with invasive cancer.
  - Sessile polyp: Sessile polyp (adenoma [tubular, tubulovillous or villous]) with invasive cancer.

7.2. Especificación en lenguaje fuente

```
1  BEGIN
2
3   BEGIN_VARS
4
5    (*OUTPUTS*)
6    tratamiento      : quimioterapia , tratamiento_3 , tratamiento_6 ,
         tratamiento_7 , tratamiento_8 ;
7    cirugia          : supervision , con_bloc_removal , one_stage_bloc_removal ,
         reserction_with_diversion , stent , diversion ;
8    hacer_prueba     : prueba7_1 , prueba7_2 , prueba_8 , prueba_15_1 ,
         prueba_15_2 , prueba_11 , prueba_15 , prueba_13 , rehacer_prueba_15_2 ;
9
10
11   (* INPUTS *)
```

```
12  presentacion_clinica   : cancer_colon , sospechosa , pendunculated_polyp
        , sessile_polyp ;
13  v_prueba7_1      : single_specimen , fragmented_specimen , negativo ,
        desconocido ;
14  v_prueba7_2      : single_specimen , fragmented_specimen , negativo ,
        desconocido ;
15  v_prueba_8       : resecable_obstruyendo , resecable_no_obstruyendo ,
        localmente_no_resecable , medicamente_inoperable , desconocido ;
16  v_prueba_15_1     : si , no , desconocido ;
17  v_prueba_15_2     : si , no ;
18  v_prueba_11      : sincronos_liver_resecable ,
        sincronos_liver_no_resecable , sincronos_abdominal , desconocido ;
19  v_prueba_15      : sincronos_liver_resecable ,
        sincronos_liver_no_resecable , sincronos_abdominal , desconocido ;
20  v_prueba_13      : convertida_a_resecable , restos_no_resecables ,
        desconocido ;
21  recurrencia      : serial_cea , documented_metachronus_metastases , no ;
22  tratamiento_3_aplicado   : si , no ;
23  tratamiento_7_aplicado   : si , no ;
24  tratamiento_8_aplicado   : si , no ;
25
26  (* AUXILIARES *)
27  col_1       : si , no ;
28  col_2       : si , no ;
29  col_3       : si , no ;
```

```
30   col_5       : si,no;
31   col_6       : si,no;
32   col_7       : si,no;
33   col_8       : si,no;
34   col_9       : si,no;
35   col_10       : si,no;
36   col_c       : si,no;
37
38  END_VARS
39
40  BEGIN_KB
41
42   (*COL-1*)
43    (presentacion_clinica=pendunculated_polyp) AND (v_prueba7_1=
         desconocido) -> (hacer_prueba=prueba7_1);
44    (presentacion_clinica=pendunculated_polyp) AND (v_prueba7_1=
         single_specimen) -> (cirugia=supervision);
45    (presentacion_clinica=pendunculated_polyp) AND (v_prueba7_1=
         fragmented_specimen) -> (cirugia=con_bloc_removal);
46    (presentacion_clinica=sessile_polyp) AND (v_prueba7_2=desconocido)
          -> (hacer_prueba=prueba7_2);
47    (presentacion_clinica=sessile_polyp) AND (v_prueba7_2=
         single_specimen) -> (cirugia=supervision) OR (cirugia=
         con_bloc_removal);
```

```
48   (presentacion_clinica=sessile_polyp) AND (v_prueba7_2=
        fragmented_specimen) -> (cirugia=con_bloc_removal);
49   (presentacion_clinica=sessile_polyp)  ->  (col_3=si);
50   (presentacion_clinica=pendunculated_polyp) ->  (col_3=si);
51
52  (*COL-2*)
53   (presentacion_clinica=cancer_colon) AND (v_prueba_8=desconocido)
        -> (hacer_prueba=prueba_8);
54   (presentacion_clinica=cancer_colon) AND (v_prueba_8=
        resecable_no_obstruyendo) -> (cirugia=con_bloc_removal);
55   (presentacion_clinica=cancer_colon) AND (v_prueba_8=
        resecable_obstruyendo) -> (cirugia=one_stage_bloc_removal) OR (
        cirugia=reserction_with_diversion) OR (cirugia=diversion);
56   (presentacion_clinica=cancer_colon) AND (v_prueba_8=
        localmente_no_resecable) -> (cirugia=one_stage_bloc_removal);
57   (presentacion_clinica=cancer_colon) AND (v_prueba_8=
        localmente_no_resecable) -> (col_c=si);
58   (v_prueba_8=medicamente_inoperable) -> (col_c=si);
59   (presentacion_clinica=cancer_colon) AND (v_prueba_8=
        resecable_no_obstruyendo) -> (col_3=si);
60   (v_prueba_8=resecable_obstruyendo) -> (col_3=si);
61   (presentacion_clinica=sospechosa) -> (col_5=si);
62
63  (*COL_5*)
```

```
64   (presentacion_clinica=sospechosa) AND (v_prueba_11=desconocido) ->
         (hacer_prueba=prueba_11);
65   (presentacion_clinica=sospechosa) AND (v_prueba_11=
         sincronos_liver_resecable) -> (col_6=si);
66   (presentacion_clinica=sospechosa) AND (v_prueba_11=
         sincronos_liver_no_resecable) -> (col_7=si);
67   (presentacion_clinica=sospechosa) AND (v_prueba_11=
         sincronos_abdominal) -> (col_8=si);
68
69   (*COL_3*)
70   (col_3=si) AND (tratamiento_3_aplicado=no) -> (tratamiento=
         tratamiento_3);
71   (col_3=si) AND (tratamiento_3_aplicado=si) -> (col_9=si);
72
73   (*COL_6*)
74   (col_6=si) -> (col_9=si);
75
76   (*COL_7*)
77   (col_7=si) AND (v_prueba_13=desconocido) -> (hacer_prueba=
         prueba_13);
78   (col_7=si) AND (v_prueba_13=convertida_a_resecable) AND (
         tratamiento_7_aplicado=no) -> (tratamiento=tratamiento_7);
79   (col_7=si) AND (v_prueba_13=restos_no_resecables) -> (tratamiento=
         quimioterapia);
80
```

```
81   (*COL_8*)
82    (col_8=si) -> (col_c=si);
83   (*COL_9*)
84    (col_9=si) AND (recurrencia=serial_cea) AND (v_prueba_15_1=
         desconocido) -> (hacer_prueba=prueba_15_1);
85    (col_9=si) AND (recurrencia=serial_cea) AND (v_prueba_15_1=no) ->
         (hacer_prueba=prueba_15_2);
86    (col_9=si) AND (recurrencia=serial_cea) AND (v_prueba_15_1=no) AND
          (v_prueba_15_2=no) -> (hacer_prueba=rehacer_prueba_15_2);
87    (col_9=si) AND (recurrencia=serial_cea) AND (v_prueba_15_1=no) AND
          (v_prueba_15_2=no) AND (v_prueba_15_2=si) -> (col_10=si);
88    (col_9=si) AND (recurrencia=serial_cea) AND (v_prueba_15_1=si) ->
         (col_10=si);
89    (col_9=si) AND (recurrencia=documented_metachronus_metastases) ->
         (col_10=si);
90
91   (*COL_10*)
92    (col_10=si) -> (col_c=si);
93   (*COL_C*)
94    (col_c=si) -> (tratamiento=quimioterapia);
95
96  END_KB
97
98 END.
```

7.3. Resultados en el modelo lógico proposicional Booleano

Una vez descrito el sistema experto en nuestro lenguaje podemos obtener un sistema experto basado en lógica proposicional Booleana para los sistemas de computación algebraica: CoCoA, PolyBoRi o Singular.

7.3.1. Traducción en PolyBoRi

```
from polybori.PyPolyBoRi import *
from polybori.gbcore import *
from polybori import *

def O(M,N): return Polynomial(M)*Polynomial(N)
def NEG(M): return 1+Polynomial(M)
def Y(M,N): return NEG(O(NEG(M),NEG(N)))

declare_ring([Block("x",56)],globals());

s0=pow(x(0),2) + x(0);
s1=pow(x(1),2) + x(1);
s2=pow(x(2),2) + x(2);
s3=pow(x(3),2) + x(3);
s4=pow(x(4),2) + x(4);
s5=pow(x(5),2) + x(5);
s6=pow(x(6),2) + x(6);
s7=pow(x(7),2) + x(7);
```

```
19 s8=pow(x(8),2) + x(8);
20 s9=pow(x(9),2) + x(9);
21 s10=pow(x(10),2) + x(10);
22 s11=pow(x(11),2) + x(11);
23 s12=pow(x(12),2) + x(12);
24 s13=pow(x(13),2) + x(13);
25 s14=pow(x(14),2) + x(14);
26 s15=pow(x(15),2) + x(15);
27 s16=pow(x(16),2) + x(16);
28 s17=pow(x(17),2) + x(17);
29 s18=pow(x(18),2) + x(18);
30 s19=pow(x(19),2) + x(19);
31 s20=pow(x(20),2) + x(20);
32 s21=pow(x(21),2) + x(21);
33 s22=pow(x(22),2) + x(22);
34 s23=pow(x(23),2) + x(23);
35 s24=pow(x(24),2) + x(24);
36 s25=pow(x(25),2) + x(25);
37 s26=pow(x(26),2) + x(26);
38 s27=pow(x(27),2) + x(27);
39 s28=pow(x(28),2) + x(28);
40 s29=pow(x(29),2) + x(29);
41 s30=pow(x(30),2) + x(30);
42 s31=pow(x(31),2) + x(31);
43 s32=pow(x(32),2) + x(32);
```

```
44 s33=pow(x(33),2) + x(33);
45 s34=pow(x(34),2) + x(34);
46 s35=pow(x(35),2) + x(35);
47 s36=pow(x(36),2) + x(36);
48 s37=pow(x(37),2) + x(37);
49 s38=pow(x(38),2) + x(38);
50 s39=pow(x(39),2) + x(39);
51 s40=pow(x(40),2) + x(40);
52 s41=pow(x(41),2) + x(41);
53 s42=pow(x(42),2) + x(42);
54 s43=pow(x(43),2) + x(43);
55 s44=pow(x(44),2) + x(44);
56 s45=pow(x(45),2) + x(45);
57 s46=pow(x(46),2) + x(46);
58 s47=pow(x(47),2) + x(47);
59 s48=pow(x(48),2) + x(48);
60 s49=pow(x(49),2) + x(49);
61 s50=pow(x(50),2) + x(50);
62 s51=pow(x(51),2) + x(51);
63 s52=pow(x(52),2) + x(52);
64 s53=pow(x(53),2) + x(53);
65 s54=pow(x(54),2) + x(54);
66 s55=pow(x(55),2) + x(55);
67
68 ic0=NEG(Y(x(0),x(1)));
```

```
69 ic1=NEG(Y(x(0),x(2)));
70 ic2=NEG(Y(x(0),x(3)));
71 ic3=NEG(Y(x(1),x(2)));
72 ic4=NEG(Y(x(1),x(3)));
73 ic5=NEG(Y(x(2),x(3)));
74 ic6=NEG(Y(x(4),x(5)));
75 ic7=NEG(Y(x(4),x(6)));
76 ic8=NEG(Y(x(4),x(7)));
77 ic9=NEG(Y(x(4),x(8)));
78 ic10=NEG(Y(x(5),x(6)));
79 ic11=NEG(Y(x(5),x(7)));
80 ic12=NEG(Y(x(5),x(8)));
81 ic13=NEG(Y(x(6),x(7)));
82 ic14=NEG(Y(x(6),x(8)));
83 ic15=NEG(Y(x(7),x(8)));
84 ic16=NEG(Y(x(9),x(10)));
85 ic17=NEG(Y(x(9),x(11)));
86 ic18=NEG(Y(x(9),x(12)));
87 ic19=NEG(Y(x(9),x(13)));
88 ic20=NEG(Y(x(9),x(14)));
89 ic21=NEG(Y(x(9),x(15)));
90 ic22=NEG(Y(x(9),x(16)));
91 ic23=NEG(Y(x(10),x(11)));
92 ic24=NEG(Y(x(10),x(12)));
93 ic25=NEG(Y(x(10),x(13)));
```

```
94 ic26=NEG(Y(x(10),x(14)));
95 ic27=NEG(Y(x(10),x(15)));
96 ic28=NEG(Y(x(10),x(16)));
97 ic29=NEG(Y(x(11),x(12)));
98 ic30=NEG(Y(x(11),x(13)));
99 ic31=NEG(Y(x(11),x(14)));
100 ic32=NEG(Y(x(11),x(15)));
101 ic33=NEG(Y(x(11),x(16)));
102 ic34=NEG(Y(x(12),x(13)));
103 ic35=NEG(Y(x(12),x(14)));
104 ic36=NEG(Y(x(12),x(15)));
105 ic37=NEG(Y(x(12),x(16)));
106 ic38=NEG(Y(x(13),x(14)));
107 ic39=NEG(Y(x(13),x(15)));
108 ic40=NEG(Y(x(13),x(16)));
109 ic41=NEG(Y(x(14),x(15)));
110 ic42=NEG(Y(x(14),x(16)));
111 ic43=NEG(Y(x(15),x(16)));
112 ic44=NEG(Y(x(17),x(18)));
113 ic45=NEG(Y(x(17),x(19)));
114 ic46=NEG(Y(x(18),x(19)));
115 ic47=NEG(Y(x(20),x(21)));
116 ic48=NEG(Y(x(20),x(22)));
117 ic49=NEG(Y(x(21),x(22)));
118 ic50=NEG(Y(x(23),x(24)));
```

```
119 ic51=NEG(Y(x(23),x(25)));
120 ic52=NEG(Y(x(24),x(25)));
121 ic53=NEG(Y(x(26),x(27)));
122 ic54=NEG(Y(x(26),x(28)));
123 ic55=NEG(Y(x(26),x(29)));
124 ic56=NEG(Y(x(27),x(28)));
125 ic57=NEG(Y(x(27),x(29)));
126 ic58=NEG(Y(x(28),x(29)));
127 ic59=NEG(Y(x(30),x(31)));
128 ic60=NEG(Y(x(33),x(34)));
129 ic61=NEG(Y(x(33),x(35)));
130 ic62=NEG(Y(x(34),x(35)));
131 ic63=NEG(Y(x(36),x(37)));
132 ic64=NEG(Y(x(36),x(38)));
133 ic65=NEG(Y(x(37),x(38)));
134 ic66=NEG(Y(x(39),x(40)));
135 ic67=NEG(Y(x(41),x(42)));
136
137 J=[s0,s1,s2,s3,s4,s5,s6,s7,s8,s9,s10,s11,s12,s13,s14,s15,s16,s17,s18,
       s19,s20,s21,s22,s23,s24,s25,s26,s27,s28,s29,s30,s31,s32,s33,s34,
       s35,s36,s37,s38,s39,s40,s41,s42,s43,s44,s45,s46,s47,s48,s49,s50,
       s51,s52,s53,s54,s55,ic0,ic1,ic2,ic3,ic4,ic5,ic6,ic7,ic8,ic9,ic10,
       ic11,ic12,ic13,ic14,ic15,ic16,ic17,ic18,ic19,ic20,ic21,ic22,ic23,
       ic24,ic25,ic26,ic27,ic28,ic29,ic30,ic31,ic32,ic33,ic34,ic35,ic36,
       ic37,ic38,ic39,ic40,ic41,ic42,ic43,ic44,ic45,ic46,ic47,ic48,ic49,
```

```
    ic50,ic51,ic52,ic53,ic54,ic55,ic56,ic57,ic58,ic59,ic60,ic61,ic62,
    ic63,ic64,ic65,ic66,ic67];

r0=O(NEG(x(19)),O(NEG(1+x(20)*x(21)*x(22)),x(9)));
r1=O(NEG(x(19)),O(NEG(x(20)),x(4)));
r2=O(NEG(x(19)),O(NEG(x(21)),x(5)));
r3=O(NEG(1+x(17)*x(18)*x(19)),O(NEG(1+x(23)*x(24)*x(25)),x(10)));
r4=O(NEG(1+x(17)*x(18)*x(19)),O(NEG(x(23)),O(x(4),x(5))));
r5=O(NEG(1+x(17)*x(18)*x(19)),O(NEG(x(24)),x(5)));
r6=O(NEG(1+x(17)*x(18)*x(19)),x(48));
r7=O(NEG(x(19)),x(48));
r8=O(NEG(x(17)),O(NEG(1+x(26)*x(27)*x(28)*x(29)),x(11)));
r9=O(NEG(x(17)),O(NEG(x(27)),x(5)));
r10=O(NEG(x(17)),O(NEG(x(26)),O(x(6),O(x(7),1+x(4)*x(5)*x(6)*x(7)*x
    (8)))));
r11=O(NEG(x(17)),O(NEG(x(28)),x(6)));
r12=O(NEG(x(17)),O(NEG(x(28)),x(55)));
r13=O(NEG(x(29)),x(55));
r14=O(NEG(x(17)),O(NEG(x(27)),x(48)));
r15=O(NEG(x(26)),x(48));
r16=O(NEG(x(18)),x(49));
r17=O(NEG(x(18)),O(NEG(1+x(33)*x(34)*x(35)),x(14)));
r18=O(NEG(x(18)),O(NEG(x(33)),x(50)));
r19=O(NEG(x(18)),O(NEG(x(34)),x(51)));
r20=O(NEG(x(18)),O(NEG(x(35)),x(52)));
```

```
160 r21=O(NEG(x(48)),O(NEG(1+x(43)),x(1)));
161 r22=O(NEG(x(48)),O(NEG(x(43)),x(53)));
162 r23=O(NEG(x(50)),x(53));
163 r24=O(NEG(x(51)),O(NEG(1+x(39)*x(40)),x(16)));
164 r25=O(NEG(x(51)),O(O(NEG(x(39)),NEG(1+x(44))),x(3)));
165 r26=O(NEG(x(51)),O(NEG(x(40)),x(0)));
166 r27=O(NEG(x(52)),x(55));
167 r28=O(NEG(x(53)),O(O(NEG(x(41)),NEG(1+x(30)*x(31))),x(12)));
168 r29=O(NEG(x(53)),O(O(NEG(x(41)),NEG(x(31))),x(13)));
169 r30=O(NEG(x(53)),O(O(NEG(x(41)),O(NEG(x(31)),NEG(1+x(32)))),1+x(9)*x
        (10)*x(11)*x(12)*x(13)*x(14)*x(15)*x(16)));
170 r31=O(NEG(x(53)),O(O(NEG(x(41)),O(NEG(x(31)),O(NEG(1+x(32)),NEG(x(32)
        ))))),x(54)));
171 r32=O(NEG(x(53)),O(O(NEG(x(41)),NEG(x(30))),x(54)));
172 r33=O(NEG(x(53)),O(NEG(x(42)),x(54)));
173 r34=O(NEG(x(54)),x(55));
174 r35=O(NEG(x(55)),x(0));
175
176 K=[r0,r1,r2,r3,r4,r5,r6,r7,r8,r9,r10,r11,r12,r13,r14,r15,r16,r17,r18,
        r19,r20,r21,r22,r23,r24,r25,r26,r27,r28,r29,r30,r31,r32,r33,r34,
        r35];
177
178
179 gb=groebner_basis(J+K,heuristic=False);
```

7.3.2. Traducción en Singular

```
1  ring r=2,(x(0..55)),lp;
2
3  poly s0=(x(0)^2) + x(0);
4  poly s1=(x(1)^2) + x(1);
5  poly s2=(x(2)^2) + x(2);
6  poly s3=(x(3)^2) + x(3);
7  poly s4=(x(4)^2) + x(4);
8  poly s5=(x(5)^2) + x(5);
9  poly s6=(x(6)^2) + x(6);
10 poly s7=(x(7)^2) + x(7);
11 poly s8=(x(8)^2) + x(8);
12 poly s9=(x(9)^2) + x(9);
13 poly s10=(x(10)^2) + x(10);
14 poly s11=(x(11)^2) + x(11);
15 poly s12=(x(12)^2) + x(12);
16 poly s13=(x(13)^2) + x(13);
17 poly s14=(x(14)^2) + x(14);
18 poly s15=(x(15)^2) + x(15);
19 poly s16=(x(16)^2) + x(16);
20 poly s17=(x(17)^2) + x(17);
21 poly s18=(x(18)^2) + x(18);
22 poly s19=(x(19)^2) + x(19);
23 poly s20=(x(20)^2) + x(20);
```

```
poly s21=(x(21)^2) + x(21);
poly s22=(x(22)^2) + x(22);
poly s23=(x(23)^2) + x(23);
poly s24=(x(24)^2) + x(24);
poly s25=(x(25)^2) + x(25);
poly s26=(x(26)^2) + x(26);
poly s27=(x(27)^2) + x(27);
poly s28=(x(28)^2) + x(28);
poly s29=(x(29)^2) + x(29);
poly s30=(x(30)^2) + x(30);
poly s31=(x(31)^2) + x(31);
poly s32=(x(32)^2) + x(32);
poly s33=(x(33)^2) + x(33);
poly s34=(x(34)^2) + x(34);
poly s35=(x(35)^2) + x(35);
poly s36=(x(36)^2) + x(36);
poly s37=(x(37)^2) + x(37);
poly s38=(x(38)^2) + x(38);
poly s39=(x(39)^2) + x(39);
poly s40=(x(40)^2) + x(40);
poly s41=(x(41)^2) + x(41);
poly s42=(x(42)^2) + x(42);
poly s43=(x(43)^2) + x(43);
poly s44=(x(44)^2) + x(44);
poly s45=(x(45)^2) + x(45);
```

```
49 poly s46=(x(46)^2) + x(46);
50 poly s47=(x(47)^2) + x(47);
51 poly s48=(x(48)^2) + x(48);
52 poly s49=(x(49)^2) + x(49);
53 poly s50=(x(50)^2) + x(50);
54 poly s51=(x(51)^2) + x(51);
55 poly s52=(x(52)^2) + x(52);
56 poly s53=(x(53)^2) + x(53);
57 poly s54=(x(54)^2) + x(54);
58 poly s55=(x(55)^2) + x(55);
59
60 ideal J=s0,s1,s2,s3,s4,s5,s6,s7,s8,s9,s10,s11,s12,s13,s14,s15,s16,s17
     ,s18,s19,s20,s21,s22,s23,s24,s25,s26,s27,s28,s29,s30,s31,s32,s33,
     s34,s35,s36,s37,s38,s39,s40,s41,s42,s43,s44,s45,s46,s47,s48,s49,
     s50,s51,s52,s53,s54,s55;
61 J = std(J);
62
63 proc NEG(poly M) {return(reduce(1+M,J));}
64 proc O(poly M,poly N) {return(reduce(M*N,J));}
65 proc Y(poly M,poly N) {return(NEG(O(NEG(M),NEG(N))));}
66
67 poly ic0=NEG(Y(x(0),x(1)));
68 poly ic1=NEG(Y(x(0),x(2)));
69 poly ic2=NEG(Y(x(0),x(3)));
70 poly ic3=NEG(Y(x(1),x(2)));
```

```
71 poly ic4=NEG(Y(x(1),x(3)));
72 poly ic5=NEG(Y(x(2),x(3)));
73 poly ic6=NEG(Y(x(4),x(5)));
74 poly ic7=NEG(Y(x(4),x(6)));
75 poly ic8=NEG(Y(x(4),x(7)));
76 poly ic9=NEG(Y(x(4),x(8)));
77 poly ic10=NEG(Y(x(5),x(6)));
78 poly ic11=NEG(Y(x(5),x(7)));
79 poly ic12=NEG(Y(x(5),x(8)));
80 poly ic13=NEG(Y(x(6),x(7)));
81 poly ic14=NEG(Y(x(6),x(8)));
82 poly ic15=NEG(Y(x(7),x(8)));
83 poly ic16=NEG(Y(x(9),x(10)));
84 poly ic17=NEG(Y(x(9),x(11)));
85 poly ic18=NEG(Y(x(9),x(12)));
86 poly ic19=NEG(Y(x(9),x(13)));
87 poly ic20=NEG(Y(x(9),x(14)));
88 poly ic21=NEG(Y(x(9),x(15)));
89 poly ic22=NEG(Y(x(9),x(16)));
90 poly ic23=NEG(Y(x(10),x(11)));
91 poly ic24=NEG(Y(x(10),x(12)));
92 poly ic25=NEG(Y(x(10),x(13)));
93 poly ic26=NEG(Y(x(10),x(14)));
94 poly ic27=NEG(Y(x(10),x(15)));
95 poly ic28=NEG(Y(x(10),x(16)));
```

```
poly ic29=NEG(Y(x(11),x(12)));
poly ic30=NEG(Y(x(11),x(13)));
poly ic31=NEG(Y(x(11),x(14)));
poly ic32=NEG(Y(x(11),x(15)));
poly ic33=NEG(Y(x(11),x(16)));
poly ic34=NEG(Y(x(12),x(13)));
poly ic35=NEG(Y(x(12),x(14)));
poly ic36=NEG(Y(x(12),x(15)));
poly ic37=NEG(Y(x(12),x(16)));
poly ic38=NEG(Y(x(13),x(14)));
poly ic39=NEG(Y(x(13),x(15)));
poly ic40=NEG(Y(x(13),x(16)));
poly ic41=NEG(Y(x(14),x(15)));
poly ic42=NEG(Y(x(14),x(16)));
poly ic43=NEG(Y(x(15),x(16)));
poly ic44=NEG(Y(x(17),x(18)));
poly ic45=NEG(Y(x(17),x(19)));
poly ic46=NEG(Y(x(18),x(19)));
poly ic47=NEG(Y(x(20),x(21)));
poly ic48=NEG(Y(x(20),x(22)));
poly ic49=NEG(Y(x(21),x(22)));
poly ic50=NEG(Y(x(23),x(24)));
poly ic51=NEG(Y(x(23),x(25)));
poly ic52=NEG(Y(x(24),x(25)));
poly ic53=NEG(Y(x(26),x(27)));
```

```
121 poly ic54=NEG(Y(x(26),x(28)));
122 poly ic55=NEG(Y(x(26),x(29)));
123 poly ic56=NEG(Y(x(27),x(28)));
124 poly ic57=NEG(Y(x(27),x(29)));
125 poly ic58=NEG(Y(x(28),x(29)));
126 poly ic59=NEG(Y(x(30),x(31)));
127 poly ic60=NEG(Y(x(33),x(34)));
128 poly ic61=NEG(Y(x(33),x(35)));
129 poly ic62=NEG(Y(x(34),x(35)));
130 poly ic63=NEG(Y(x(36),x(37)));
131 poly ic64=NEG(Y(x(36),x(38)));
132 poly ic65=NEG(Y(x(37),x(38)));
133 poly ic66=NEG(Y(x(39),x(40)));
134 poly ic67=NEG(Y(x(41),x(42)));
135
136 poly r0=O(NEG(x(19)),O(NEG(1+x(20)*x(21)*x(22)),x(9)));
137 poly r1=O(NEG(x(19)),O(NEG(x(20)),x(4)));
138 poly r2=O(NEG(x(19)),O(NEG(x(21)),x(5)));
139 poly r3=O(NEG(1+x(17)*x(18)*x(19)),O(NEG(1+x(23)*x(24)*x(25)),x(10)))
      ;
140 poly r4=O(NEG(1+x(17)*x(18)*x(19)),O(NEG(x(23)),O(x(4),x(5))));
141 poly r5=O(NEG(1+x(17)*x(18)*x(19)),O(NEG(x(24)),x(5)));
142 poly r6=O(NEG(1+x(17)*x(18)*x(19)),x(48));
143 poly r7=O(NEG(x(19)),x(48));
144 poly r8=O(NEG(x(17)),O(NEG(1+x(26)*x(27)*x(28)*x(29)),x(11)));
```

```
poly r9=O(NEG(x(17)),O(NEG(x(27)),x(5)));
poly r10=O(NEG(x(17)),O(NEG(x(26)),O(x(6),O(x(7),1+x(4)*x(5)*x(6)*x
    (7)*x(8)))));
poly r11=O(NEG(x(17)),O(NEG(x(28)),x(6)));
poly r12=O(NEG(x(17)),O(NEG(x(28)),x(55)));
poly r13=O(NEG(x(29)),x(55));
poly r14=O(NEG(x(17)),O(NEG(x(27)),x(48)));
poly r15=O(NEG(x(26)),x(48));
poly r16=O(NEG(x(18)),x(49));
poly r17=O(NEG(x(18)),O(NEG(1+x(33)*x(34)*x(35)),x(14)));
poly r18=O(NEG(x(18)),O(NEG(x(33)),x(50)));
poly r19=O(NEG(x(18)),O(NEG(x(34)),x(51)));
poly r20=O(NEG(x(18)),O(NEG(x(35)),x(52)));
poly r21=O(NEG(x(48)),O(NEG(1+x(43)),x(1)));
poly r22=O(NEG(x(48)),O(NEG(x(43)),x(53)));
poly r23=O(NEG(x(50)),x(53));
poly r24=O(NEG(x(51)),O(NEG(1+x(39)*x(40)),x(16)));
poly r25=O(NEG(x(51)),O(O(NEG(x(39)),NEG(1+x(44))),x(3)));
poly r26=O(NEG(x(51)),O(NEG(x(40)),x(0)));
poly r27=O(NEG(x(52)),x(55));
poly r28=O(NEG(x(53)),O(O(NEG(x(41)),NEG(1+x(30)*x(31))),x(12)));
poly r29=O(NEG(x(53)),O(O(NEG(x(41)),NEG(x(31))),x(13)));
poly r30=O(NEG(x(53)),O(O(NEG(x(41)),O(NEG(x(31)),NEG(1+x(32)))),1+x
    (9)*x(10)*x(11)*x(12)*x(13)*x(14)*x(15)*x(16)));
```

```
poly r31=O(NEG(x(53)),O(O(NEG(x(41)),O(NEG(x(31)),O(NEG(1+x(32)),NEG(
    x(32))))),x(54)));
poly r32=O(NEG(x(53)),O(O(NEG(x(41)),NEG(x(30))),x(54)));
poly r33=O(NEG(x(53)),O(NEG(x(42)),x(54)));
poly r34=O(NEG(x(54)),x(55));
poly r35=O(NEG(x(55)),x(0));
ideal K=ic0,ic1,ic2,ic3,ic4,ic5,ic6,ic7,ic8,ic9,ic10,ic11,ic12,ic13,
    ic14,ic15,ic16,ic17,ic18,ic19,ic20,ic21,ic22,ic23,ic24,ic25,ic26,
    ic27,ic28,ic29,ic30,ic31,ic32,ic33,ic34,ic35,ic36,ic37,ic38,ic39,
    ic40,ic41,ic42,ic43,ic44,ic45,ic46,ic47,ic48,ic49,ic50,ic51,ic52,
    ic53,ic54,ic55,ic56,ic57,ic58,ic59,ic60,ic61,ic62,ic63,ic64,ic65,
    ic66,ic67,r0,r1,r2,r3,r4,r5,r6,r7,r8,r9,r10,r11,r12,r13,r14,r15,
    r16,r17,r18,r19,r20,r21,r22,r23,r24,r25,r26,r27,r28,r29,r30,r31,
    r32,r33,r34,r35;

ideal BASE=slimgb(J+K);
```

7.3.3. Traducción en CoCoA

```
USE Z/(2)[x[0..55]];

S0:=(x[0]^2) + x[0];
S1:=(x[1]^2) + x[1];
S2:=(x[2]^2) + x[2];
S3:=(x[3]^2) + x[3];
```

```
7 S4:=(x[4]^2) + x[4];
8 S5:=(x[5]^2) + x[5];
9 S6:=(x[6]^2) + x[6];
10 S7:=(x[7]^2) + x[7];
11 S8:=(x[8]^2) + x[8];
12 S9:=(x[9]^2) + x[9];
13 S10:=(x[10]^2) + x[10];
14 S11:=(x[11]^2) + x[11];
15 S12:=(x[12]^2) + x[12];
16 S13:=(x[13]^2) + x[13];
17 S14:=(x[14]^2) + x[14];
18 S15:=(x[15]^2) + x[15];
19 S16:=(x[16]^2) + x[16];
20 S17:=(x[17]^2) + x[17];
21 S18:=(x[18]^2) + x[18];
22 S19:=(x[19]^2) + x[19];
23 S20:=(x[20]^2) + x[20];
24 S21:=(x[21]^2) + x[21];
25 S22:=(x[22]^2) + x[22];
26 S23:=(x[23]^2) + x[23];
27 S24:=(x[24]^2) + x[24];
28 S25:=(x[25]^2) + x[25];
29 S26:=(x[26]^2) + x[26];
30 S27:=(x[27]^2) + x[27];
31 S28:=(x[28]^2) + x[28];
```

```
S29:=(x[29]^2) + x[29];
S30:=(x[30]^2) + x[30];
S31:=(x[31]^2) + x[31];
S32:=(x[32]^2) + x[32];
S33:=(x[33]^2) + x[33];
S34:=(x[34]^2) + x[34];
S35:=(x[35]^2) + x[35];
S36:=(x[36]^2) + x[36];
S37:=(x[37]^2) + x[37];
S38:=(x[38]^2) + x[38];
S39:=(x[39]^2) + x[39];
S40:=(x[40]^2) + x[40];
S41:=(x[41]^2) + x[41];
S42:=(x[42]^2) + x[42];
S43:=(x[43]^2) + x[43];
S44:=(x[44]^2) + x[44];
S45:=(x[45]^2) + x[45];
S46:=(x[46]^2) + x[46];
S47:=(x[47]^2) + x[47];
S48:=(x[48]^2) + x[48];
S49:=(x[49]^2) + x[49];
S50:=(x[50]^2) + x[50];
S51:=(x[51]^2) + x[51];
S52:=(x[52]^2) + x[52];
S53:=(x[53]^2) + x[53];
```

```
57 S54:=(x[54]^2) + x[54];
58 S55:=(x[55]^2) + x[55];
59
60 MEMORY.J:=Ideal(S0,S1,S2,S3,S4,S5,S6,S7,S8,S9,S10,S11,S12,S13,S14,S15
       ,S16,S17,S18,S19,S20,S21,S22,S23,S24,S25,S26,S27,S28,S29,S30,S31,
       S32,S33,S34,S35,S36,S37,S38,S39,S40,S41,S42,S43,S44,S45,S46,S47,
       S48,S49,S50,S51,S52,S53,S54,S55);
61
62 Define NEG(M) RETURN NF(1+M,MEMORY.J); EndDefine;
63 Define O(M,N) RETURN NF(M*N,MEMORY.J); EndDefine;
64 Define Y(M,N) RETURN (NEG(O(NEG(M),NEG(N)))); EndDefine;
65
66 IC0:=NEG(Y(x[0],x[1]));
67 IC1:=NEG(Y(x[0],x[2]));
68 IC2:=NEG(Y(x[0],x[3]));
69 IC3:=NEG(Y(x[1],x[2]));
70 IC4:=NEG(Y(x[1],x[3]));
71 IC5:=NEG(Y(x[2],x[3]));
72 IC6:=NEG(Y(x[4],x[5]));
73 IC7:=NEG(Y(x[4],x[6]));
74 IC8:=NEG(Y(x[4],x[7]));
75 IC9:=NEG(Y(x[4],x[8]));
76 IC10:=NEG(Y(x[5],x[6]));
77 IC11:=NEG(Y(x[5],x[7]));
78 IC12:=NEG(Y(x[5],x[8]));
```

```
79 IC13:=NEG(Y(x[6],x[7]));
80 IC14:=NEG(Y(x[6],x[8]));
81 IC15:=NEG(Y(x[7],x[8]));
82 IC16:=NEG(Y(x[9],x[10]));
83 IC17:=NEG(Y(x[9],x[11]));
84 IC18:=NEG(Y(x[9],x[12]));
85 IC19:=NEG(Y(x[9],x[13]));
86 IC20:=NEG(Y(x[9],x[14]));
87 IC21:=NEG(Y(x[9],x[15]));
88 IC22:=NEG(Y(x[9],x[16]));
89 IC23:=NEG(Y(x[10],x[11]));
90 IC24:=NEG(Y(x[10],x[12]));
91 IC25:=NEG(Y(x[10],x[13]));
92 IC26:=NEG(Y(x[10],x[14]));
93 IC27:=NEG(Y(x[10],x[15]));
94 IC28:=NEG(Y(x[10],x[16]));
95 IC29:=NEG(Y(x[11],x[12]));
96 IC30:=NEG(Y(x[11],x[13]));
97 IC31:=NEG(Y(x[11],x[14]));
98 IC32:=NEG(Y(x[11],x[15]));
99 IC33:=NEG(Y(x[11],x[16]));
100 IC34:=NEG(Y(x[12],x[13]));
101 IC35:=NEG(Y(x[12],x[14]));
102 IC36:=NEG(Y(x[12],x[15]));
103 IC37:=NEG(Y(x[12],x[16]));
```

```
104 IC38:=NEG(Y(x[13],x[14]));
105 IC39:=NEG(Y(x[13],x[15]));
106 IC40:=NEG(Y(x[13],x[16]));
107 IC41:=NEG(Y(x[14],x[15]));
108 IC42:=NEG(Y(x[14],x[16]));
109 IC43:=NEG(Y(x[15],x[16]));
110 IC44:=NEG(Y(x[17],x[18]));
111 IC45:=NEG(Y(x[17],x[19]));
112 IC46:=NEG(Y(x[18],x[19]));
113 IC47:=NEG(Y(x[20],x[21]));
114 IC48:=NEG(Y(x[20],x[22]));
115 IC49:=NEG(Y(x[21],x[22]));
116 IC50:=NEG(Y(x[23],x[24]));
117 IC51:=NEG(Y(x[23],x[25]));
118 IC52:=NEG(Y(x[24],x[25]));
119 IC53:=NEG(Y(x[26],x[27]));
120 IC54:=NEG(Y(x[26],x[28]));
121 IC55:=NEG(Y(x[26],x[29]));
122 IC56:=NEG(Y(x[27],x[28]));
123 IC57:=NEG(Y(x[27],x[29]));
124 IC58:=NEG(Y(x[28],x[29]));
125 IC59:=NEG(Y(x[30],x[31]));
126 IC60:=NEG(Y(x[33],x[34]));
127 IC61:=NEG(Y(x[33],x[35]));
128 IC62:=NEG(Y(x[34],x[35]));
```

```
129 IC63:=NEG(Y(x[36],x[37]));
130 IC64:=NEG(Y(x[36],x[38]));
131 IC65:=NEG(Y(x[37],x[38]));
132 IC66:=NEG(Y(x[39],x[40]));
133 IC67:=NEG(Y(x[41],x[42]));
134
135 R0:=O(NEG(x[19]),O(NEG(1+x[20]*x[21]*x[22]),x[9]));
136 R1:=O(NEG(x[19]),O(NEG(x[20]),x[4]));
137 R2:=O(NEG(x[19]),O(NEG(x[21]),x[5]));
138 R3:=O(NEG(1+x[17]*x[18]*x[19]),O(NEG(1+x[23]*x[24]*x[25]),x[10]));
139 R4:=O(NEG(1+x[17]*x[18]*x[19]),O(NEG(x[23]),O(x[4],x[5])));
140 R5:=O(NEG(1+x[17]*x[18]*x[19]),O(NEG(x[24]),x[5]));
141 R6:=O(NEG(1+x[17]*x[18]*x[19]),x[48]);
142 R7:=O(NEG(x[19]),x[48]);
143 R8:=O(NEG(x[17]),O(NEG(1+x[26]*x[27]*x[28]*x[29]),x[11]));
144 R9:=O(NEG(x[17]),O(NEG(x[27]),x[5]));
145 R10:=O(NEG(x[17]),O(NEG(x[26]),O(x[6],O(x[7],1+x[4]*x[5]*x[6]*x[7]*x
        [8]))));
146 R11:=O(NEG(x[17]),O(NEG(x[28]),x[6]));
147 R12:=O(NEG(x[17]),O(NEG(x[28]),x[55]));
148 R13:=O(NEG(x[29]),x[55]);
149 R14:=O(NEG(x[17]),O(NEG(x[27]),x[48]));
150 R15:=O(NEG(x[26]),x[48]);
151 R16:=O(NEG(x[18]),x[49]);
152 R17:=O(NEG(x[18]),O(NEG(1+x[33]*x[34]*x[35]),x[14]));
```

```
153 R18:=O(NEG(x[18]),O(NEG(x[33]),x[50]));
154 R19:=O(NEG(x[18]),O(NEG(x[34]),x[51]));
155 R20:=O(NEG(x[18]),O(NEG(x[35]),x[52]));
156 R21:=O(NEG(x[48]),O(NEG(1+x[43]),x[1]));
157 R22:=O(NEG(x[48]),O(NEG(x[43]),x[53]));
158 R23:=O(NEG(x[50]),x[53]);
159 R24:=O(NEG(x[51]),O(NEG(1+x[39]*x[40]),x[16]));
160 R25:=O(NEG(x[51]),O(O(NEG(x[39]),NEG(1+x[44])),x[3]));
161 R26:=O(NEG(x[51]),O(NEG(x[40]),x[0]));
162 R27:=O(NEG(x[52]),x[55]);
163 R28:=O(NEG(x[53]),O(O(NEG(x[41]),NEG(1+x[30]*x[31])),x[12]));
164 R29:=O(NEG(x[53]),O(O(NEG(x[41]),NEG(x[31])),x[13]));
165 R30:=O(NEG(x[53]),O(O(NEG(x[41]),O(NEG(x[31]),NEG(1+x[32]))),1+x[9]*x
        [10]*x[11]*x[12]*x[13]*x[14]*x[15]*x[16]));
166 R31:=O(NEG(x[53]),O(O(NEG(x[41]),O(NEG(x[31]),O(NEG(1+x[32]),NEG(x
        [32])))),x[54]));
167 R32:=O(NEG(x[53]),O(O(NEG(x[41]),NEG(x[30])),x[54]));
168 R33:=O(NEG(x[53]),O(NEG(x[42]),x[54]));
169 R34:=O(NEG(x[54]),x[55]);
170 R35:=O(NEG(x[55]),x[0]);
171 K:=Ideal(IC0,IC1,IC2,IC3,IC4,IC5,IC6,IC7,IC8,IC9,IC10,IC11,IC12,IC13,
        IC14,IC15,IC16,IC17,IC18,IC19,IC20,IC21,IC22,IC23,IC24,IC25,IC26,
        IC27,IC28,IC29,IC30,IC31,IC32,IC33,IC34,IC35,IC36,IC37,IC38,IC39,
        IC40,IC41,IC42,IC43,IC44,IC45,IC46,IC47,IC48,IC49,IC50,IC51,IC52,
        IC53,IC54,IC55,IC56,IC57,IC58,IC59,IC60,IC61,IC62,IC63,IC64,IC65,
```

```
    IC66 , IC67 , R0 , R1 , R2 , R3 , R4 , R5 , R6 , R7 , R8 , R9 , R10 , R11 , R12 , R13 , R14 , R15 ,
    R16 , R17 , R18 , R19 , R20 , R21 , R22 , R23 , R24 , R25 , R26 , R27 , R28 , R29 , R30 , R31 ,
    R32 , R33 , R34 , R35 ) ;
172
173 GBasis(MEMORY. J+K) ;
```

7.4. Resultados en el modelo 'Concepto-Atributo-Valor'

Una vez descrito el sistema experto en nuestro lenguaje podemos obtener un sistema experto basado en el modelo 'Concepto-Atributo-Valor' para los sistemas de computación algebraica: CoCoA o Singular.

7.4.1. Traducción en Singular

```
1 ring r=11,(x(0..25)),lp;
2
3 poly s0=(x(0)-4) * (x(0)-3) * (x(0)-2) * (x(0)-1) * (x(0)-0);
4 poly s1=(x(1)-5) * (x(1)-4) * (x(1)-3) * (x(1)-2) * (x(1)-1) * (x(1)
    -0);
5 poly s2=(x(2)-8) * (x(2)-7) * (x(2)-6) * (x(2)-5) * (x(2)-4) * (x(2)
    -3) * (x(2)-2) * (x(2)-1) * (x(2)-0);
6 poly s3=(x(3)-3) * (x(3)-2) * (x(3)-1) * (x(3)-0);
7 poly s4=(x(4)-3) * (x(4)-2) * (x(4)-1) * (x(4)-0);
8 poly s5=(x(5)-3) * (x(5)-2) * (x(5)-1) * (x(5)-0);
9 poly s6=(x(6)-4) * (x(6)-3) * (x(6)-2) * (x(6)-1) * (x(6)-0);
10 poly s7=(x(7)-2) * (x(7)-1) * (x(7)-0);
```

```
poly s8=(x(8)-1) * (x(8)-0);
poly s9=(x(9)-3) * (x(9)-2) * (x(9)-1) * (x(9)-0);
poly s10=(x(10)-3) * (x(10)-2) * (x(10)-1) * (x(10)-0);
poly s11=(x(11)-2) * (x(11)-1) * (x(11)-0);
poly s12=(x(12)-2) * (x(12)-1) * (x(12)-0);
poly s13=(x(13)-1) * (x(13)-0);
poly s14=(x(14)-1) * (x(14)-0);
poly s15=(x(15)-1) * (x(15)-0);
poly s16=(x(16)-1) * (x(16)-0);
poly s17=(x(17)-1) * (x(17)-0);
poly s18=(x(18)-1) * (x(18)-0);
poly s19=(x(19)-1) * (x(19)-0);
poly s20=(x(20)-1) * (x(20)-0);
poly s21=(x(21)-1) * (x(21)-0);
poly s22=(x(22)-1) * (x(22)-0);
poly s23=(x(23)-1) * (x(23)-0);
poly s24=(x(24)-1) * (x(24)-0);
poly s25=(x(25)-1) * (x(25)-0);

ideal J=s0,s1,s2,s3,s4,s5,s6,s7,s8,s9,s10,s11,s12,s13,s14,s15,s16,s17
    ,s18,s19,s20,s21,s22,s23,s24,s25;
J = std(J);

proc NEG(poly M) {return(reduce(M^10-1,J));}
proc O(poly M,poly N) {return(reduce(M*N,J));}
```

```
35
36 poly r0=O(NEG((x(3)-2)),O(NEG((x(4)-3)),(x(2)-0)));
37 poly r1=O(NEG((x(3)-2)),O(NEG((x(4)-0)),(x(1)-0)));
38 poly r2=O(NEG((x(3)-2)),O(NEG((x(4)-1)),(x(1)-1)));
39 poly r3=O(NEG((x(3)-3)),O(NEG((x(5)-3)),(x(2)-1)));
40 poly r4=O(NEG((x(3)-3)),O(NEG((x(5)-0)),O((x(1)-0),(x(1)-1))));
41 poly r5=O(NEG((x(3)-3)),O(NEG((x(5)-1)),(x(1)-1)));
42 poly r6=O(NEG((x(3)-3)),(x(18)-0));
43 poly r7=O(NEG((x(3)-2)),(x(18)-0));
44 poly r8=O(NEG((x(3)-0)),O(NEG((x(6)-4)),(x(2)-2)));
45 poly r9=O(NEG((x(3)-0)),O(NEG((x(6)-1)),(x(1)-1)));
46 poly r10=O(NEG((x(3)-0)),O(NEG((x(6)-0)),O((x(1)-2),O((x(1)-3),(x(1)
       -5)))));
47 poly r11=O(NEG((x(3)-0)),O(NEG((x(6)-2)),(x(1)-2)));
48 poly r12=O(NEG((x(3)-0)),O(NEG((x(6)-2)),(x(25)-0)));
49 poly r13=O(NEG((x(6)-3)),(x(25)-0));
50 poly r14=O(NEG((x(3)-0)),O(NEG((x(6)-1)),(x(18)-0)));
51 poly r15=O(NEG((x(6)-0)),(x(18)-0));
52 poly r16=O(NEG((x(3)-1)),(x(19)-0));
53 poly r17=O(NEG((x(3)-1)),O(NEG((x(9)-3)),(x(2)-5)));
54 poly r18=O(NEG((x(3)-1)),O(NEG((x(9)-0)),(x(20)-0)));
55 poly r19=O(NEG((x(3)-1)),O(NEG((x(9)-1)),(x(21)-0)));
56 poly r20=O(NEG((x(3)-1)),O(NEG((x(9)-2)),(x(22)-0)));
57 poly r21=O(NEG((x(18)-0)),O(NEG((x(13)-1)),(x(0)-1)));
58 poly r22=O(NEG((x(18)-0)),O(NEG((x(13)-0)),(x(23)-0)));
```

```
59 poly r23=O(NEG((x(20)-0)),(x(23)-0));
60 poly r24=O(NEG((x(21)-0)),O(NEG((x(11)-2)),(x(2)-7)));
61 poly r25=O(NEG((x(21)-0)),O(O(NEG((x(11)-0)),NEG((x(14)-1))),(x(0)-3)
       ));
62 poly r26=O(NEG((x(21)-0)),O(NEG((x(11)-1)),(x(0)-0)));
63 poly r27=O(NEG((x(22)-0)),(x(25)-0));
64 poly r28=O(NEG((x(23)-0)),O(O(NEG((x(12)-0)),NEG((x(7)-2))),(x(2)-3))
       );
65 poly r29=O(NEG((x(23)-0)),O(O(NEG((x(12)-0)),NEG((x(7)-1))),(x(2)-4))
       );
66 poly r30=O(NEG((x(23)-0)),O(O(NEG((x(12)-0)),O(NEG((x(7)-1)),NEG((x
       (8)-1)))),(x(2)-8)));
67 poly r31=O(NEG((x(23)-0)),O(O(NEG((x(12)-0)),O(NEG((x(7)-1)),O(NEG((x
       (8)-1)),NEG((x(8)-0))))),(x(24)-0)));
68 poly r32=O(NEG((x(23)-0)),O(O(NEG((x(12)-0)),NEG((x(7)-0))),(x(24)-0)
       ));
69 poly r33=O(NEG((x(23)-0)),O(NEG((x(12)-1)),(x(24)-0)));
70 poly r34=O(NEG((x(24)-0)),(x(25)-0));
71 poly r35=O(NEG((x(25)-0)),(x(0)-0));
72
73 ideal K=r0,r1,r2,r3,r4,r5,r6,r7,r8,r9,r10,r11,r12,r13,r14,r15,r16,r17
       ,r18,r19,r20,r21,r22,r23,r24,r25,r26,r27,r28,r29,r30,r31,r32,r33,
       r34,r35;
74
75 ideal BASE=slimgb(J+K);
```

7.4.2. Traducción en CoCoA

```
1  USE Z/(11)[x[0..25]];
2
3  S0:=(x[0]-4) * (x[0]-3) * (x[0]-2) * (x[0]-1) * (x[0]-0);
4  S1:=(x[1]-5) * (x[1]-4) * (x[1]-3) * (x[1]-2) * (x[1]-1) * (x[1]-0);
5  S2:=(x[2]-8) * (x[2]-7) * (x[2]-6) * (x[2]-5) * (x[2]-4) * (x[2]-3) *
       (x[2]-2) * (x[2]-1) * (x[2]-0);
6  S3:=(x[3]-3) * (x[3]-2) * (x[3]-1) * (x[3]-0);
7  S4:=(x[4]-3) * (x[4]-2) * (x[4]-1) * (x[4]-0);
8  S5:=(x[5]-3) * (x[5]-2) * (x[5]-1) * (x[5]-0);
9  S6:=(x[6]-4) * (x[6]-3) * (x[6]-2) * (x[6]-1) * (x[6]-0);
10 S7:=(x[7]-2) * (x[7]-1) * (x[7]-0);
11 S8:=(x[8]-1) * (x[8]-0);
12 S9:=(x[9]-3) * (x[9]-2) * (x[9]-1) * (x[9]-0);
13 S10:=(x[10]-3) * (x[10]-2) * (x[10]-1) * (x[10]-0);
14 S11:=(x[11]-2) * (x[11]-1) * (x[11]-0);
15 S12:=(x[12]-2) * (x[12]-1) * (x[12]-0);
16 S13:=(x[13]-1) * (x[13]-0);
17 S14:=(x[14]-1) * (x[14]-0);
18 S15:=(x[15]-1) * (x[15]-0);
19 S16:=(x[16]-1) * (x[16]-0);
20 S17:=(x[17]-1) * (x[17]-0);
21 S18:=(x[18]-1) * (x[18]-0);
22 S19:=(x[19]-1) * (x[19]-0);
```

```
23 S20:=(x[20]-1) * (x[20]-0);
24 S21:=(x[21]-1) * (x[21]-0);
25 S22:=(x[22]-1) * (x[22]-0);
26 S23:=(x[23]-1) * (x[23]-0);
27 S24:=(x[24]-1) * (x[24]-0);
28 S25:=(x[25]-1) * (x[25]-0);
29
30
31 MEMORY.J:=Ideal(S0,S1,S2,S3,S4,S5,S6,S7,S8,S9,S10,S11,S12,S13,S14,S15
       ,S16,S17,S18,S19,S20,S21,S22,S23,S24,S25);
32
33 Define NEG(M) RETURN NF(M^10-1,MEMORY.J); EndDefine;
34 Define O(M,N) RETURN NF(M*N,MEMORY.J); EndDefine;
35
36 R0:=O(NEG((x[3]-2)),O(NEG((x[4]-3)),(x[2]-0)));
37 R1:=O(NEG((x[3]-2)),O(NEG((x[4]-0)),(x[1]-0)));
38 R2:=O(NEG((x[3]-2)),O(NEG((x[4]-1)),(x[1]-1)));
39 R3:=O(NEG((x[3]-3)),O(NEG((x[5]-3)),(x[2]-1)));
40 R4:=O(NEG((x[3]-3)),O(NEG((x[5]-0)),O((x[1]-0),(x[1]-1))));
41 R5:=O(NEG((x[3]-3)),O(NEG((x[5]-1)),(x[1]-1)));
42 R6:=O(NEG((x[3]-3)),(x[18]-0));
43 R7:=O(NEG((x[3]-2)),(x[18]-0));
44 R8:=O(NEG((x[3]-0)),O(NEG((x[6]-4)),(x[2]-2)));
45 R9:=O(NEG((x[3]-0)),O(NEG((x[6]-1)),(x[1]-1)));
```

```
46 R10:=O(NEG((x[3]-0)),O(NEG((x[6]-0)),O((x[1]-2),O((x[1]-3),(x[1]-5)))
       ));
47 R11:=O(NEG((x[3]-0)),O(NEG((x[6]-2)),(x[1]-2)));
48 R12:=O(NEG((x[3]-0)),O(NEG((x[6]-2)),(x[25]-0)));
49 R13:=O(NEG((x[6]-3)),(x[25]-0));
50 R14:=O(NEG((x[3]-0)),O(NEG((x[6]-1)),(x[18]-0)));
51 R15:=O(NEG((x[6]-0)),(x[18]-0));
52 R16:=O(NEG((x[3]-1)),(x[19]-0));
53 R17:=O(NEG((x[3]-1)),O(NEG((x[9]-3)),(x[2]-5)));
54 R18:=O(NEG((x[3]-1)),O(NEG((x[9]-0)),(x[20]-0)));
55 R19:=O(NEG((x[3]-1)),O(NEG((x[9]-1)),(x[21]-0)));
56 R20:=O(NEG((x[3]-1)),O(NEG((x[9]-2)),(x[22]-0)));
57 R21:=O(NEG((x[18]-0)),O(NEG((x[13]-1)),(x[0]-1)));
58 R22:=O(NEG((x[18]-0)),O(NEG((x[13]-0)),(x[23]-0)));
59 R23:=O(NEG((x[20]-0)),(x[23]-0));
60 R24:=O(NEG((x[21]-0)),O(NEG((x[11]-2)),(x[2]-7)));
61 R25:=O(NEG((x[21]-0)),O(O(NEG((x[11]-0)),NEG((x[14]-1))),(x[0]-3)));
62 R26:=O(NEG((x[21]-0)),O(NEG((x[11]-1)),(x[0]-0)));
63 R27:=O(NEG((x[22]-0)),(x[25]-0));
64 R28:=O(NEG((x[23]-0)),O(O(NEG((x[12]-0)),NEG((x[7]-2))),(x[2]-3)));
65 R29:=O(NEG((x[23]-0)),O(O(NEG((x[12]-0)),NEG((x[7]-1))),(x[2]-4)));
66 R30:=O(NEG((x[23]-0)),O(O(NEG((x[12]-0)),O(NEG((x[7]-1)),NEG((x[8]-1)
       ))),(x[2]-8)));
67 R31:=O(NEG((x[23]-0)),O(O(NEG((x[12]-0)),O(NEG((x[7]-1)),O(NEG((x
       [8]-1)),NEG((x[8]-0)))))),(x[24]-0)));
```

```
68 R32:=O(NEG((x[23]-0)),O(O(NEG((x[12]-0)),NEG((x[7]-0))),(x[24]-0)));
69 R33:=O(NEG((x[23]-0)),O(NEG((x[12]-1)),(x[24]-0)));
70 R34:=O(NEG((x[24]-0)),(x[25]-0));
71 R35:=O(NEG((x[25]-0)),(x[0]-0));
72
73 K:=Ideal(R0,R1,R2,R3,R4,R5,R6,R7,R8,R9,R10,R11,R12,R13,R14,R15,R16,
       R17,R18,R19,R20,R21,R22,R23,R24,R25,R26,R27,R28,R29,R30,R31,R32,
       R33,R34,R35);
74
75 GBasis(MEMORY.J+K);
```

8. Comparación de modelos

En este trabajo, consideraremos diferentes modelos de representación para implementar sistemas expertos en sistemas de computación algebraica. Cada modelo de representación tiene ventajas e inconvenientes. El modelo más adecuado depende de la especificación de la aplicación y por lo tanto, debe ser elegido dependiendo de cada caso particular.

El modelo 'Concepto-Atributo-Valor' y el modelo basado en lógica proposicional Booleana, pueden ser usados para representar conocimiento. Ambos modelos son equivalentes en el sentido de que cada sistema experto representado bajo un modelo puede ser representado bajo el otro modelo. El modelo 'Concepto-Atributo-Valor' presenta la ventaja sobre la lógica proposicional Booleana de que generalmente implica un menor número de variables y un menor número de fórmulas en la base de conocimiento por lo que se obtiene un sistema experto más pequeño. Sin embargo, los sistemas expertos implementados en lógica proposicional pueden ser implementados en un sistema de computación algebraica, PolyBoRi, el cual realiza el cálculo muy rápido. En general, hemos observado que cuando un sistema experto trata con un gran número de variables, las cuales pueden tomar un gran número de valores posibles, el modelo 'Concepto-Atributo-Valor' es más adecuado que el modelo lógico proposicional Booleano. Por el contrario, cuando un gran número de variables solo pueden tomar dos posibles valores, la lógica proposicional Booleana proporciona un mejor rendimiento.

Hemos diseñado dos experimentos: uno que comprueba que un sistema experto desarrollado en el modelo 'Concepto-Atributo-Valor' puede ser mejor que el mismo desarrollado en el modelo lógico proposicional Booleano; y otro, que comprueba lo contrario, es decir, que un sistema experto desarrollado en el modelo lógico proposicional Booleano puede ser mejor que el mismo desarrollado en el modelo 'Concepto-Atributo-Valor'.

8.1. Experimento 1

Dado un número natural, m, consideraremos un sistema experto con tres variables,X_1, X_2, X_3 de tal modo que cada variable puede tomar un valor de un conjunto de m posibles valores $\{v_1, ..., v_m\}$. La base de conocimiento de este sistema experto está compuesto por tan solo una regla:

$$(X_1 = X_2) \rightarrow (X_2 \neq X_3)$$

Por ejemplo, si $m = 6$, especificaremos nuestro sistema experto en nuestro lenguaje de la siguiente manera:

```
BEGIN
  BEGIN_VARS
    X[3]: v1, v2, v3, v4, v5, v6;
  END_VARS

  BEGIN_KB
     (X[1]=X[2]) -> (X[2]<>X[3] )
  END_KB
END
```

Desde nuestro lenguaje hemos traducido el sistema experto en ambos enfoques algebraicos:

- Sentencias para nuestro sistema de computación algebraica, *PolyBoRi*, de acuerdo con el enfoque basado en lógica proposicional Booleana. Hemos elegido *PolyBoRi* como el sistema de computación algebraica ya que es el más rápido para implementar algebraicamente sistemas expertos basados en lógica proposicional Booleana.

- Sentencias para el sistema de computación algebraica, *Singular*, de acuerdo con el enfoque algebraico basado en el modelo 'Concepto-Atributo-Valor'.

En la figura 8.1, mostramos una gráfica en la que estudiamos el rendimiento de ambos enfoques algebraicos. Como se puede observar, el enfoque algebraico basado en el modelo 'Concepto-Atributo-Valor' proporciona mejores resultados.

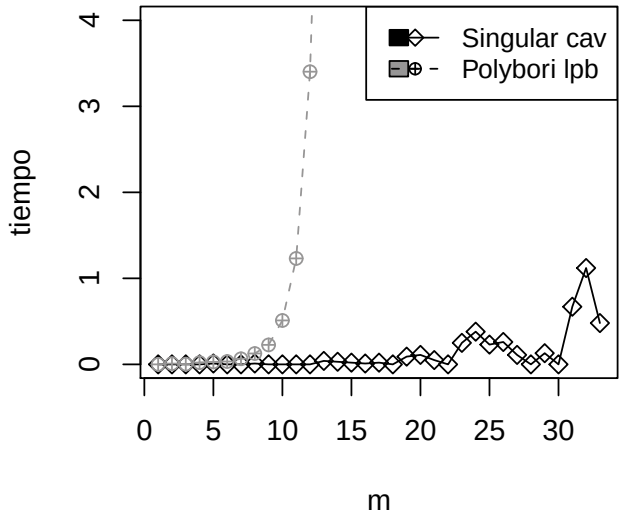

Figura 1: Experimento 1
Comparación de modelos con los sistemas de computación algebraica de mejor rendimiento

8.2. Experimento 2

Dado un número natural, n, consideraremos un sistema experto con n variables, $X_1, X_2, \ldots, X_n$, de tal modo que cada variable puede tomar un valor de un conjunto de 5 posibles valores $\{v_1, v_2, \ldots, v_5\}$. La base de conocimiento de este sistema experto esta compuesto por $n-1$ reglas:

$$(X_1 = v_1) \rightarrow (X_2 \neq v_2)$$

$$(X_2 = v_1) \rightarrow (X_3 \neq v_2)$$

$$\ldots$$

$$(X_{n-1} = v_1) \rightarrow (X_n \neq v_2)$$

Por ejemplo, si $n = 6$, especificaremos nuestro sistema experto en nuestro lenguaje de la siguiente manera:

```
BEGIN
  BEGIN_VARS
    X[6]: v1, v2, v3, v4, v5;
  END_VARS

  BEGIN_KB
     (X[1]=v1) -> (X[2]<>v2);
     (X[2]=v1) -> (X[3]<>v2);
     (X[3]=v1) -> (X[4]<>v2);
     (X[4]=v1) -> (X[5]<>v2);
```

```
      (X[5]=v1) -> (X[6]<>v2);

  END_KB

END.
```

Desde nuestro lenguaje hemos traducido el sistema experto en ambos enfoques algebraicos:

- Sentencias para nuestro sistema de computación algebraica, *PolyBoRi*, de acuerdo con el enfoque basado en lógica proposicional Booleana. Hemos elegido *PolyBoRi* como el sistema de computación algebraica ya que es el más rápido para implementar algebraicamente sistemas expertos basados en lógica proposicional Booleana.
- Sentencias para el sistema de computación algebraica, *Singular*, de acuerdo con el enfoque algebraico basado en el modelo 'Concepto-Atributo-Valor'.

En la figura 8.2, mostramos una gráfica en la que estudiamos el rendimiento de ambos enfoques algebraicos. Como se puede observar, el enfoque algebraico basado en el modelo lógico proposicional Booleano proporciona mejores resultados.

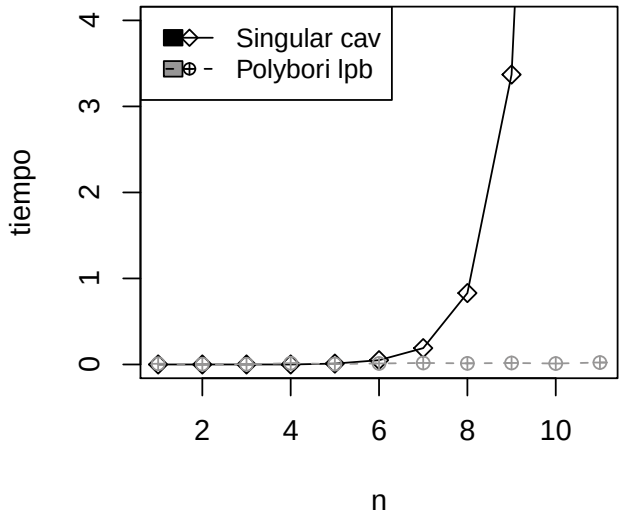

Figura 2: Experimento 2
Comparación de modelos con los sistemas de computación algebraica de mejor rendimiento

9. Trabajo futuro

En este proyecto final de carrera hemos desarrollado un lenguaje para implementar sistemas expertos por medio de sistemas de computación algebraica. El lenguaje que hemos desarrollado podría ampliarse en el futuro para que permita lo siguiente:

1. Definir para cada variable el número de valores tanto discretos como continuos.
2. Definir fórmulas atómicas en las que aparezcan las relaciones $\geq$ y $\leq$.
3. Definir funciones auxiliares diseñadas por el usuario.

Además, creemos que sería relevante que el lenguaje sirviera para manejar sistemas expertos que manejan incertidumbre en el conocimiento. A continuación, describimos el modelo algebraico basado en lógica multivaluada para manejar incertidumbre.

9.1. Sistemas expertos basados en lógica proposicional multivaluada

La lógica proposicional multivaluada puede ser usada para representar incertidumbre en el conocimiento.

9.1.1. Formalismo de representación

En lógica proposicional multivaluada, las variables $X_1, ...X_m$ pueden tomar diferentes grados de verdad. Los posibles valores de verdad serán nombrados por números naturales entre 0 y $q-1$, donde 0 significa, como es habitual, 'falso' y $q-1$ 'verdadero'. Los lógicos han desarrollado diferentes lógicas proposicionales multivaluadas con diferentes grados de verdad. Cada lógica no sólo implementa de manera diferente las conectivas lógicas $\wedge$, $\vee$, $\neg$, $\rightarrow$, sino que también definen nuevas conectivas. Cada conectiva unaria, f_u (respectivamente conectiva binaria, f_b), tiene asociada una función $F_u : \{0, ..., q-1\} \longrightarrow \{0, ..., q-1\}$ (respectivamente una función $F_b : \{0, ..., q-1\} \times \{0, ..., q-1\} \longrightarrow \{0, ..., q-1\}$).

Definición Una fórmula A es definida recursivamente de la siguiente manera:

- X_i donde X_i es una variable.
- $f_u(B)$ donde B es una formula y f_u es un conectivo unario de la lógica.
- $f_b(B, C)$ donde B y C son fórmulas, and f_b es un conectivo binario de la lógica.

Usaremos $\mathcal{C}$ para denotar un conjunto de fórmulas. Llamaremos estado, S, a una posible situación en la cual cada variable toma el grado de verdad entre 0 y $q-1$. Dado un estado S y una variable X, la expresión $S(X)$ denota el valor que la variable X toma en el estado S. Dado una fórmula A y un estado S, la expresión $S(A)$ denota un valor entre 0 y $q-1$ asociado a la fórmula A en el estado S.

Definición Sea A una formula. Sea S un estado. $S(A)$ es definida de la siguiente manera:

Caso A es X_i donde X_i es una variable.

$S(A) = S(X_i)$

Caso A es $f_u(B)$ donde B es una formula y f_u un conectivo unario asociado a la función $F_u : \{0, ..., q-1\} \to \{0, ..., q-1\}$.

$S(A) = F_u(S(B))$

Caso A es $f_b(B, C)$ donde B y C son fórmulas.

$S(A) = F_b(S(B), S(C))$

Definición Sea A una fórmula y sea S un estado. La fórmula A se cumple en S sí y solo sí $S(A) = q-1$

Por medio de las fórmulas describimos la entrada, la salida y la base de conocimiento de un sistema experto.

Entrada La entrada es un conjunto de fórmulas con la forma X o $f_u(X)$ donde X es una variable de entrada y f_u es un conectivo unario. El símbolo $\mathcal{I}$ denota un conjunto de fórmulas representando la entrada del sistema experto.

Base de conocimiento La base de conocimiento es un conjunto finito de fórmulas. El símbolo $\mathcal{K}$ denota el conjunto de fórmulas que representan la base de conocimiento del sistema experto.

Salida La salida es un conjunto de fórmulas con la forma X or $f_u(X)$ donde X es una variable de salida. El símbolo $\mathcal{O}$ denota el conjunto de fórmulas que representan la salida de conocimiento del sistema experto.

9.1.2. Modelo algebraico

El problema de determinar si una fórmula puede ser deducida de otras puede ser tratado algebraicamente cuando q es una potencia de un número primo [1] (esto es $q = p^n$ donde p es un primo y n un número natural) .

En primer lugar, vamos a considerar como las fórmulas son traducidas a polinomios. Primero de todo, necesitamos definir el siguiente ideal J en $\mathbb{Z}_p[x_1, \ldots, x_m, t]$:

$$J = \langle x_1^q - x_1, \ldots, x_m^q - x_m, r \rangle$$

donde p es un primo, $q = p^n$ y r es un polinomio irreducible en $\mathbb{Z}_p$ de grado n.

Una vez definido el ideal J, las fórmulas son transformadas en un polinomio $\mathbb{Z}_q[x_1, \ldots, x_m]$ a través de la siguiente función:

Definición Definimos recursivamente la función $\varphi : \mathcal{C} \longrightarrow \mathcal{A}$ de la manera siguiente:

Caso A es X_i donde X_i es una variable.

$$\varphi(A) = \varphi(X_i) = x_i$$

Caso A es $f_u(B)$ donde B es una fórmula.

$$\varphi(A) = \mathrm{NF}(p_u(\varphi(B)), J)$$

donde $p_u(x)$ es un polinomio particular que traduce la conectiva f_u.

Caso A es $f_b(B, C)$ donde B y C son fórmulas.

$$\varphi(A) = \mathrm{NF}(p_b(\varphi(B), \varphi(C)), J)$$

donde $p_b(x, y)$ es un polinomio particular que traduce la conectiva f_b.

En relación con los sistemas expertos, nosotros consideraremos los siguientes ideales:

Ideal J Ideal necesario para definir el anillo $\mathcal{A}$ en el que las fórmulas son representadas.

$$J = \langle x_1^q - x_1, \ldots, x_m^q - x_m \rangle$$

Ideal I Ideal realacionado con la entrada del sistema experto.

$$I = \langle \varphi(A_i) | A_i \in \mathcal{I} \rangle$$

Ideal K Ideal realacionado con la base de conocimiento del sistema experto.

$$K = \langle \varphi(A_i) | A_i \in \mathcal{K} \rangle$$

Una vez traducidas las fórmulas en polinomios, podremos tratar algebraicamente el problema de determinar si una fórmula puede ser deducida por un sistema experto.

Teorema 9.1 *Una fórmula B puede ser deducida por un sistema experto sí y solo sí:*

$$\varphi(B) \in I + J + K$$

Este teorema puede ser también reescrito de la siguiente manera:

Teorema 9.2 *Una fórmula B puede ser deducida por un sistema experto sí y solo sí:*

$$NF(\varphi(B), I + J + K) = 0$$

10. Conclusiones

En este trabajo final de carrera hemos diseñado un lenguaje para implementar rápida y cómodamente sistemas expertos a través de sistemas de computación algebraica. Este lenguaje contempla las siguientes ventajas:

i) El lenguaje es independiente del enfoque algebraico elegido para implementar el sistema experto. En este trabajo mostraremos cómo traducir un sistema experto descrito en nuestro lenguaje en ambos enfoques algebraicos. De esta manera podemos generar fácilmente el código del sistema experto en ambos enfoques algebraicos con el fin de experimentar con ellos y elegir el que mejor rendimiento muestre.

ii) El lenguaje es independiente del sistema de computación algebraica usado para implementar el sistema experto. De esta manera, cambiar el sistema de álgebra computacional sólo conlleva volver a compilar el sistema experto descrito en nuestro lenguaje.

iii) El lenguaje no usa notación matemática. Por lo tanto, los desarrolladores no necesitan conocer conceptos matemáticos como bases de Gröebner o formas normales, que son comúnmente utilizados en estos enfoques algebraicos.

iv) Nuestro lenguaje proporciona una manera sencilla de implementar sistemas expertos. De hecho, el código de un sistema experto es normalmente mucho más reducido y mucho más comprensible que el código descrito en los sistemas de computación algebraica. Nuestro lenguaje proporciona una manera muy intuitiva y fácil de escribir reglas (las cuáles son la manera más natural de describir la base de conocimiento de un sistema experto).

A. APÉNDICE: Especificación del traductor en EBNF y JAVACC

La gramática en EBNF [20], predictiva recursiva por la izquierda y LL(1), es la siguiente:

unprograma ::= 'BEGIN'
'BEGIN_VARS' decvariable {decvariable} 'END_VARS'
'BEGIN_KB' {regla} 'END_KB'
['BEGIN_INPUT' [input] 'END_INPUT']
'END' .

decvariable ::= id ['[' numero ']'] { ',' variable} ':' id {',' id} ';'

variable ::= id ['[' numero ']']

regla ::= expresion [restoregla] ';'

restoregla ::=consecuente {consecuente}
| {antecedente} "→" expresion {consecuente}

consecuente ::= 'OR' expresion

antecedente::= 'AND' expresion

expresion ::= '(' variable operador variable ')'

operador::= '=' | '<>'

variable ::= id ['[' <numero> ']']

input ::= expresion {',' expresion} ';'

id ::= caracter {caracter | "0" |' numero | "_" }

numero ::= caracternumerico{["0" | caracternumerico]}

caracter ::= restocaracter { "_"| restocaracter}

restocaracter ::= "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i" | "j" | "k" | "l" | "m" | "n" | "ñ" | "o" |
"p" |"q" | "r" | "s" | "t" | "u" | "w" | "x" | "y" | "z"

caracternumerico ::= "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

Se ha elegido JavaCC como plataforma para generar el analizador léxico-sintáctico por la posibilidad de obtener un traductor que pueda ser ejecutado en distintas plataformas siempre que se tenga instalada la máquina virtual de Java. Además es cómodo introducir la generación de código directamente en el traductor en lenguaje JAVA.

A partir de la definición en EBNF es muy sencillo conseguir la gramática para JavaCC [21] es:

unprograma ::= "BEGIN"
"BEGIN_VARS" (decvariable())+ "END_VARS"
"BEGIN_KB" (regla())* "END_KB"
["BEGIN_INPUT" [input()] "END_INPUT"]
"END".

decvariable ::= <id> ["[" numero "]"] (',' <id> ['[' <numero> ']'])* ":" <id> ("," <id>)* ";"

regla ::= expresion() [restoregla()] ";"

restoregla ::=consecuente() (consecuente())*
| (antecedente())* "→" expresion() (consecuente())*

consecuente ::= "OR" expresion()

antecedente ::= "AND" expresion()

expresion ::= "(" variable() operador() variable() ")"

operador ::= "=" | "<>"

variable ::= <id> ["[" <numero> "]"]

input ::= expresion() ("," expresion())* ";"

id ::= ["A"-"Z", "a"-"z" , "_"](["A"-"Z", "a"-"z","0"-"9"' , "_" ,"\"])*

numero ::= ["1"-"9"](["0"-"9"])*

B. APÉNDICE: Comparación de sistemas de computación algebraica

B.1. Mejores CAS para experimento 1

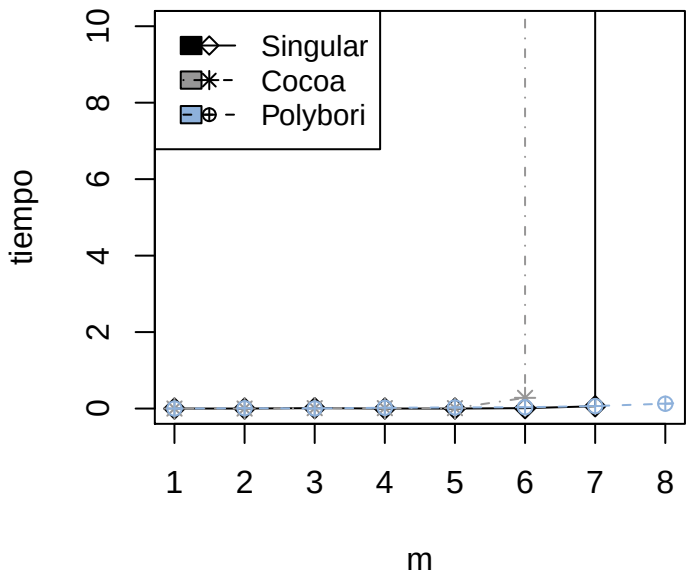

Figura 3: Experimento 1
Resultados de PolyBoRi, Singular y CoCoA y el modelo lógico proposcional booleano

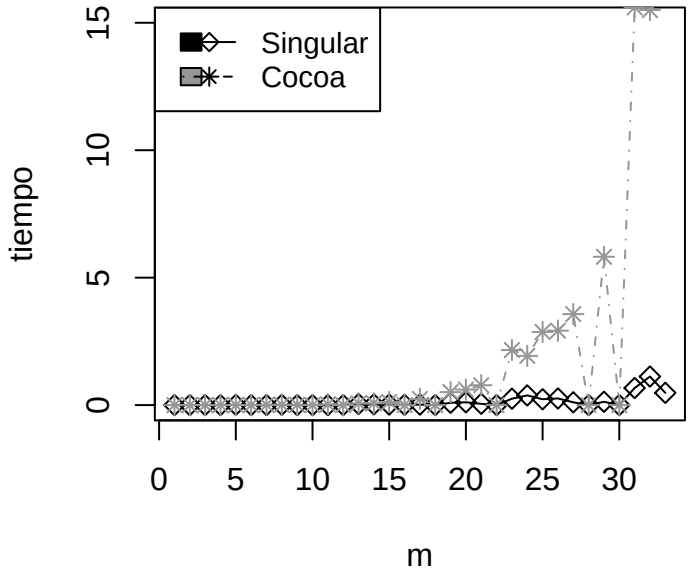

Figura 4: Experimento 1
Resultados de Singular y CoCoA y el modelo 'Concepto-Atributo-Valor'

B.2. Mejores CAS para experimento 2

La figura que se muestra a continuación, demuestra que el sistema algebraico de cómputo más acertado para representar un sistema experto con muchos valores para pocas variables es el sistema PolyBoRi.

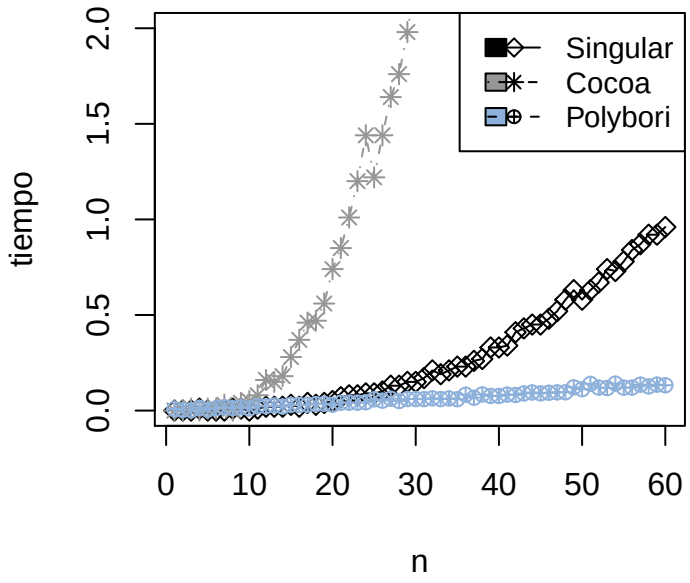

Figura 5: Experimento 2
Resultados de PolyBoRi, Singular y CoCoA y el modelo lógico proposcional booleano

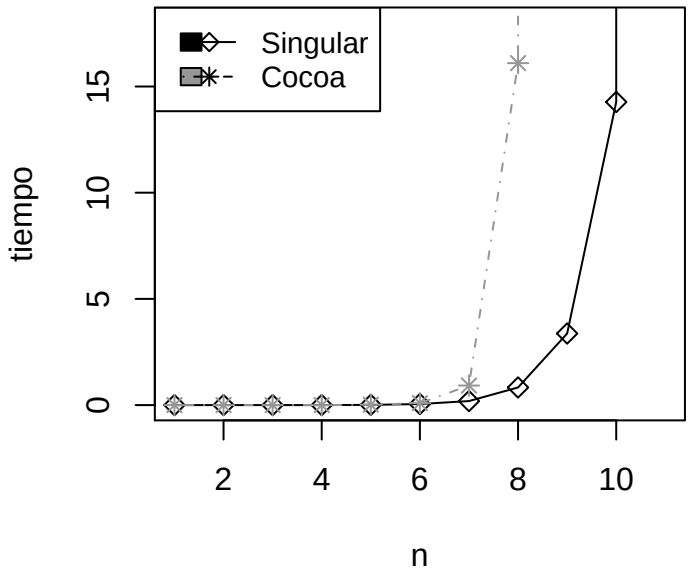

Figura 6: Experimento 2
Resultados de Singular y CoCoA y el modelo 'Concepto-Atributo-Valor

C. APÉNDICE: Scripts para la automatización de pruebas

Se han generado dos programas para generar un sistema experto cuyas características que se corresponden a las especificaciones de los experimentos de los apartados 8.1 y 8.2. También se han creado tres *scripts* para ejecutar estos sistemas expertos de manera sistemática en los sistemas de computación algebraica PolyBoRi, Singular y CoCoA.

C.1. Generador de sistemas expertos

Código necesario para generar sistemas expertos especificados en el lenguaje fuente propuesto en este proyecto. Se tiene en cuenta dos parámetros n y m, que son, número de variables y números de valores por variable correspondientemente.

Experimento1

```
1 /*
2  *  Attribution-Noncommercial 3.0 Unported - http://creativecommons.
       org/licenses/by-nc/3.0/
3  */
4 package generador;
5
6 import java.io.BufferedWriter;
7 import java.io.FileWriter;
8
9 /**
10  *
11  * @author Roberto Maestre Martinez
```

```
12  */
13 public class Generador {
14
15
16     public static void stringToFile(String msg,boolean append) {
17         try{
18   String FILE_CODE="fuenteexp1.txt";
19   FileWriter fstream = new FileWriter(FILE_CODE,append);
20   BufferedWriter out = new BufferedWriter(fstream);
21   out.write(msg);
22   out.close();
23  }catch (Exception e){
24   System.err.println("Error: " + e.getMessage());
25  }
26      }
27
28     /**
29      * @param args the command line arguments
30      */
31     public static void main(String[] args) throws Exception {
32
33         if (args.length!=2){
34             throw new Exception("Numero de parametros erroneos");
35         }
36         int n=Integer.parseInt(args[0]);
```

```
37          int m=Integer.parseInt(args[1]);
38          if (n<2) throw new Exception("Deben existir al menos dos
                variables");
39          if (m<2) throw new Exception("Deben existir al menos dos
                valores");
40
41          System.out.println("Generando "+n+" variables con "+m+"
                valores cada una");
42
43          stringToFile("BEGIN\n",false);
44
45              stringToFile("\tBEGIN_VARS\n",true);
46                      stringToFile("\t\tvar["+(n-1)+"]:\t",true);
47                      for(int x=1;x<m;x++){
48                          stringToFile("val"+x,true);
49                          if (x<m-1) stringToFile(",",true);
50                          else stringToFile(";\n",true);
51                      }
52              stringToFile("\tEND_VARS\n\n",true);
53
54              stringToFile("\tBEGIN_KB\n",true);
55                  for(int i=1;i<n-2;i++){
56                      stringToFile("\t\t(var["+i+"]=var["+(i+1)+"])\t
                            ->\t(var["+(i+1)+"]<>var["+(i+2)+"]);\n",
                            true);
```

```
57                     }
58                 stringToFile("\n\tEND_KB\n\n",true);
59
60             stringToFile("END.",true);
61         }
62
63 }
```

Experimento 2

```
1 /*
2  *  Attribution-Noncommercial 3.0 Unported - http://creativecommons.
       org/licenses/by-nc/3.0/
3  */
4 package generador;
5
6 import java.io.BufferedWriter;
7 import java.io.FileWriter;
8
9 /**
10  *
11  * @author Roberto Maestre Martinez
12  */
13 public class Generador {
14
15
```

```
16      public static void stringToFile(String msg,boolean append) {
17          try{
18   String FILE_CODE="fuenteexp2.txt";
19   FileWriter fstream = new FileWriter(FILE_CODE,append);
20   BufferedWriter out = new BufferedWriter(fstream);
21   out.write(msg);
22   out.close();
23  }catch (Exception e){
24   System.err.println("Error: " + e.getMessage());
25  }
26       }
27
28      /**
29       * @param args the command line arguments
30       */
31      public static void main(String[] args) throws Exception {
32
33          if (args.length!=2){
34              throw new Exception("Numero de parametros erroneos");
35          }
36          int n=Integer.parseInt(args[0]);
37          int m=Integer.parseInt(args[1]);
38          if (n<2) throw new Exception("Deben existir al menos dos
                variables");
39          if (m<2) throw new Exception("Deben existir al menos dos
```

```
        valores");
40
41      System.out.println("Generando "+n+" variables con "+m+"
            valores cada una");
42
43      stringToFile("BEGIN\n",false);
44
45          stringToFile("\tBEGIN_VARS\n",true);
46              for(int i=1;i<n;i++){
47                  stringToFile("\t\tvar"+i+":\t",true);
48                  for(int x=0;x<m;x++){
49                      stringToFile("val"+x,true);
50                      if (x<m-1) stringToFile(",",true);
51                      else stringToFile(";\n",true);
52                  }
53              }
54          stringToFile("\tEND_VARS\n\n",true);
55
56          stringToFile("\tBEGIN_KB\n",true);
57              for(int i=1;i<n-1;i++){
58                  stringToFile("\t\t(var"+i+"=val1) \t -> \t (var"
                        +(i+1)+"<>val2);\n",true);
59              }
60          stringToFile("\n\tEND_KB\n\n",true);
61
```

```
62            stringToFile("END.",true);
63        }
64
65 }
```

C.2. Automatización

Los siguientes *scripts* permitirán automatizar las pruebas a través de los siguientes pasos:

- Generar el código específico del sistema experto en nuestro lenguaje teniendo en cuenta los parámetros n y m que son el número de variables y el número de valores por variable correspondientemente.
- Ejecutar el traductor para traducir el sistema experto generado con la opción "-b" para tener en cuenta el tiempo de cálculo de la base de Gröebner.
- Ejecutar en el sistema de computación algebraica Singular, CoCoA o PolyBoRi el sistema experto, volcando la salida de la ejecución en el fichero de resultados.

El fichero de resultados será utilizado para comparar analíticamente ambos experimentos.

PolyBoRi

```
1 #!/bin/bash
2 SALIDA="resultados.txt"
3 N=20;
4
5 #Voy al directorio donde estan todos los scripts
6 cd /home/roberto/traductor
```

```
7
8 #Reinicio la salida
9 echo "" > $SALIDA
10 for (( c=2; c<=N; c++ ))
11 do
12  #Compilo la pruebte, ese decir, el codigo fuente
13  java -jar Experimento1.jar 4 $c
14  #Creo el sistema experto
15  java Traductor 1 1 -b < fuenteexp1.txt
16  #ejecuto el sistema experto obteniendo el tiempo de salida
17  python semlpb_polybori.txt >> $SALIDA
18 done
```

Singular

```
1 #!/bin/bash
2 SALIDA="resultados.txt"
3 N=35;
4
5 #Voy al directorio donde estan todos los scripts
6 cd /home/roberto/traductor
7
8 #Reinicio la salida
9 echo "" > $SALIDA
10 for (( c=3; c<=N; c++ ))
11 do
```

```
12 #Compilo la pruebte, ese decir, el codigo fuente
13 java -jar Experimento1.jar 4 $c
14 #Creo el sistema experto
15 java Traductor 2 1 -b < fuenteexp1.txt
16 echo "quit();" >> semcva_singular.txt
17 #ejecuto el sistema experto obteniendo el tiempo de salida
18 Singular semcva_singular.txt > tmp
19 #sOlo necesita la linea 5 que es la que tiene el tiempo
20 count=1 # counting lines from 1
21 while read -r line ; do
22   if [ $count == 6 ] ; then
23   echo $line >> $SALIDA
24   fi
25   let count=$count+1
26 done < tmp
27 done
```

CoCoA

```
1 #!/bin/bash
2 SALIDA="resultados.txt"
3 N=35;
4
5 #Voy al directorio donde estan todos los scripts
6 cd /home/roberto/traductor
7
```

```
8 #Reinicio la salida
9 echo "" > $SALIDA
10 for (( c=3; c<=N; c++ ))
11 do
12  #Compilo la pruebte, ese decir, el codigo fuente
13  java -jar Experimento1.jar 4 $c
14  #Creo el sistema experto
15  java Traductor 2 2 -b < fuenteexp1.txt
16  echo "QUIT;" >> semcva_cocoa.txt
17  #ejecuto el sistema experto obteniendo el tiempo de salida
18  cocoa semcva_cocoa.txt > tmp
19  #sOlo necesita la linea 5 que es la que tiene el tiempo
20  count=1 # counting lines from 1
21  while read -r line ; do
22   if [ $count == 18 ] ; then
23   echo ${line:11:4} >> $SALIDA
24   fi
25   let count=$count+1
26  done < tmp
27 done
```

D. APÉNDICE: Introducción a las bases de Gröbner

(Traducción del artículo 'Gröbner Bases: A Short Introduction for Systems Theorists' [22])

El método de las bases de Gröbner proporciona un enfoque uniforme para solucionar un amplio abanico de problemas expresados en términos de *conjuntos de polinomios multivariantes*. Un polinomio mutlivariante es un polinomio en más de una variable, como por ejemplo $P(x, y) = a_{22}x^2y^2 + a_{21}x^2y + a_{11}xy^2 + a_{10}x + a_{01}y + a_{00}$
Se deben definir varios conceptos que ayudarán a entender mejor que son las bases de Gröbner.

D.1. División (reducción) de polinomios multivariantes

Definimos los siguientes polinomios bivariantes,

- $g = x^2y^3 + 3xy^2 - 5x$
- $f_1 = xy - 2y$,
- $f_2 = 2y^2 - x^2$,

y el siguiente con junto de polinomios

- $F = \{f_1, f_2\}$

sus monomios están ordenados lexicográficamente teniendo y mayor peso que x y presentados en orden descendente de izquierda a derecha. El mayor monomio (es decir, el de mas a la izquierda) es llamado el monomio líder en el polinomio. **Hay infinitas maneras de ordenar los polinomios que son admisibles en la teoría de las bases de Gröbner**

Un posible paso para dividir ("reducir"), puede ser, "reducir el polinomio g módulo f_1":

- $h = g - (3y)f_1 = -5x + 6y^2 + x^2y^3$,

observamos que uno de los monomios de g se cancela con el monomio líder de $-(3y)f_1$. En esta situación podemos escribir la operación de la siguiente manera (se lee: "g se reduce a h modulo $f1$"):

- $g \rightarrow_{f_1} h$

En general muchas reducciones son posibles. Dado un conjunto de polinomios F y un polinomio g, muchas reducciones de g módulo polinomios en F son posibles, en nuestro ejemplo podemos tener:

- $g \rightarrow_{f_1} h_2$, donde $h_2 = g - (xy^2)f_1$
- $g \rightarrow_{f_2} h_3$, donde $h_3 = g - (\frac{1}{2}x^2y)f_2$

La división de polinomios multivariantes siempre termina pero no es única.
Escribimos $g \rightarrow_F h$ sí

- $g \rightarrow_f h$ para algún $f \in F$
- y también escribimos, $g \rightarrow_{*_F} h$ si g es reducido a h en un número finito de pasos.

También escribimos $\underline{h}_F$ si h no puede ser reducido más. Dos puntos fundamentales para el concepto de reducción son:

- **Terminación**: Para cualquier g Y F, no hay infinitos pasos de reducción modulo F empezando desde g
- **La reducción es algorítmica**: Hay un algoritmo RF el cual produce una forma reducida de F para cualquier polinomio dado g, por ejemplo, para todo g y F. $g \rightarrow_{F_*} RF(F, g)_F$
- **No-singularidad** Dado g y F pueden existir H y K, de manera que:

  $\underline{h}_F \leftarrow_{F^*} g \rightarrow_{F^*} \underline{k}_F$,pero $h \neq k$

D.2. Definición normal de base de Gröbner

F es una base de Gröbner $:\Longleftrightarrow \rightarrow_F$ es única

$$\bigvee_{g,h,k} (\underline{h}_F \leftarrow_{F^*} g \rightarrow_{F^*} \underline{k}_F) \Longrightarrow h = k$$

D.3. La aplicación de teoría de bases de Gröbner

A primera vista, uno no puede ver por qué la propiedad definitoria de las bases de Gröbner debería jugar un papel fundamental. La importancia de esta propiedad se deriva del siguiente hecho:

Hecho: Las bases de Gröbner tienen 'muchas buenas propiedades' y por lo tanto, para bases de Gröebner, muchos problemas fundamentales pueden ser resueltos por algoritmos 'sencillos'.

Ejemplo (El problema principal de la teoria de ideales de polinomios):

Sea F un conjunto de polinomios

Si F es una base de Gröbner, entonces:

$f \in \text{Ideal}(F) \Longleftrightarrow f \rightarrow_{*_F} 0$

Aquí, $\text{Ideal}(F)$ es un ideal generado por F, es decir, el conjunto de todos los polinomios de la forma $\sum_{i=1}^{m} p_i f_i$, con f_i en F y polinomios aleatorios p_i. Como una consecuencia de la propiedad anterior, la cuestión relacionada con la pertenencia o no $f \in \text{Ideal}(F)$, para las bases de Gröbner F, puede ser deducida solamente reduciendo f módulo F y comprobando si el resultado es o no 0. Para un F general, esta pregunta es muy difícil de decidir y, de hecho, en la literatura antigua sobre teoria de ideales de polinomios fué llamada 'El problema principal de la teoria de ideales de polinomios'

Ejemplo (El problema de 'La eliminación'): Dado F un conjunto de polinomios en las indeterminadas $x_1, \ldots, x_n$ y dado $i \leq n$:

Si F es una base de Gröbner, entonces:

$\text{Ideal}(F) \cap K[x_1, \ldots, x_n] = \text{Ideal}(F \cap K[x_1, \ldots, x_n])$

Como consecuencia, una base para la "i-ésima eliminación ideal"$\text{Ideal}(F) \cap K[x_1, \ldots, x_n]$ de unas bases de Gröbner finitas F pueden ser obtenidos solamnete cogiendo aquellos polinomios en F que dependen solo de los primeros i indeterminados. De nuevo, este problema es muy dificil para un F general.

D.4. ¿Como puede una base de Gröbner ser construida?

El principal problema ahora es como, dado un conjunto F arbitrario y finito de polinomios (multivariantes), uno puede encontrar un conjunto de polinomios G tal que $\text{Ideal}(F) = \text{Ideal}(G)$ y G es una base de Gröbner.

El problema puede ser solucionado por el siguiente algoritmo:

Empieza con $G := F$

Para cada par de polinomios $f_1, f_2 \in G$

Computa el "S-Polinomio"de f_1, f_2

Reduce forma h w.r.t. G.

Si $h = 0$, considera el siguiente par

Si $h \neq 0$, añade h a G e itera.

El algoritmo anterior necesita de la computación de "S-Polinomios". De nuevo, damos la definición con un ejemplo:

$$f_1 := xy - 2y, f_2 := 2y^2 - x^2,$$

$$\text{S-Polinomio}[f_1, f_2] = yf_1 - \frac{1}{2}xf_2 = \frac{x^3}{2} - 2y^2$$

E. APÉNDICE: Código Traductor JavaCC

```
1 /*
2  *  Attribution-Noncommercial 3.0 Unported - http://creativecommons.
       org/licenses/by-nc/3.0/
3  */
4 options {Ignore_case = false;
5   Common_Token_Action = true;
6   Error_Reporting = true;
7   Output_directory="R:\\TFC-DATOS\\javacc-4.0\\traductor";
8 }
9 PARSER_BEGIN (Traductor)
10
11 import java.util.*;
12 import java.io.*;
13 import tablasimboloscompilador.*;
14
15  public class Traductor {
16
17   //Tabla de simbolos del sistema
18   private static TS ts;
19   //Flags globales para indicar el tipo de compilacion
20   private static String MODELO="";
21   private static String SALIDA="";
22   //Guardara donde se genera el codigo
```

```
23  private static String OUTPUT_FILE;
24  //Estructuras globales para guardar polinomios y operadores
25  private static Vector<String> polinomios;
26  private static Vector<Boolean> operadores;
27  //No sindica si la primera expresion de una regla corresponde a un
28  //consecuente o a un antecedente
29  private static boolean FLAGANTECEDENTE;
30  private static boolean FLAGNEGARTODO;
31  private static boolean FLAGEXPRESIONVALORCOMPLEJO;
32  private static boolean FLAGEXPRESIONVALORCOMPLEJONEGADO;
33  private static boolean FLAGEXISTEENTRADA;
34  //Guardamos el numero de polinomios de restriccion, polinomios de
        variables
35  //polinomios de reglas y polinomios de entrada para luego generar
        las llamadas
36  //para el calculo de ideales y de la base de groebner
37  private static int NUMPOLRES=0;
38  private static int NUMPOLVARS=0;
39  private static int NUMREGLAS=0;
40  private static int NUMREGLASINPUT=0;
41  //En la generacion de codigo, si esta activado este flag, se
        incorporan
42  //marcas de tiempo
43  private static boolean BENCHMARK;
44
```

```
45  //Crea un puntero que apunta a la estructura global polinomios,
        devuelve
46  //el puntero y borra la estructura global
47  public static Vector<String> repols(){
48   Vector<String> aux=(Vector) polinomios.clone();
49   polinomios.clear();
50   return aux;
51
52  }
53
54  //Crea un puntero que apunta a la estructura global operadores,
        devuelve
55  //el puntero y borra la estructura global
56  public static Vector<Boolean> reops(){
57   Vector<Boolean> aux=(Vector) operadores.clone();
58   operadores.clear();
59   return aux;
60
61  }
62
63  //Annade los polinomios de la estructura global "polinomios"
64  //a un auxiliar y vacia la estructura global
65  public static void aniadepols(Vector<String> v){
66   for (int i=0;i<polinomios.size();i++){
67    v.add(polinomios.get(i));
```

```
68     }
69     polinomios.clear();
70   }
71
72   //Annade los polinomios de la estructura global "operadores"
73   //a un auxiliar y vacia la estructura global
74   public static void aniadeops(Vector<Boolean> v){
75     for (int i=0;i<operadores.size();i++){
76       v.add(operadores.get(i));
77     }
78     operadores.clear();
79   }
80
81   //Imprime un vector con variables de un polinomio.
82   //Funcion usada para DEBUG
83   public static void imprime(Vector<String> v){
84     System.out.print("");
85     for (int i=0;i<v.size();i++){
86       System.out.print(v.get(i)+"␣");
87     }
88     System.out.println("");
89   }
90
91   //Devuelve un vector de polinomios en una cadena
```

```
92  public static String acadena(Vector<String> v,Vector<Boolean> op,
        boolean negado){
93   //Si solo es un polinomio, lo devuelvo directamente
94   if (v.size()==1) {
95    return isneg(v.get(0),negado,op.get(0));
96   } else {
97   //Si no, tengo que hacer pares para devolver O(a,O(b,c))
98    return encadenaOrs(v,op,0,negado);
99   }
100 }
101 //Funcion recursiva para generar los OR(a,b) encadenados a partir
        de un vector
102 //de polinomios
103 public static String encadenaOrs(Vector<String> v,Vector<Boolean>
        op,int index,boolean negado){
104  if (index==v.size()-2){
105   return "O("+isneg(v.get(index),negado,op.get(index))+","+isneg(v.
        get(index+1),negado,op.get(index+1))+")";
106  } else {
107   return "O("+isneg(v.get(index),negado,op.get(index))+","+
        encadenaOrs(v,op,index+1,negado)+")";
108  }
109 }
110
111
```

```
112  //Funcion recursiva para generar los OR(a,b) encadenados a partir
         de un vector
113  //de polinomios
114  public static String encadenaOrSimple(Vector<String> v,int index){
115    if (index==v.size()-2){
116    return "O("+v.get(index)+","+v.get(index+1)+")";
117   } else {
118    return "O("+v.get(index)+","+encadenaOrSimple(v,index+1)+")";
119   }
120  }
121
122
123  //Annade neg(s) a una variable "s" pasada pos parametro comprobando
         el operador
124  //y comprobando
125  public static String isneg(String s,boolean negado,boolean op){
126   //Si es antecedente => negado==true  -> la operacion negada
127   //         negado==false -> la operacion normal
128   boolean flag=false;
129   if (!negado) flag=!op;
130   else    flag=op;
131   //Devuelvo el valor correcto
132   if (flag)
133    return "NEG("+s+")";
134   else
```

```
135     return s;
136   }
137
138   //Muestra un mensaje de error indicando que los parametros son
          incorrectos
139   //por consola
140   public static void mostrar_error_de_parametros(){
141     System.out.println("\nUSO DEL TRADUCTOR:");
142     System.out.println(">java Traductor \"MODELO\" \"SALIDA\" [-ts][-
          b] < fuente.txt \n");
143     System.out.println("\t -ts: Muestra la tabla de simbolos");
144     System.out.println("\t -b: Annade marcas de benchmark\n");
145     System.out.println("Posibles Usos:");
146     System.out.println(">java Traductor 1 1 < fuente.txt (Modelo LPB
          y sistema Polybori)");
147     System.out.println(">java Traductor 1 2 < fuente.txt (Modelo LPB
          y sistema Singular)");
148     System.out.println(">java Traductor 1 3 < fuente.txt (Modelo LPB
          y sistema Cocoa)");
149     System.out.println(">java Traductor 2 1 < fuente.txt (Modelo CAV
          y sistema Singular)");
150     System.out.println(">java Traductor 2 2 < fuente.txt (Modelo CAV
          y sistema Cocoa)");
151     System.out.println("\n[LPB]=Modelo logico proposicional booleano"
          );
```

```
152     System.out.println("[CAV]=Modelo 'Concepto-Atributo-Valor' \n");
153     System.exit(0);
154   }
155
156
157
158
159   public static void generarPolinomiosRestriccion(){
160    try {
161     if (MODELO.equals("PROPOSICIONAL")) {
162      //Obtengo los pares con el formato proposicional e indico los
163      //separadores para cada modelo
164      if (SALIDA.equals("POLYBORI")) {
165       //Declarar los imports y las funciones
166       stringToFile("from polybori.PyPolyBoRi import *",true);
167       stringToFile("from polybori.gbcore import * ",true);
168       stringToFile("from polybori import * ",true);
169       if (BENCHMARK) stringToFile("import time ",true);
170       stringToFile("",true);
171       stringToFile("def O(M,N): return Polynomial(M)*Polynomial(N) ",
              true);
172       stringToFile("def NEG(M): return 1+Polynomial(M) ",true);
173       stringToFile("def Y(M,N): return NEG(O(NEG(M),NEG(N))) ",true);
174       stringToFile("",true);
175       //Declaro el anillo
```

```
176         int totalvariables=5;
177         stringToFile("declare_ring([Block(\"x\","+ts.
                getTotalVariablesRestriccionLogica()+")],globals());",true);
178         stringToFile("",true);
179         //Genero los polinomios de variable
180         String polvars="";
181         Vector<String> pares = ts.getRestriccionesVariables("(",")");
182         NUMPOLVARS=0;
183         for (int i=0;i<pares.size();i++){
184          stringToFile("s"+i+"=pow("+pares.get(i)+",2)␣+␣"+pares.get(i)+
                ";",true);
185          polvars+="s"+i;
186          if (i<pares.size()-1) polvars+=",";
187          //Indico el numero de polinomios de variables
188          NUMPOLVARS++;
189         }
190         stringToFile("",true);
191         //Declaro el ideal con los polinomios de restriccion
192         //stringToFile("",true);
193         //stringToFile("ideal i="+polvars+";",true);
194         //stringToFile("",true);
195         //Genero los polinomios de restriccion
196         String polres="";
197         pares = ts.getRestriccionesProposicional("(",")");
198         int cont=0;
```

```
199         NUMPOLRES=0;
200         for (int i=1;i<pares.size();i=i+2){
201          //System.out.println("polres"+cont+"="+pares.get(i-1)+"*"+
                 pares.get(i)+";");
202          stringToFile("ic"+cont+"=NEG(Y("+pares.get(i-1)+","+pares.get(
                 i)+"));",true);
203          polres+="ic"+cont;
204          if (i<pares.size()-2) polres+=",";
205          cont++;
206          NUMPOLRES++; //Indico el numero de polinomios de variables
207         }
208         stringToFile("",true);
209         stringToFile("J=["+polvars+","+polres+"];",true);
210         stringToFile("",true);
211
212        } else if (SALIDA.equals("SINGULAR")) {
213         Vector<String> pares = ts.getRestriccionesProposicional("[","]"
                );
214         //Declaro el anillo
215         int totalvariables=5;
216         stringToFile("ring r=2,(x(0.."+(ts.
                getTotalVariablesRestriccionLogica()-1)+")),lp;",true);
217         stringToFile("",true);
218         //Genero los polinomios de variable
219         String polvars="";
```

```
220         pares = ts.getRestriccionesVariables("(",")");
221         NUMPOLVARS=0;
222         for (int i=0;i<pares.size();i++){
223          stringToFile("poly s"+i+"=("+pares.get(i)+"^2)+"+pares.get(i
                 )+";",true);
224          polvars+="s"+i;
225          if (i<pares.size()-1) polvars+=",";
226          //Indico el numero de polinomios de variables
227         NUMPOLVARS++;
228         }
229         stringToFile("",true);
230         //Genero el ideal i
231         stringToFile("ideal J="+polvars+";",true);
232         stringToFile("J = std(J);",true);
233         stringToFile("",true);
234         //Incluyo las funciones
235         stringToFile("proc NEG(poly M) {return(reduce(1+M,J));} ",true)
                 ;
236         stringToFile("proc O(poly M,poly N) {return(reduce(M*N,J));}",
                 true);
237         stringToFile("proc Y(poly M,poly N) {return(NEG(O(NEG(M),NEG(N)
                 )));} ",true);
238         stringToFile("",true);
239         //Genero los polinomios de restriccion
240         String polres="";
```

```
241         pares = ts.getRestriccionesProposicional("(",")");
242         int cont=0;
243         NUMPOLRES=0;
244         for (int i=1;i<pares.size();i=i+2){
245          //System.out.println("polres"+cont+"="+pares.get(i-1)+"*"+
                 pares.get(i)+";");
246          stringToFile("poly ic"+cont+"=NEG(Y("+pares.get(i-1)+","+pares
                 .get(i)+"));",true);
247          polres+="ic"+cont;
248          if (i<pares.size()-2) polres+=",";
249          cont++;
250          NUMPOLRES++; //Indico el numero de polinomios de variables
251         }
252         stringToFile("",true);
253
254        } else if (SALIDA.equals("COCOA")) {
255         Vector<String> pares = ts.getRestriccionesProposicional("[","]"
                );
256         //Declaro el anillo
257         stringToFile("USE Z/(2)[x[0.."+(ts.
                getTotalVariablesRestriccionLogica()-1)+"]];",true);
258         stringToFile("",true);
259         //Genero los polinomios de variable
260         String polvars="";
261         pares = ts.getRestriccionesVariables("[","]");
```

```
262         NUMPOLVARS=0;
263         for (int i=0;i<pares.size();i++){
264          stringToFile("S"+i+":=("+pares.get(i)+"^2)␣+␣"+pares.get(i)+";
                 ",true);
265          polvars+="S"+i;
266          if (i<pares.size()-1) polvars+=",";
267          //Indico el numero de polinomios de variables
268          NUMPOLVARS++;
269         }
270         stringToFile("",true);
271         //Genereo el ideal i
272         stringToFile("MEMORY.J:=Ideal("+polvars+");",true);
273         //Incluyo las funciones
274         stringToFile("",true);
275
276         //Incluyo las funciones
277         stringToFile("Define␣NEG(M)␣RETURN␣NF(1+M,MEMORY.J);␣EndDefine;
                ",true);
278         stringToFile("Define␣O(M,N)␣RETURN␣NF(M*N,MEMORY.J);␣EndDefine;
                ",true);
279         stringToFile("Define␣Y(M,N)␣RETURN␣(NEG(O(NEG(M),NEG(N))));␣
                EndDefine;",true);
280         stringToFile("",true);
281         //Genero los polinomios de restriccion
282         String polres="";
```

```
283         pares = ts.getRestriccionesProposicional("[","]");
284         int cont=0;
285         NUMPOLRES=0;
286         for (int i=1;i<pares.size();i=i+2){
287          //System.out.println("polres"+cont+"="+pares.get(i-1)+"*"+
                 pares.get(i)+";");
288          stringToFile("IC"+cont+":=NEG(Y("+pares.get(i-1)+","+pares.get
                 (i)+"));",true);
289          polres+="IC"+cont;
290          if (i<pares.size()-2) polres+=",";
291          cont++;
292          NUMPOLRES++; //Indico el numero de polinomios de variables
293         }
294         stringToFile("",true);
295        }
296
297       } else if (MODELO.equals("NUEVOALGEBRAICO")) {
298        if (SALIDA.equals("SINGULAR")) {
299
300         stringToFile("ring␣r="+ts.getMayorPrimo()+",(x(0.."+(ts.
               getNumVars()-1)+")),lp;␣",true);
301         stringToFile("",true);
302         //Genero los polvars
303         String auxpolvars="",polvars="";
304         NUMPOLVARS=0;
```

```
305         for (int i=0;i<ts.getNumVars();i++){
306          auxpolvars+="poly s"+i+"=";
307          polvars+="s"+i;
308          if (i<ts.getNumVars()-1) polvars+=",";
309          for (int v=ts.getNumValoresDeVariable(i)-1;v>=0;v--){
310           auxpolvars+="(x("+i+")-"+v+")";
311           if (v>0) auxpolvars+="*";
312          }
313          auxpolvars+=";\n";
314          NUMPOLVARS++;
315         }
316         stringToFile(auxpolvars,true);
317         //Genero el ideal J
318         stringToFile("ideal J="+polvars+";",true);
319         stringToFile("J = std(J);",true);
320         //Incluyo las funciones
321         stringToFile("",true);
322         stringToFile("proc NEG(poly M){return(reduce(M^"+(ts.
                getMayorPrimo()-1)+"-1,J));}",true);
323         stringToFile("proc O(poly M,poly N){return(reduce(M*N,J));}",
                true);
324         stringToFile("",true);
325
326
327        } else if (SALIDA.equals("COCOA")) {
```

```
328
329         stringToFile("USE Z/("+ts.getMayorPrimo()+")[x[0.."+(ts.
                getNumVars()-1)+"]];",true);
330         stringToFile("",true);
331         //Genero los polvars
332         String auxpolvars="",polvars="";
333         NUMPOLVARS=0;
334         for (int i=0;i<ts.getNumVars();i++){
335          auxpolvars+="S"+i+":=";
336          polvars+="S"+i;
337          if (i<ts.getNumVars()-1) polvars+=",";
338          for (int v=ts.getNumValoresDeVariable(i)-1;v>=0;v--){
339           auxpolvars+="(x["+i+"]-"+v+")";
340           if (v>0) auxpolvars+=" * ";
341          }
342          auxpolvars+=";\n";
343          NUMPOLVARS++;
344         }
345         stringToFile(auxpolvars,true);
346         stringToFile("",true);
347         //Genereo el ideal i
348         stringToFile("MEMORY.J:=Ideal("+polvars+");",true);
349         //Incluyo las funciones
350         stringToFile("",true);
351
```

```
352         stringToFile("Define NEG(M) RETURN NF(M^"+(ts.getMayorPrimo()
                -1)+"-1,MEMORY.J); EndDefine;",true);
353         stringToFile("Define O(M,N) RETURN NF(M*N,MEMORY.J); EndDefine;
                ",true);
354         stringToFile("",true);
355        }
356       }
357      } catch (Exception e) {
358       //e.printStackTrace();
359       System.exit(0);
360      }
361
362    }
363
364
365
366
367    public static void stringToFile(String msg,boolean append) {
368     try{
369       String FILE_CODE="";
370
371       String LOG_POLYBORI="semlpb_polybori.txt";
372       String LOG_SINGULAR="semlpb_singular.txt";
373       String LOG_COCOA="semlpb_cocoa.txt";
374       String NMA_SINGULAR="semcva_singular.txt";
```

```
375     String NMA_COCOA="semcva_cocoa.txt";
376
377     if (MODELO.equals("PROPOSICIONAL")) {
378      if (SALIDA.equals("POLYBORI")) {
379       FILE_CODE=LOG_POLYBORI;
380      } else if (SALIDA.equals("SINGULAR")) {
381       FILE_CODE=LOG_SINGULAR;
382      } else if (SALIDA.equals("COCOA")) {
383       FILE_CODE=LOG_COCOA;
384      }
385     } else if (MODELO.equals("NUEVOALGEBRAICO")) {
386      if (SALIDA.equals("SINGULAR")) {
387       FILE_CODE=NMA_SINGULAR;
388      } else if (SALIDA.equals("COCOA")) {
389       FILE_CODE=NMA_COCOA;
390      }
391     }
392     //Indico el fichero de salida en una variable global para luego
393     //indicar donde se genero el codigo al usuario
394     OUTPUT_FILE=FILE_CODE;
395     FileWriter fstream = new FileWriter(FILE_CODE,append);
396     BufferedWriter out = new BufferedWriter(fstream);
397     out.write(msg+"\n");
398     out.close();
399    }catch (Exception e){
```

```
400       System.err.println("Error: " + e.getMessage());
401     }
402   }
403
404
405
406   public static void calcularbasegroebner(){
407     if (MODELO.equals("PROPOSICIONAL")) {
408       if (SALIDA.equals("POLYBORI")) {
409         //Genero los polvars
410         String polvars="";
411         for (int i=0;i<NUMPOLVARS;i++){
412           polvars+="v"+i;
413           if (i<NUMPOLVARS-1) polvars+=",";
414           else if (i==NUMPOLVARS-1) polvars+=",";
415         }
416         //Genero los polres
417         String polres="";
418         for (int i=0;i<NUMPOLRES;i++){
419           polres+="ic"+i;
420           if (i<NUMPOLRES-1) polres+=",";
421           else if (i==NUMPOLRES-1) polres+=",";
422         }
423         //Genero los polinputs
424         String polinputs="";
```

```
425         for (int i=0;i<NUMREGLASINPUT;i++){
426          polinputs+="i"+i;
427          if (i<NUMREGLASINPUT-1) polinputs+=",";
428          else if (i==NUMREGLASINPUT-1) polinputs+="";
429         }
430         //Genero los polinomios de reglas
431         String polreglas="";
432         for (int i=0;i<NUMREGLAS;i++){
433          polreglas+="r"+i;
434          if (i<NUMREGLAS-1) polreglas+=",";
435         }
436         stringToFile("",true);
437         if(FLAGEXISTEENTRADA) stringToFile("I=["+polinputs+"];",true);
438         stringToFile("K=["+polreglas+"];",true);
439         stringToFile("",true);
440         //Creo la llamada
441         if (BENCHMARK) stringToFile("a=time.time();",true);
442
443         if(FLAGEXISTEENTRADA) stringToFile("gb=groebner_basis(J+K+I,
                heuristic=False);",true);
444         else stringToFile("gb=groebner_basis(J+K,heuristic=False);",
                true);
445
446         if (BENCHMARK)
447          stringToFile("print time.time()-a;",true);
```

```
448         else {
449          // ts.getNumSoluciones()-1 => debido a la optimizacion el
                 ultimo valor
450          //           se sustituye por el negado del resto
451          stringToFile("nf=eliminate(gb)[1];",true);
452          for (int s=0;s<ts.getNumSoluciones()-1;s++)
453           stringToFile("print(nf(x("+s+")));",true);
454         }
455
456        } else if (SALIDA.equals("SINGULAR")) {
457         //Genero los polvars
458         String polvars="";
459         for (int i=0;i<NUMPOLVARS;i++){
460          polvars+="v"+i;
461          if (i<NUMPOLVARS-1) polvars+=",";
462          else if (i==NUMPOLVARS-1) polvars+=",";
463         }
464         //Genero los polres
465         String polres="";
466         for (int i=0;i<NUMPOLRES;i++){
467          polres+="ic"+i;
468          if (i<NUMPOLRES-1) polres+=",";
469          else if (i==NUMPOLRES-1) polres+=",";
470         }
471         //Genero los polinputs
```

```
472         String polinputs="";
473         for (int i=0;i<NUMREGLASINPUT;i++){
474          polinputs+="i"+i;
475          if (i<NUMREGLASINPUT-1) polinputs+=",";
476          else if (i==NUMREGLASINPUT-1) polinputs+="";
477         }
478         //Genero los polinomios de reglas
479         String polreglas="";
480         for (int i=0;i<NUMREGLAS;i++){
481          polreglas+="r"+i;
482          if (i<NUMREGLAS-1) polreglas+=",";
483         }
484         stringToFile("ideal␣K="+polres+polreglas+";",true);
485         //stringToFile("K = std(K);",true);
486         if(FLAGEXISTEENTRADA) stringToFile("ideal␣I="+polinputs+";",
                true);
487         //if(FLAGEXISTEENTRADA) stringToFile("I = std(I);",true);
488         stringToFile("",true);
489         //Creao la llamada
490         if (BENCHMARK) {
491          stringToFile("int␣t=0;\ntimer=0;",true);
492          stringToFile("system(\"--ticks-per-sec\",1000);",true);
493          stringToFile("t=timer;",true);
494         }
```

```
495         if(FLAGEXISTEENTRADA) stringToFile("ideal BASE=slimgb(J+K+I);",
                true);
496         else stringToFile("ideal BASE=slimgb(J+K);",true);
497         if (BENCHMARK)
498          stringToFile("print(timer-t);",true);
499         else {
500          stringToFile("ideal BASESTD=std(BASE); ",true);
501          // ts.getNumSoluciones()-1 => debido a la optimizacion el
                ultimo valor
502          //           se sustituye por el negado del resto
503          for (int s=0;s<ts.getNumSoluciones()-1;s++)
504           stringToFile("print(reduce(x("+s+"),BASESTD));",true);
505         }
506        } else if (SALIDA.equals("COCOA")) {
507         //Genero los polvars
508         String polvars="";
509         for (int i=0;i<NUMPOLVARS;i++){
510          polvars+="V"+i;
511          if (i<NUMPOLVARS-1) polvars+=",";
512          else if (i==NUMPOLVARS-1) polvars+=",";
513         }
514         //Genero los polres
515         String polres="";
516         for (int i=0;i<NUMPOLRES;i++){
517          polres+="IC"+i;
```

```
518         if (i<NUMPOLRES-1) polres+=",";
519       }
520       //Genero los polinputs
521       String polinputs="";
522       for (int i=0;i<NUMREGLASINPUT;i++){
523        polinputs+="I"+i;
524        if (i<NUMREGLASINPUT-1) polinputs+=",";
525       }
526       //Genero los polinomios de reglas
527       String polreglas="";
528       for (int i=0;i<NUMREGLAS;i++){
529        polreglas+="R"+i;
530        if (i<NUMREGLAS-1) polreglas+=",";
531       }
532       //Creao la llamada
533       if(FLAGEXISTEENTRADA) stringToFile("I:=Ideal("+polinputs+");",
              true);
534       if (polres.length()>0)
535        stringToFile("K:=Ideal("+polres+","+polreglas+");",true);
536       else
537        stringToFile("K:=Ideal("+polreglas+");",true);
538       //Obtengo todas las "salidas solucion"
539       // ts.getNumSoluciones()-1 => debido a la optimizacion el
              ultimo valor
540       //          se sustituye por el negado del resto
```

```
541         if (BENCHMARK) {
542          if(FLAGEXISTEENTRADA) stringToFile("Time T:=GBasis(MEMORY.J+K+
                 I);",true);
543          else stringToFile("Time T:=GBasis(MEMORY.J+K);",true);
544         } else {
545          for (int s=0;s<ts.getNumSoluciones()-1;s++)
546           if(FLAGEXISTEENTRADA) stringToFile("NF(x["+s+"],MEMORY.J+K+I)
                  ;",true);
547           else stringToFile("NF(x["+s+"],MEMORY.J+K);",true);
548         }
549
550        }
551       } else if (MODELO.equals("NUEVOALGEBRAICO")) {
552        if (SALIDA.equals("SINGULAR")) {
553         //Genero los polvars
554         String polvars="";
555         for (int i=0;i<NUMPOLVARS;i++){
556          polvars+="v"+i;
557          if (i<NUMPOLVARS-1) polvars+=",";
558          else if (i==NUMPOLVARS-1) polvars+=",";
559         }
560         //Genero los polinputs
561         String polinputs="";
562         for (int i=0;i<NUMREGLASINPUT;i++){
563          polinputs+="i"+i;
```

```
564         if (i<NUMREGLASINPUT-1) polinputs+=",";
565         else if (i==NUMREGLASINPUT-1) polinputs+="";
566       }
567       //Genero los polinomios de reglas
568       String polreglas="";
569       for (int i=0;i<NUMREGLAS;i++){
570        polreglas+="r"+i;
571        if (i<NUMREGLAS-1) polreglas+=",";
572       }
573       stringToFile("",true);
574       stringToFile("ideal␣K="+polreglas+";",true);
575       //stringToFile("K = std(K);",true);
576       if(FLAGEXISTEENTRADA) stringToFile("ideal␣I="+polinputs+";",
              true);
577       //if(FLAGEXISTEENTRADA) stringToFile("I = std(I);",true);
578       //Creao la llamada
579       stringToFile("",true);
580       //Creao la llamada
581       if (BENCHMARK) {
582        stringToFile("int␣t=0;\ntimer=0;",true);
583        stringToFile("system(\"--ticks-per-sec\",1000);",true);
584        stringToFile("t=timer;",true);
585       }
586       if(FLAGEXISTEENTRADA) stringToFile("ideal␣BASE=slimgb(J+K+I);",
              true);
```

```
587         else stringToFile("ideal␣BASE=slimgb(J+K);",true);
588         if (BENCHMARK) {
589          stringToFile("print(timer-t);",true);
590         } else {
591          stringToFile("ideal␣BASESTD=std(BASE);␣",true);
592          stringToFile("",true);
593          stringToFile("print(reduce(x(0),BASESTD));",true);
594         }
595        } else if (SALIDA.equals("COCOA")) {
596         //Genero los polvars
597         String polvars="";
598         for (int i=0;i<NUMPOLVARS;i++){
599          polvars+="V"+i;
600          if (i<NUMPOLVARS-1) polvars+=",";
601          else if (i==NUMPOLVARS-1) polvars+=",";
602         }
603         //Genero los polinputs
604         String polinputs="";
605         for (int i=0;i<NUMREGLASINPUT;i++){
606          polinputs+="I"+i;
607          if (i<NUMREGLASINPUT-1) polinputs+=",";
608         }
609         //Genero los polinomios de reglas
610         String polreglas="";
611         for (int i=0;i<NUMREGLAS;i++){
```

```
612         polreglas+="R"+i;
613         if (i<NUMREGLAS-1) polreglas+=",";
614        }
615        //Creao la llamada
616        stringToFile("",true);
617        stringToFile("K:=Ideal("+polreglas+");",true);
618        if(FLAGEXISTEENTRADA) stringToFile("I:=Ideal("+polinputs+");",
               true);
619        stringToFile("",true);
620        if (BENCHMARK) {
621         if(FLAGEXISTEENTRADA) stringToFile("Time␣T:=GBasis(MEMORY.J+K+
                1);",true);
622         else stringToFile("Time␣T:=GBasis(MEMORY.J+K);",true);
623        } else
624         if(FLAGEXISTEENTRADA) stringToFile("NF(x[0],MEMORY.J+K+1);",
               true);
625         else stringToFile("NF(x[0],MEMORY.J+K);",true);
626       }
627      }
628    }
629
630
631    public static void mostrarWarnings(){
632     Vector<String> warnings = ts.getWarningVariablesNoUsadas();
633     if (warnings.size()>0) {
```

```
634     System.out.println("Warning! Variables no usadas: "+warnings);
635    }
636   }
637
638   /*
639   BEGIN
```

```
640   */
641   //Inico del programa
642   public static void main (String[] args)  {
643
644   try {
645
646   //Compruebo el numero de parametros que se le pasan al compilador
647   if (args.length>4 || args.length<2) {
648     mostrar_error_de_parametros();
649   }
650   //Clasifico las opciones, indicando error si no estan dentro de las
          opciones
651   //validadas
652   if (args[0].equals("1")) {
653    MODELO="PROPOSICIONAL";
654    if (args[1].equals("1")) {
655     SALIDA="POLYBORI";
```

```
656     } else if (args[1].equals("2")) {
657       SALIDA="SINGULAR";
658     } else if (args[1].equals("3")) {
659       SALIDA="COCOA";
660     } else {
661     mostrar_error_de_parametros();
662     }
663   }
664   else if (args[0].equals("2")) {
665    MODELO="NUEVOALGEBRAICO";
666     if (args[1].equals("1")) {
667       SALIDA="SINGULAR";
668     } else if (args[1].equals("2")) {
669       SALIDA="COCOA";
670     } else {
671     mostrar_error_de_parametros();
672     }
673   } else {
674     mostrar_error_de_parametros();
675   }
676
677   boolean mostrarTS=false;
678   BENCHMARK=false;
679
680   if (args.length==3) {
```

```
681     if (args[2].equals("-ts")) {
682       mostrarTS=true;
683     } else if (args[2].equals("-b")) {
684       BENCHMARK=true;
685     } else {
686       mostrar_error_de_parametros();
687     }
688   }
689
690   if (args.length==4) {
691     if (args[2].equals("-ts") && args[3]==null) {
692       mostrarTS=true;
693     } else if (args[2].equals("-b") && args[3]==null) {
694       BENCHMARK=true;
695     }
696     else if (args[2].equals("-b") && args[3].equals("-b")) {
697       BENCHMARK=true;
698     } else if (args[2].equals("-ts") && args[3].equals("-b")) {
699       BENCHMARK=true;
700       mostrarTS=true;
701     } else if (args[2].equals("-b") && args[3].equals("-ts")) {
702       BENCHMARK=true;
703       mostrarTS=true;
704     }
705     else {
```

```
706       mostrar_error_de_parametros();
707     }
708   }
709
710   //Creo la tabla de simbolos
711   ts = new TS();
712   //Inicio la estructura en donde ire guardando los polinomios
713   polinomios = new Vector<String>();
714   //Inicio la estructura en donde ire guardando los operadores de los
          polinomois
715   operadores = new Vector<Boolean>();
716   //Lanzo el proceso de compilacion
717   Traductor anLS = new Traductor(System.in);
718
719   anLS.unPrograma(ts);
720   if (mostrarTS)
721     System.out.println(ts.imprimirTS());
722   mostrarWarnings();
723
724
725   //Genero los ficheros de la tabla de simbolos correspondientes
726   //a la opcion del modelo elegido
727
728   if (MODELO.equals("PROPOSICIONAL")){
729     ts.salidaAFichero(0,"ts_mlp_1.txt",null);
```

```
730   } else if (MODELO.equals("NUEVOALGEBRAICO")){
731    ts.salidaAFichero(1,"ts_mcva_1.txt","ts_mcva_2.txt");
732   }
733
734   String msg_modelo_salida="";
735   if (MODELO.equals("PROPOSICIONAL")) msg_modelo_salida="Logico␣
          proposicional␣booleano";
736   else if (MODELO.equals("NUEVOALGEBRAICO")) msg_modelo_salida="'
          Concepto-Atributo-Valor'";
737
738   String msg_sistema_salida="";
739   if (SALIDA.equals("POLYBORI")) msg_sistema_salida="Polybori";
740   else if (SALIDA.equals("SINGULAR")) msg_sistema_salida="Singular";
741   else if (SALIDA.equals("COCOA")) msg_sistema_salida="Cocoa";
742
743   System.out.println("\nEl␣codigo␣se␣ha␣generado␣en␣el␣fichero␣"+
          OUTPUT_FILE.toUpperCase());
744   System.out.println("␣MODELO:␣␣"+msg_modelo_salida);
745   System.out.println("␣SISTEMA:␣"+msg_sistema_salida);
746
747   } catch( TokenMgrError tme ) {
748     //Para personlizar el error tocar en TokenMgrError.java
749     System.out.println(tme);
750     System.exit(0);
751   } catch(ParseException pee ) {
```

```
752      //Para personlizar el error tocar en ParseException.java
753      System.out.println(pee);
754      System.exit(0);
755    } catch (Exception e) {
756      System.out.println(e);
757      System.exit(0);
758    }
759
760    }
761  }
762
763
764 PARSER_END (Traductor)
765
766 void unPrograma(TS ts):
767  {}
768  {
769   "BEGIN" "BEGIN_VARS" (decvariable(ts))+  { try{ ts.generarIndices()
         ; }catch (Exception e){System.out.println(e); System.exit(0);}
         stringToFile("",false);} "END_VARS" {
         generarPolinomiosRestriccion();} "BEGIN_KB" (regla())*   "END_KB
         " {FLAGEXISTEENTRADA=false;} ["BEGIN_INPUT" input() "END_INPUT"]
          {calcularbasegroebner();} "END."
770  }
771
```

```
772 TOKEN_MGR_DECLS:
773  {
774   static int numLin = 0;
775   static String linea = "";
776   static void CommonTokenAction(Token pieza) {
777    linea = linea + pieza.image;
778   }
779  }
780 void decvariable(TS ts):
781  {
782   Token piezavariable=null,piezanumero=null;String valor="";
783   Vector<String> variablesEncadenadas = new Vector<String>();
784  }
785  {
786   piezavariable=<id>  ["[" piezanumero=<numero> "]"]
787   {
788    //Inserto la variable
789    try{
790     //Trato el numero de variables
791     int numvariables=1;
792     if (piezanumero!=null)
793      numvariables=Integer.parseInt(piezanumero.toString());
794     //Inserto la variable
795     //Encadeno la primera variable
796     variablesEncadenadas.add(piezavariable.toString());
```

```
797     ts.insertaVariable(piezavariable.toString(),piezavariable.
            toString(),numvariables);
798     //Ahora inserto todos los valores para la variable
799
800    } catch(Exception e){
801     System.out.println("Error en linea "+piezavariable.beginLine+" \n
            ");
802     System.out.println(e);
803     //e.printStackTrace();
804     System.exit(0);
805    }
806   }
807   //NUEVO, inserto las variables con los valores de la anterior si
        esque existen encadenadas
808   //
        ____________________________________________________________________

809   {    //Flag para saber si en las definiciones de variables se han
        encadenado
810    //si es asi guardare cual es la padre para mas tarde saber si
         puedo compararlas
811    boolean encadenadas=false;
812   }
813   (
814    "," piezavariable=<id>  ["[" piezanumero=<numero> "]"]
```

```
815     //Inserto la variable
816     {
817      variablesEncadenadas.add(piezavariable.toString());
818      encadenadas=true;
819      try{
820       //Trato el numero de variables
821       int numvariables=1;
822       if (piezanumero!=null)
823        numvariables=Integer.parseInt(piezanumero.toString());
824       //Inserto la variable
825       ts.insertaVariable(piezavariable.toString(),variablesEncadenadas
              .get(0),numvariables);
826       //Ahora inserto todos los valores para la variable
827      } catch(Exception e){
828       System.out.println("Error en linea "+piezavariable.beginLine+" \
              n");
829       System.out.println(e);
830       //e.printStackTrace();
831       System.exit(0);
832      }
833     }
834    )*
835
836    ":" valor=valor()
837    {
```

```
838     //Inserto el valor de la variable
839     try{
840      //Inserto los valores en cada pieza encadenada
841      for (int i=0;i<variablesEncadenadas.size();i++){
842       ts.insertaValorEnVariable(variablesEncadenadas.get(i).toString()
             ,valor);
843      }
844     } catch(Exception e){
845      System.out.println("Error en linea "+piezavariable.beginLine+" \n
            ");
846      System.out.println(e);
847      //e.printStackTrace();
848      System.exit(0);
849     }
850    }
851    (",") valor=valor()
852    {
853     //Inserto el valor de la variable
854     try{
855      //Inserto los valores en cada pieza encadenada
856      for (int i=0;i<variablesEncadenadas.size();i++){
857       ts.insertaValorEnVariable(variablesEncadenadas.get(i).toString()
             ,valor);
858      }
859     } catch(Exception e){
```

```
860      System.out.println("Error en linea "+piezavariable.beginLine+" \n
             ");
861      System.out.println(e);
862      //e.printStackTrace();
863      System.exit(0);
864     }
865    }
866    )* ";"
867  }
868 String valor():
869  {Token pieza;}
870  {
871   pieza=<id>
872   {
873    return pieza.toString();
874   }
875  }
876 void regla():
877  {
878   String resto="";
879   Vector<String> p0;
880   Vector<Boolean> p0ops;
881   boolean isExpresionComplejaLaPrimera=false;
882   p0 = new Vector<String>();
883   p0ops = new Vector<Boolean>();
```

```
884 }
885 {
886  try {
887  expresion()
888   {
889    p0=repols();
890    p0ops=reops();
891    isExpresionComplejaLaPrimera=FLAGEXPRESIONVALORCOMPLEJO;
892   }
893   [resto=restoregla()] ";"
894  {
895   //Trato la primera expresion
896   String ex1="";
897   //Miro si la primera expresion es compleja o no lo es para asi
          aplicarle
898   //la negacion a cada regla por ser antecedente, o no aplicar esta
          regla.
899   if (isExpresionComplejaLaPrimera){
900    //Llamamos a la cadena con el FLAG de antecedente a
         FLAGANTECEDENTE
901    //porque se negara todo dependiente si es antecedente o
         consecuente
902    //en la funcion acadena
903    ex1=acadena(p0,p0ops,FLAGANTECEDENTE);
904   } else {
```

```
905      //Generamos la cadena aplicando la negacion normal
906      ex1=acadena(p0,p0ops,FLAGANTECEDENTE);
907     }
908     //Aqui tambien mirar el tipo de modelo y el tipo de variable
909     if (MODELO.equals("PROPOSICIONAL")) {
910
911       if (SALIDA.equals("POLYBORI")) {
912        if (!resto.equals("")) {
913         stringToFile("r"+NUMREGLAS+":=O("+ex1+","+resto+");",true);
914        } else {
915         stringToFile("r"+NUMREGLAS+":="+ex1+";",true);
916        }
917       } else if (SALIDA.equals("SINGULAR")) {
918        if (!resto.equals("")) {
919         stringToFile("poly r"+NUMREGLAS+":=O("+ex1+","+resto+");",true)
                 ;
920        } else {
921         stringToFile("poly r"+NUMREGLAS+":="+ex1+";",true);
922        }
923       } else if (SALIDA.equals("COCOA")) {
924        if (!resto.equals("")) {
925         stringToFile("R"+NUMREGLAS+":=O("+ex1+","+resto+");",true);
926        } else {
927         stringToFile("R"+NUMREGLAS+":="+ex1+";",true);
928        }
```

```
929         }
930
931       } else if (MODELO.equals("NUEVOALGEBRAICO")) {
932
933         if (SALIDA.equals("SINGULAR")) {
934           if (!resto.equals("")) {
935             stringToFile("poly r"+NUMREGLAS+"=O("+ex1+","+resto+");",true)
                    ;
936           } else {
937             stringToFile("poly r"+NUMREGLAS+"="+ex1+";",true);
938           }
939         } else if (SALIDA.equals("COCOA")) {
940           if (!resto.equals("")) {
941             stringToFile("R"+NUMREGLAS+":=O("+ex1+","+resto+");",true);
942           } else {
943             stringToFile("R"+NUMREGLAS+":="+ex1+";",true);
944           }
945         }
946
947       }
948     NUMREGLAS++;
949     }
950     }
951     catch (Exception e){
952       System.out.println(e);
```

```
953     System.exit(0);
954    }
955   }
956  String restoregla():
957   {
958    Vector<String> p1,p2,p3;
959    p1 = new Vector<String>();
960    p2 = new Vector<String>();
961    p3 = new Vector<String>();
962    Vector<Boolean> p1ops,p2ops,p3ops;
963    p1ops = new Vector<Boolean>();
964    p2ops = new Vector<Boolean>();
965    p3ops = new Vector<Boolean>();
966   }
967   {
968
969    //PRIMERA FORMA DE LAS REGLAS
970    {FLAGANTECEDENTE=false;}
971    consecuente() {p1=repols();p1ops=reops();} (consecuente() {
          aniadepols(p1);aniadeops(p1ops);})*
972    { return acadena(p1,p1ops,false); }
973
974
975    //SEGUNDA FORMA DE LA REGLAS
976    |
```

```
977    {FLAGANTECEDENTE=true;}
978     (antecedente() {aniadepols(p2);aniadeops(p2ops);} )*
979     "->" expresion() {p3=repols();p3ops=reops();} (consecuente() {
            aniadepols(p3);aniadeops(p3ops);})*
980     {
981       if (p2.size()>0) {
982        return "O("+acadena(p2,p2ops,true)+","+acadena(p3,p3ops,false)+"
              )";
983       } else {
984         return acadena(p3,p3ops,false);
985       }
986
987     }
988  }
989 void consecuente():
990  { }
991  {
992   //("OR" | "O") expresion()
993   ("OR") expresion()
994  }
995 void antecedente():
996  { }
997  {
998   //("AND" | "Y") expresion()
999   ("AND") expresion()
```

```
1000  }
1001 void expresion():
1002  {Token piezavariable=null;String[] v1,v2;String aux="";boolean
          operador;}
1003  {
1004   "(" v1=variable() operador=operador() v2=variable() ")"
1005   //En este punto tengo la variable y el valor o variable asignados.
1006   //Obtengo la primera variable y miro si existe "el segundo" como
          valor
1007   //o como variable y obtengo la informacion.
1008   {
1009   try{
1010
1011    //ASIGNACION VARIABLE-VALOR
1012    if (!ts.existeVariable(v2[0])){
1013    //marco la variable para luego mostrar warnings si no ha sido
          usada
1014    ts.marcaVariable(v1[0]+"["+v1[1]+"]");
1015
1016    FLAGEXPRESIONVALORCOMPLEJO=false;
1017    //Tenemos que comprobar que se este accediendo a la variable
          correctamente
1018    //si no no podremos proseguir. Es una medida de restriccion para
          clarificar
1019    //el codigo.
```

```
1020
1021    // INFORMACION ESTRUCTURA DE DATOS
1022    // v[0] Contiene el nombre de la variable
1023    // v[1] Contiene el indice de acceso a ella
1024    // Comprobamos que si la variable esta declarada como array, solo
            se permita el accceso a ella
1025    // desde nombrevar[indice] y no como: nombrevar
1026    ts.comprubaaccesoavariable(v1[0],v1[1]);
1027
1028     if (MODELO.equals("PROPOSICIONAL")) {
1029
1030      if (SALIDA.equals("POLYBORI") || (SALIDA.equals("SINGULAR"))) {
1031       int[] variable = ts.getIdentificadorReal(v1[0],Integer.parseInt
              (v1[1]),v2[0]);
1032       //Optimizacion de los ultimos valores
1033       String variableOptimizada="x("+variable[0]+")";
1034       //Compruebo si alguna de las dos variables es el ultimo valor
1035       if (variable[2]==1) {
1036        variableOptimizada="1+";
1037        Vector<String> ids1=ts.getIdentificadoresRealOptimizado(v1[0],
              Integer.parseInt(v1[1]));
1038        for (int c=0;c<ids1.size();c++){
1039         variableOptimizada +="x("+ids1.get(c)+")";
1040         if (c<ids1.size()-1) variableOptimizada+="*";
1041        }
```

```
1042        }
1043        polinomios.add(variableOptimizada);
1044        //FLAGNEGARTODO a false siempre en esta posicion
1045        FLAGNEGARTODO=false;
1046        operadores.add(operador);
1047       } else if (SALIDA.equals("COCOA")) {
1048        int[] variable = ts.getIdentificadorReal(v1[0],Integer.parseInt
              (v1[1]),v2[0]);
1049        //Optimizacion de los ultimos valores
1050        String variableOptimizada="x["+variable[0]+"]";
1051        //Compruebo si alguna de las dos variables es el ultimo valor
1052        if (variable[2]==1) {
1053         variableOptimizada="1+";
1054         Vector<String> ids1=ts.getIdentificadoresRealOptimizado(v1[0],
               Integer.parseInt(v1[1]));
1055         for (int c=0;c<ids1.size();c++){
1056          variableOptimizada +="x["+ids1.get(c)+"]";
1057          if (c<ids1.size()-1) variableOptimizada+="*";
1058         }
1059        }
1060        polinomios.add(variableOptimizada);
1061        //FLAGNEGARTODO a false siempre en esta posicion
1062        FLAGNEGARTODO=false;
1063        operadores.add(operador);
1064       }
```

```
1065
1066      } else if (MODELO.equals("NUEVOALGEBRAICO")) {
1067
1068        if (SALIDA.equals("SINGULAR")) {
1069          int[] variable = ts.getIdentificadorRealAlgebraico(v1[0],
                  Integer.parseInt(v1[1]),v2[0]);
1070          polinomios.add("(x("+variable[0]+")-"+variable[1]+")");
1071          operadores.add(operador);
1072          //FLAGNEGARTODO a false siempre en esta posicion
1073          FLAGNEGARTODO=false;
1074        } else if (SALIDA.equals("COCOA")) {
1075          int[] variable = ts.getIdentificadorRealAlgebraico(v1[0],
                  Integer.parseInt(v1[1]),v2[0]);
1076          polinomios.add("(x["+variable[0]+"]-"+variable[1]+")");
1077          operadores.add(operador);
1078          //FLAGNEGARTODO a false siempre en esta posicion
1079          FLAGNEGARTODO=false;
1080        }
1081
1082      }
1083
1084
1085     //ASIGNACION VARIABLE-VARIABLE
1086     } else {
```

```
1087        // Ya que la expresion trabaja con dos variables, tendre que
                realizar la comprobacion semantica
1088        // que solo permite comparar variables cuando han sido definidas
                en la misma linea del fichero
1089        // fuente, si no, pararemos la ejecucion y mostraremos un error
                semantico.
1090        // Para este proposito usaremos un atributo de la clave Variable
                del paquete TablaSimbolosCompilador,
1091        // en este atributo tenemos el valor de la variable padre, si
                coinciden, significa que podemos comparar
1092        // ambas variables ya que estan definidas en la misma linea.
1093        // La funcion ts.isComparables lanzara una excepcion si las
                variables no pueden ser comparadas.
1094        ts.isComparables(v1[0],v2[0]);
1095
1096        //marco las variables para luego mostrar warnings si no han sido
                usadas
1097        ts.marcaVariable(v1[0]+"["+v1[1]+"]");
1098        ts.marcaVariable(v2[0]+"["+v2[1]+"]");
1099
1100        FLAGEXPRESIONVALORCOMPLEJO=true;
1101
1102        if (MODELO.equals("PROPOSICIONAL")) {
1103          if (SALIDA.equals("POLYBORI") || SALIDA.equals("SINGULAR")) {
1104            String auxvalorcomplejo="";
```

```
1105        Vector<String> paraencadenarenors=new Vector<String>();
1106        Vector<String> valorescomun = ts.getValoresEnComun(v1[0],v2[0])
                ;
1107        for (int i=0;i<valorescomun.size();i++){
1108         int[] variable1 = ts.getIdentificadorReal(v1[0],Integer.
                parseInt(v1[1]),valorescomun.get(i));
1109         int[] variable2 = ts.getIdentificadorReal(v2[0],Integer.
                parseInt(v2[1]),valorescomun.get(i));
1110         //Optimizacion de los ultimos valores
1111         String variable1Optimizada="x("+variable1[0]+")";
1112         String variable2Optimizada="x("+variable2[0]+")";
1113         //Compruebo si alguna de las dos variables es el ultimo valor
1114         if (variable1[2]==1) {
1115          variable1Optimizada="1+";
1116          Vector<String> ids1=ts.getIdentificadoresRealOptimizado(v1
                 [0],Integer.parseInt(v1[1]));
1117          for (int c=0;c<ids1.size();c++){
1118           variable1Optimizada +="x("+ids1.get(c)+")";
1119           if (c<ids1.size()-1) variable1Optimizada+="*";
1120          }
1121         }
1122         if (variable2[2]==1) {
1123          variable2Optimizada="1+";
1124          Vector<String> ids2=ts.getIdentificadoresRealOptimizado(v2
                 [0],Integer.parseInt(v2[1]));
```

```
1125          for (int c=0;c<ids2.size();c++){
1126           variable2Optimizada +="x("+ids2.get(c)+")";
1127           if (c<ids2.size()-1) variable2Optimizada+="*";
1128          }
1129         }
1130         paraencadenarenors.add("Y("+variable1Optimizada+""+","+
                variable2Optimizada+")");
1131        }
1132        //EXPRESIONES COMPUESTAS
1133        //Si no tiene valores en comun, devuelvo un 1 (false)
1134        if (paraencadenarenors.size()==0){
1135         polinomios.add("1");
1136         operadores.add(operador);
1137        } else {
1138         //Si tiene valores en comun los encadeno recursivamente en o(y
                (a,{o(..)})) y devuelvo
1139         //la expresion compleja
1140            if (paraencadenarenors.size()>1)
1141          auxvalorcomplejo=encadenaOrSimple(paraencadenarenors,0);
1142         else
1143          auxvalorcomplejo=paraencadenarenors.get(0);
1144         polinomios.add(auxvalorcomplejo);
1145         //Fuerzo el operador a true, es decir ==, el negado se lo
                pongo
1146         //luego en el nievel regla() si el FLAGNEGARTODO esta activo
```

```
1147          operadores.add(operador);
1148         }
1149         if(operador) FLAGNEGARTODO=false;
1150         else     FLAGNEGARTODO=true;
1151
1152
1153        } else if (SALIDA.equals("COCOA")) {
1154         String auxvalorcomplejo="";
1155         Vector<String> paraencadenarenors=new Vector<String>();
1156         Vector<String> valorescomun = ts.getValoresEnComun(v1[0],v2[0])
                 ;
1157         for (int i=0;i<valorescomun.size();i++){
1158          int[] variable1 = ts.getIdentificadorReal(v1[0],Integer.
                 parseInt(v1[1]),valorescomun.get(i));
1159          int[] variable2 = ts.getIdentificadorReal(v2[0],Integer.
                 parseInt(v2[1]),valorescomun.get(i));
1160          //Optimizacion de los ultimos valores
1161          String variable1Optimizada="x["+variable1[0]+"]";
1162          String variable2Optimizada="x["+variable2[0]+"]";
1163          //Compruebo si alguna de las dos variables es el ultimo valor
1164          if (variable1[2]==1) {
1165           variable1Optimizada="1+";
1166           Vector<String> ids1=ts.getIdentificadoresRealOptimizado(v1
                  [0],Integer.parseInt(v1[1]));
1167           for (int c=0;c<ids1.size();c++){
```

```
1168            variable1Optimizada +="x["+ids1.get(c)+"]";
1169            if (c<ids1.size()-1) variable1Optimizada+="*";
1170           }
1171          }
1172          if (variable2[2]==1) {
1173           variable2Optimizada="1+";
1174           Vector<String> ids2=ts.getIdentificadoresRealOptimizado(v2
                  [0],Integer.parseInt(v2[1]));
1175           for (int c=0;c<ids2.size();c++){
1176            variable2Optimizada +="x["+ids2.get(c)+"]";
1177            if (c<ids2.size()-1) variable2Optimizada+="*";
1178           }
1179          }
1180          paraencadenarenors.add("Y("+variable1Optimizada+""+","+
                 variable2Optimizada+")");
1181         }
1182         //EXPRESIONES COMPUESTAS
1183         //Si no tiene valores en comun, devuelvo un 1 (false)
1184         if (paraencadenarenors.size()==0){
1185          polinomios.add("1");
1186          operadores.add(operador);
1187         } else {
1188          //Si tiene valores en comun los encadeno recursivamente en o(y
                 (a,{o(..)})) y devuelvo
1189          //la expresion compleja
```

```
1190            if (paraencadenarenors.size()>1)
1191             auxvalorcomplejo=encadenaOrSimple(paraencadenarenors,0);
1192            else
1193             auxvalorcomplejo=paraencadenarenors.get(0);
1194            polinomios.add(auxvalorcomplejo);
1195            //Fuerzo el operador a true, es decir ==, el negado se lo
                   pongo
1196            //luego en el nievel regla() si el FLAGNEGARTODO esta activo
1197            operadores.add(operador);
1198           }
1199           if(operador) FLAGNEGARTODO=false;
1200           else    FLAGNEGARTODO=true;
1201          }
1202
1203         } else if(MODELO.equals("NUEVOALGEBRAICO")) {
1204          if (SALIDA.equals("SINGULAR")) {
1205           int[] aux1=ts.getIdentificadorRealAlgebraicoSinValor(v1[0],
                   Integer.parseInt(v1[1]));
1206           int[] aux2=ts.getIdentificadorRealAlgebraicoSinValor(v2[0],
                   Integer.parseInt(v2[1]));
1207           String poly1="x("+aux1[0]+")";
1208           String poly2="x("+aux2[0]+")";
1209           polinomios.add(poly1+"-"+poly2);
1210           operadores.add(operador);
1211           //FLAGNEGARTODO a false siempre en esta posicion
```

```
1212          FLAGNEGARTODO=false;
1213        } else if (SALIDA.equals("COCOA")) {
1214          int[] aux1=ts.getIdentificadorRealAlgebraicoSinValor(v1[0],
                  Integer.parseInt(v1[1]));
1215          int[] aux2=ts.getIdentificadorRealAlgebraicoSinValor(v2[0],
                  Integer.parseInt(v2[1]));
1216          String poly1="x["+aux1[0]+"]";
1217          String poly2="x["+aux2[0]+"]";
1218          polinomios.add(poly1+"-"+poly2);
1219          operadores.add(operador);
1220          //FLAGNEGARTODO a false siempre en esta posicion
1221          FLAGNEGARTODO=false;
1222        }
1223       }
1224
1225     }
1226    } catch(Exception e){
1227       System.out.println(e);
1228       //e.printStackTrace();
1229       System.exit(0);
1230    }
1231    }
1232  }
1233 boolean operador():
1234  {}
```

```
1235  {
1236   "=" {return true;}
1237   //| ("!=" | "<>") {return false;}
1238   | "<>" {return false;}
1239  }
1240 String[] variable():
1241  {Token identificador=null;Token idnumero=null;}
1242  {
1243   identificador=<id> ["[" idnumero=<numero> "]"]
1244   {
1245    //Devuelvo el identificado y el numero del indice si lo tuviera
1246    String[] res = new String[2];
1247    String numero="0";
1248    if (idnumero!=null) numero=idnumero.toString();
1249    res[0]=identificador.toString();
1250    //Si se ha declarado la variable como "var" y no como "var[1]", lo
          indicamos
1251    //por codigo, es decir, establecemos a 1 el indice de acceso.
1252    if (idnumero==null) numero="1";
1253    res[1]=numero;
1254    return res;
1255   }
1256  }
1257 void input():
1258  { Vector<String> p1,p2,p3;
```

```
1259    p1 = new Vector<String>();
1260    Vector<Boolean> p1ops,p2ops,p3ops;
1261    p1ops = new Vector<Boolean>();
1262  }
1263  {
1264    {FLAGEXISTEENTRADA=true;}
1265    expresion()
1266    {
1267     p1=repols();
1268     p1ops=reops();
1269     stringToFile("",true);
1270     if (MODELO.equals("PROPOSICIONAL")) {
1271      if (SALIDA.equals("POLYBORI")) {
1272       stringToFile("i"+NUMREGLASINPUT+"="+acadena(p1,p1ops,false)+";"
              ,true);
1273      } else if(SALIDA.equals("SINGULAR")){
1274       stringToFile("poly i"+NUMREGLASINPUT+"="+acadena(p1,p1ops,false
              )+";",true);
1275      } else if(SALIDA.equals("COCOA")){
1276       stringToFile("I"+NUMREGLASINPUT+":="+acadena(p1,p1ops,false)+";
              ",true);
1277      }
1278     } else if (MODELO.equals("NUEVOALGEBRAICO")) {
1279      if (SALIDA.equals("SINGULAR")) {
```

```
1280          stringToFile("poly_i"+NUMREGLASINPUT+"="+acadena(p1,p1ops,false
                  )+";",true);
1281         } else if (SALIDA.equals("COCOA")) {
1282          stringToFile("I"+NUMREGLASINPUT+":="+acadena(p1,p1ops,false)+";
                  ",true);
1283         }
1284        }
1285        NUMREGLASINPUT++;
1286       }
1287       ("," expresion()
1288       {
1289        p1=repols();
1290        p1ops=reops();
1291        if (MODELO.equals("PROPOSICIONAL")) {
1292         if (SALIDA.equals("POLYBORI")) {
1293          stringToFile("i"+NUMREGLASINPUT+"="+acadena(p1,p1ops,false)+";"
                  ,true);
1294         } else if(SALIDA.equals("SINGULAR")){
1295          stringToFile("poly_i"+NUMREGLASINPUT+"="+acadena(p1,p1ops,false
                  )+";",true);
1296         } else if(SALIDA.equals("COCOA")){
1297          stringToFile("I"+NUMREGLASINPUT+":="+acadena(p1,p1ops,false)+";
                  ",true);
1298         }
1299        } else if (MODELO.equals("NUEVOALGEBRAICO")) {
```

```
1300        if (SALIDA.equals("SINGULAR")) {
1301         stringToFile("poly_i"+NUMREGLASINPUT+"="+acadena(p1,p1ops,false
                 )+";",true);
1302        } else if (SALIDA.equals("COCOA")) {
1303         stringToFile("I"+NUMREGLASINPUT+":="+acadena(p1,p1ops,false)+";
                 ",true);
1304        }
1305       }
1306       NUMREGLASINPUT++;
1307      }
1308      )* ";"
1309   }
1310 TOKEN:
1311   {
1312    < id : ["A"-"Z","a"-"z","a","_"](["A"-"Z","a"-"z","0"-"9","_","\""
          ])* >
1313   }
1314 TOKEN:
1315   {
1316    < numero : ["1"-"9"](["0"-"9"])* >
1317   }
1318 SKIP :
1319   {
1320    < "//" (~ ["\n", "\r"])* >
1321    { linea = linea + image; }
```

```
1322  }
1323 SKIP :
1324  {
1325   < "/*" (~ [ ])* "*/" >
1326   { linea = linea + image; }
1327  }
1328 SKIP :
1329  {
1330   < "(*" (~ ["\n", "\r"])* "*)" >
1331   { linea = linea + image; }
1332  }
1333 SKIP:
1334  {
1335   < "␣" | "\t" | "\r" >
1336   { linea = linea + image; }
1337  }
1338 SKIP:
1339  {
1340   "\n"
1341   {
1342    numLin++;
1343    //System.out.println(numLin + ": " + linea);
1344    linea = "";
1345   }
1346  }
```

F. APÉNDICE: Código de tabla de símbolos

F.1. Variable

```
1 /*
2  *  Attribution-Noncommercial 3.0 Unported - http://creativecommons.
       org/licenses/by-nc/3.0/
3  */
4 package tablasimboloscompilador;
5
6 import java.util.Vector;
7
8 /**
9  *
10 * @author Roberto Maestre Martinez
11 */
12 public class Variable {
13
14     public Vector<String> valores;
15     private String variable;
16     public int numvariables;
17     public int indice2;
18
19     public Variable(){
20         this.valores = new Vector<String>();
```

```
21      }
22
23      public String getNombreVariable(){
24          return this.variable;
25      }
26
27      Variable(String variable, int numvariables) {
28          this.valores = new Vector<String>();
29          this.variable=variable;
30          this.numvariables=numvariables;
31      }
32
33
34  }
```

F.2. Gestor de la tabla de símbolos

```
1 /*
2  *  Attribution-Noncommercial 3.0 Unported - http://creativecommons.
       org/licenses/by-nc/3.0/
3  */
4 package tablasimboloscompilador;
5
6 import excepciones.TSException_indicedevariablenoespecificado;
7 import excepciones.TSException_indicedevariablenorango;
8 import excepciones.TSException_indicevariablefueraderango;
9 import excepciones.TSException_nombrevariableduplicada;
10 import excepciones.TSException_valordevariableyaexiste;
11 import excepciones.TSException_valornoencontrado;
12 import excepciones.TSException_valoryvaribalecoinciden;
13 import excepciones.TSException_variablenoencontrada;
14 import java.util.Iterator;
15 import java.util.Set;
16 import java.util.TreeSet;
17 import java.util.Vector;
18
19 /**
20  *
21  * @author Roberto Maestre Martinez
22  */
```

```
23 public class TS {
24
25         //Estructuras globales para soportar la tabla de simbolos
26         private Vector<Variable> vars;
27         private Vector<String> palabrasReservadas;
28         //Estructuras auxiliares para marcar las variables que se
               utilizan
29         //Se utilizan TreeSet por su alta velocidad en las consultas
30         Set <String> marcas;
31
32
33         public TS(){
34             //Iniciamos el vector de punteros a variables
35             this.vars=new Vector<Variable>();
36             //Inicamos las estructuras de marcas
37             marcas = new TreeSet <String>();
38             //Iniciamos el vector de palabras reservadas del lenguaje
                   , es decir, no se podran definir
39             //variables ni valores con estas palabras
40             this.palabrasReservadas=new Vector<String>();
41             this.palabrasReservadas.add("BEGIN");
42             this.palabrasReservadas.add("BEGINVARS");
43             this.palabrasReservadas.add("ENDVARS");
44             this.palabrasReservadas.add("BEGINRULES");
45             this.palabrasReservadas.add("ENDRULES");
```

```
46              this.palabrasReservadas.add("BEGININPUT");
47              this.palabrasReservadas.add("ENDINPUT");
48              this.palabrasReservadas.add("OR");
49              this.palabrasReservadas.add("O");
50              this.palabrasReservadas.add("AND");
51              this.palabrasReservadas.add("A");
52              this.palabrasReservadas.add("Y");
53          }
54
55          public void marcaVariable(String variable){
56              this.marcas.add(variable);
57          }
58
59          public Vector<String> getWarningVariablesNoUsadas(){
60              Vector<String> aux=new Vector<String>();
61              for (int v=0;v<this.vars.size();v++){
62                  for(int i=1;i<=this.vars.get(v).numvariables;i++){
63                          //Si el indice de acceso es 1, se ha podido
                               llamar a la varaible por "var" en vez de
                               por "var[1]"
64                          String check = this.vars.get(v).
                               getNombreVariable()+"["+i+"]";
65                          if (!this.marcas.contains(check)){
66                              aux.add(check);
67                          }
```

```
68                  }
69              }
70              return aux;
71          }
72
73
74          public boolean esPrimo(int numero){
75              int contador = 2;
76              boolean primo=true;
77              while ((primo) && (contador!=numero)){
78                  if (numero % contador == 0)
79                      primo = false;
80                      contador++;
81                  }
82              return primo;
83          }
84
85
86          public int getNumVars(){
87              int aux=0;
88              for (int i=0;i<this.vars.size();i++){
89                  if (this.vars.get(i).numvariables>1)
90                      aux+=this.vars.get(i).numvariables-1;
91              }
92              return this.vars.size()+aux;
```

```
93          }
94
95          public int getNumSoluciones(){
96              return this.vars.get(0).valores.size();
97          }
98
99          public int getMayorPrimo(){
100             int mayor=Integer.MIN_VALUE;
101             for (int i=0;i<this.vars.size();i++) {
102                 if (this.vars.get(i).valores.size()>mayor)
103                     mayor=this.vars.get(i).valores.size();
104             }
105             mayor++;
106             while (!this.esPrimo(mayor)) mayor++;
107          return mayor;
108          }
109
110         public int getTotalVariablesRestriccionLogica(){
111             int aux=0;
112             for (int a=0;a<this.vars.size();a++) {
113                     aux+=(this.vars.get(a).numvariables*(this.vars.
                            get(a).valores.size()-1));
114             }
115             return aux;
116         }
```

```
117
118         public void comprubaaccesoavariable(String variable,String
                indice) throws TSException_variablenoencontrada,
                TSException_indicedevariablenorango,
                TSException_indicedevariablenoespecificado{
119             boolean enc=false;
120             int i=0;
121             int maxvars=this.vars.size();
122             while(i<maxvars && !enc){
123                 if (this.vars.get(i).getNombreVariable().equals(
                        variable)){
124                     enc=true;
125                     if (this.vars.get(i).numvariables>0 && indice.
                            equals("0")){
126                         throw new
                                TSException_indicedevariablenoespecificado
                                (variable);
127                     }
128                 }
129             i++;
130             }
131             if(!enc){
132                 throw new TSException_variablenoencontrada(variable);
133             }
134         }
```

```
135
136         public Vector<String> getValoresEnComun(String var1,String
                var2) throws TSException_variablenoencontrada{
137             Vector<String> valores = new Vector<String>();
138             int indicevar1=0,indicevar2=0;
139             boolean encvar1=false, encvar2=false;
140             int maxvars=this.vars.size();
141             int i=0;
142             while (i<maxvars && (!encvar1 || !encvar2)) {
143                 if (this.vars.get(i).getNombreVariable().equals(var1)
                        ){
144                     encvar1=true;
145                     indicevar1=i;
146                 }
147                 if (this.vars.get(i).getNombreVariable().equals(var2)
                        ){
148                     encvar2=true;
149                     indicevar2=i;
150                 }
151             i++;
152             }
153             //Si no encuentra alguna de las variables, lanzamos la
                    excepcion correspondiente
154             if (!encvar1)
155                 throw new TSException_variablenoencontrada(var1);
```

```
156             if (!encvar2)
157                 throw new TSException_variablenoencontrada(var2);
158             //Si ha encontrado las variables, chequeamos los valores
                    que tienen en comun y devolvemos
159             //la lista con dichos valores comunes a ambas variables
160             for (int a=0;a<this.vars.get(indicevar1).valores.size();a
                    ++){
161                 for (int b=0;b<this.vars.get(indicevar2).valores.size
                        ();b++){
162                     //System.out.println(this.vars.get(indicevar1).
                            valores.get(a)+":::::"+this.vars.get(
                            indicevar2).valores.get(b));
163                     if (this.vars.get(indicevar1).valores.get(a).
                            equals(this.vars.get(indicevar2).valores.get(b
                            ))){
164                         valores.add(this.vars.get(indicevar1).valores
                                .get(a));
165                     }
166                 }
167             }
168             return valores;
169         }
170
171         public void generarIndices(){
172             //Genero el indice2 para el modelo polinomico
```

```
173             int aux=0;
174             for (int i=0;i<this.vars.size();i++){
175                 this.vars.get(i).indice2 = aux;
176                 int numerovariables=this.vars.get(i).numvariables;
177                 if (numerovariables==0) numerovariables=1;
178                 aux +=(this.vars.get(i).valores.size())*
                        numerovariables;
179             }
180         }
181
182
183         //Para el modelo algebraico
184         public int[] getIdentificadorRealAlgebraico(String variable,
                int indicearray,String valor) throws
                TSException_variablenoencontrada,
                TSException_valornoencontrado,
                TSException_indicedevariablenorango{
185             //Inicio la respuesta
186             int[] aux=null;
187             int i=0;
188             int maxvars=this.vars.size();
189             boolean enc=false;
190             while(i<maxvars && !enc){
191                 if (this.vars.get(i).getNombreVariable().equals(
                        variable)){
```

```
192                 //Una vez encontrada la variable compruebo que el
                        indice de acceso
193                 //no sobrepase a lo declarado
194                 if (indicearray>this.vars.get(i).numvariables){
195                     throw new TSException_indicedevariablenorango
                            (indicearray,this.vars.get(i).numvariables
                            );
196                 }
197                 enc=true;
198                 boolean encvalor=false;
199                 int c=0;
200                 int maxvalores=this.vars.get(i).valores.size();
201                 while (c<maxvalores && !encvalor) {
202                     if (this.vars.get(i).valores.get(c).equals(
                            valor)){
203                         encvalor=true;
204                         aux = new int[3];
205                         //Devuelvo el identificador real de la
                                variable, es decir, variable
206                         // teniendo en cuenta el valor del array
                                si esta declarado y la posicion de la
                                variable
207                         int numvarsconmasdedosvalores=0;
208                         for (int x=0;x<i;x++){
```

```
209                                 if ((this.vars.get(x).numvariables)
                                        >1)
210                                     numvarsconmasdedosvalores+=(this.
                                            vars.get(x).numvariables-1);
211                                 }
212                                 //Ahora sumar los indices que se dejan
                                        por detras en la misma variable
213                                 aux[0]=(i+indicearray+
                                        numvarsconmasdedosvalores)-1;
214                                 aux[1]=c;
215                             }
216                         c++;
217                         }
218                         if (!encvalor){
219                             throw new TSException_valornoencontrado(valor
                                    );
220                         }
221                     }
222                 i++;
223                 }
224                 if (!enc){
225                     throw new TSException_variablenoencontrada(variable);
226                 }
227                 return aux;
228             }
```

```
229
230
231
232
233         //Para el modelo algebraico
234         public int[] getIdentificadorRealAlgebraicoSinValor(String
              variable,int indicearray) throws
              TSException_variablenoencontrada,
              TSException_valornoencontrado,
              TSException_indicedevariablenorango{
235           //Inicio la respuesta
236           int[] aux=null;
237           int i=0;
238           int maxvars=this.vars.size();
239           boolean enc=false;
240           while(i<maxvars && !enc){
241               if (this.vars.get(i).getNombreVariable().equals(
                      variable)){
242                   //Una vez encontrada la variable compruebo que el
                          indice de acceso
243                   //no sobrepase a lo declarado
244                   if (indicearray>this.vars.get(i).numvariables){
245                       throw new TSException_indicedevariablenorango
                              (indicearray,this.vars.get(i).numvariables
                              );
```

```
246                     }
247                     enc=true;
248                     int c=0;
249                     //Ahora sumar los indices que se dejan por detras
                            en la misma variable
250                      aux = new int[2];
251                      int numvarsconmasdedosvalores=0;
252                              for (int x=0;x<i;x++){
253                                  if ((this.vars.get(x).numvariables)
                                        >1)
254                                      numvarsconmasdedosvalores+=(this.
                                            vars.get(x).numvariables-1);
255                              }
256                     aux[0]=(i+indicearray+numvarsconmasdedosvalores)
                            -1;
257                     aux[1]=c;
258                 }
259             i++;
260             }
261             if (!enc){
262                 throw new TSException_variablenoencontrada(variable);
263             }
264             return aux;
265         }
266
```

```
267
268
269
270
271
272
273         //Para el modelo proposicional
274         public int[] getIdentificadorReal(String variable,int
                indicearray,String valor) throws
                TSException_variablenoencontrada,
                TSException_valornoencontrado,
                TSException_indicedevariablenorango{
275             //Inicio la respuesta
276             int[] aux=null;
277             int i=0;
278             int maxvars=this.vars.size();
279             boolean enc=false;
280             while(i<maxvars && !enc){
281                 if (this.vars.get(i).getNombreVariable().equals(
                        variable)){
282                     //Una vez encontrada la variable compruebo que el
                            indice de acceso
283                     //no sobrepase a lo declarado
284                     if (indicearray>this.vars.get(i).numvariables){
```

```
285                 throw new TSException_indicedevariablenorango
                        (indicearray, this.vars.get(i).numvariables
                        );
286             }
287             enc=true;
288             boolean encvalor=false;
289             int c=0;
290             int maxvalores=this.vars.get(i).valores.size();
291             while (c<maxvalores && !encvalor) {
292                 if (this.vars.get(i).valores.get(c).equals(
                        valor)){
293                     encvalor=true;
294                     aux = new int[3];
295                     //Devuelvo el identificador real de la
                            variable, es decir, variable
296                     // teniendo en cuenta el valor del array
                            si esta declarado y la posicion de la
                            variable
297
298                     //Ademas para la optimizacion tengo que
                            devolver el id - el numero de
                            variables que tenga
299                     //por detras que tengan mas de 1 valor
                            por variable. Ademas hay que
                            contabilizar dentro de la
```

```
300                     //variable que se este accediendo el
                            indice que tiene y sumar uno por cada
                            indice que deje por detras
301                     //de su misma variable
302                     int numvarsconmasdedosvalores=0;
303                     for (int x=0;x<i;x++){
304                         if ((this.vars.get(x).valores.size())
                                >1)
305                             numvarsconmasdedosvalores+=(this.
                                    vars.get(x).numvariables);
306                     }
307                     //Ahora sumar los indices que se dejan
                            por detras en la misma variable
308                     numvarsconmasdedosvalores+=indicearray-1;
309                     aux[0]=(this.vars.get(i).indice2+((
                            indicearray-1)*this.vars.get(i).
                            valores.size())+c)-
                            numvarsconmasdedosvalores;
310                     aux[1]=c;
311                     // Si es el ultimo valor marco un flag
                            para la optimizacion
312                     if (c==maxvalores-1) aux[2]=1;
313                     else aux[2]=0;
314                 }
315             c++;
```

```
316                     }
317                     if (!encvalor){
318                         throw new TSException_valornoencontrado(valor
                                );
319                     }
320                 }
321             i++;
322             }
323             if (!enc){
324                 throw new TSException_variablenoencontrada(variable);
325             }
326             return aux;
327         }
328
329
330         public Vector<String> getIdentificadoresRealOptimizado(String
                variable,int indice) throws
               TSException_variablenoencontrada,
               TSException_valornoencontrado,
               TSException_indicedevariablenorango{
331             Vector<String> identificadores=null;
332             boolean enc=false;
333             int i=0;
334             int maxvars=this.vars.size();
335             while(i<maxvars && !enc){
```

```
336                 if (this.vars.get(i).getNombreVariable().equals(
                        variable)){
337                     enc=true;
338                     identificadores=new Vector<String>();
339                     for (int c=0;c<this.vars.get(i).valores.size()-1;
                            c++){
340                         int[] identificadorReal=this.
                                getIdentificadorReal(this.vars.get(i).
                                getNombreVariable(),indice, this.vars.get(
                                i).valores.get(c));
341                         identificadores.add(identificadorReal[0]+"");
342                     }
343                 }
344             i++;
345             }
346             if (!enc) {
347             //Si no ha encontrado la variable, lanzamos excepcion
348                 throw new TSException_variablenoencontrada(variable);
349             }
350             return identificadores;
351             }
352
353         public boolean existeVariable(String variable){
354             boolean enc=false;
355             int i=0;
```

```
356             int maxvars=this.vars.size();
357             while(i<maxvars && !enc){
358                 if (this.vars.get(i).getNombreVariable().equals(
                        variable))
359                     enc=true;
360             i++;
361             }
362             return enc;
363         }
364
365
366         public void insertaVariable(String variable,int numvariables)
                throws TSException_nombrevariableduplicada,
               TSException_indicedevariablenorango,
               TSException_valoryvaribalecoinciden,
               TSException_indicevariablefueraderango{
367             //Compruebo que no sea una palabra reservada del lenguaje
368             for (int i=0;i<this.palabrasReservadas.size();i++){
369                 if (variable.equals(this.palabrasReservadas.get(i))){
370                     throw new TSException_nombrevariableduplicada(
                            variable);
371                 }
372             }
373             //Compruebo que no se repita la variable
374             for (int i=0;i<this.vars.size();i++){
```

```
375                 if (this.vars.get(i).getNombreVariable().equals(
                        variable)){
376                     throw new TSException_nombrevariableduplicada(
                            variable);
377                 }
378             }
379             //Compruebo que el indice del array sea correcto
380             if (numvariables<1 || numvariables>10000) {
381                 throw new TSException_indicevariablefueraderango(
                        numvariables);
382             }
383             //Compruebo que la variable no exista en ninguno de los
                    valores
384             boolean enc=false;
385             int i=0;
386             int maxvars=this.vars.size();
387             while(i<maxvars && !enc){
388                 int c=0;
389                 int maxvalores=this.vars.get(i).valores.size();
390                 while(c<maxvalores && !enc){
391                     if (this.vars.get(i).valores.get(c).equals(
                            variable)) {
392                         enc=true;
393                     }
394                 c++;
```

```
395                 }
396             i++;
397             }
398             if (enc){
399                 throw new TSException_valoryvaribalecoinciden(
                        variable);
400             }
401             this.vars.add(new Variable(variable,numvariables));
402         }
403
404
405
406
407         public void insertaValorEnVariable(String variable,String
                valor) throws TSException_variablenoencontrada,
                TSException_valordevariableyaexiste,
                TSException_valoryvaribalecoinciden {
408             //Busco la variable
409             int max=this.vars.size();
410             boolean enc=false;
411             int i=-1;
412             int x=0;
413             //Primero compruebo que no coincida el ninguna variable
                    con el valor
```

```
414             //depaso miro si la variable existe, si es asi guardo la
                    posicion en la
415             //que se encurntra
416             while (x<max) {
417                     if (this.vars.get(x).getNombreVariable().equals(
                            valor)){
418                         throw new TSException_valoryvaribalecoinciden
                                (valor);
419                     }
420                     if (this.vars.get(x).getNombreVariable().equals(
                            variable)) {
421                         enc=true;
422                         i=x;
423                     }
424              x++;
425              }
426             //Si la variable existe ya proseguimos
427             if(enc) {
428                      boolean valorenc=false;
429                      int maxvalor=this.vars.get(i).valores.size();
430                      int c=0;
431                      while (c<maxvalor && !valorenc){
432                          if (this.vars.get(i).valores.get(c).equals(
                                valor)){
433                              valorenc=true;
```

```
434                         }
435                     c++;
436                     }
437                     //Si se ha encontrado el valor, lanzo excepcion
                            ya que no pueden estar repetidos
438                     if (valorenc){
439                         throw new TSException_valordevariableyaexiste
                                (variable,valor);
440                     //Si no lo ha encontrado, inserto el nuevo valor
441                     } else {
442                         this.vars.get(i).valores.add(valor);
443                     }
444             } else {
445             //Si no ha encontrado la variable, lanzamos excepcion
446                 throw new TSException_variablenoencontrada(variable);
447             }
448         }
449
450
451         public Vector<String> getRestriccionesVariables(String si,
                String sd) throws TSException_variablenoencontrada,
                TSException_valornoencontrado,
                TSException_indicedevariablenorango{
452             Vector<String> res=new Vector<String>();
453             for (int a=0;a<this.vars.size();a++) {
```

```
454                 for (int b=0;b<this.vars.get(a).numvariables;b++) {
455                     for (int i=0;i<this.vars.get(a).valores.size()-1;
                            i++) {
456                         String aux1="",aux2="";
457                             int[] var1=this.getIdentificadorReal(this
                                    .vars.get(a).getNombreVariable(),b+1,
                                    this.vars.get(a).valores.get(i));
458                             aux1="x"+si+var1[0]+sd;
459                             res.add(aux1);
460                     }
461                 }
462             }
463             return res;
464         }
465
466         public Vector<String> getRestriccionesProposicional(String si
                ,String sd) throws TSException_variablenoencontrada,
                TSException_valornoencontrado,
                TSException_indicedevariablenorango{
467             Vector<String> pares=new Vector<String>();
468             for (int a=0;a<this.vars.size();a++) {
469                 for (int b=0;b<this.vars.get(a).numvariables;b++) {
470                     for (int i=0;i<this.vars.get(a).valores.size()-1;
                            i++) {
```

```
471                         for (int c=i+1;c<this.vars.get(a).valores.
                                size()-1;c++) {
472                             String aux1="",aux2="";
473                             //NoCompruebo si es el ultimo valor para
                                    devolver optimizado, ya que la primera
                                     variable
474                             //del binomio nunca puede ser el ultimo
                                    valor de la variable
475                             int[] var1=this.getIdentificadorReal(this
                                    .vars.get(a).getNombreVariable(),b+1,
                                    this.vars.get(a).valores.get(i));
476                             aux1="x"+si+var1[0]+sd;
477
478                             int[] var2=this.getIdentificadorReal(this
                                    .vars.get(a).getNombreVariable(),b+1,
                                    this.vars.get(a).valores.get(c));
479                             aux2="x"+si+var2[0]+sd;
480                             pares.add(aux1);
481                             pares.add(aux2);
482                         }
483                     }
484                 }
485             }
486             return pares;
487         }
```

```
488
489
490
491         public int getNumValoresDeVariable(int indice3) throws
                TSException_variablenoencontrada{
492             boolean enc=false;
493             int i=0,centvalor=0,aux=0;
494             while (!enc && i < this.vars.size()){
495                 int c=0;
496                 while (!enc && c<this.vars.get(i).numvariables){
497                     if (indice3==aux){
498                         enc=true;
499                         centvalor=this.vars.get(i).valores.size();
500                     }
501                     aux++;
502                 c++;
503                 }
504                 i++;
505             }
506             if (!enc)
507                 throw new TSException_variablenoencontrada("");
508             return centvalor;
509         }
510
511
```

```
512         public Vector<String> getRestriccionesAlgebraicas(String si,
                String sd) throws TSException_variablenoencontrada,
                TSException_valornoencontrado,
                TSException_indicedevariablenorango{
513             Vector<String> pares=new Vector<String>();
514             for (int a=0;a<this.vars.size();a++) {
515                 for (int b=0;b<this.vars.get(a).numvariables;b++) {
516
517                 }
518             }
519             return pares;
520         }
521
522
523         public String imprimirTS() throws
                TSException_variablenoencontrada,
                TSException_valornoencontrado,
                TSException_indicedevariablenorango{
524             String res="\n Tabla de simbolos\n
                    ==========\n";
525             int tot=0;
526             int aux=0;
527              for (int i=0;i<this.vars.size();i++){
528                 for (int nv=0;nv<this.vars.get(i).numvariables;nv++)
                        {
```

```
529                 for (int c=0;c<this.vars.get(i).valores.size();c
                        ++){
530                     //Calculo del indice real proposicional
531                     String strreal="";
532                     if (c<this.vars.get(i).valores.size()-1){
533                         strreal="{"+(tot-aux)+"}";
534                     } else aux++;
535                     //Calculo el indice real algebraico
536                     String indice3="";
537                     int auxants=0;
538                     for (int x=0;x<i;x++){
539                         if (this.vars.get(x).numvariables>1)
540                             auxants+= this.vars.get(x).
                                    numvariables-1;
541                     }
542                     if (c==0){
543                         indice3="["+(i+nv+auxants)+"]";
544                     }
545                     res+="{"+tot+"}\t"+strreal+"\t"+indice3+"\t"+
                        this.vars.get(i).getNombreVariable()+"["+(
                        nv+1)+"]="+this.vars.get(i).valores.get(c)
                        +"\n";
546                     tot++;
547                 }
548             }
```

```
549                     res+="————————————————————————————————————————————\
                            n";
550                 }
551                 return res;
552             }
553
554 }
```

G. APÉNDICE: Modificación, compilación y empaquetado del traductor

G.1. Compilación

REQUISITO: Tener en el classpath las rutas donde se encuentra "java.exe" y "javac.exe"

Dependiendo de la versión de Java, normalmente se encuentran en:

- java.exe:

```
C:\Program Files\Java\jre1.5.0_12\bin
```

- javac.exe:

```
C:\Program Files\Java\jdk1.6.0_16\bin
```

Los pasos para realizar las modificaciones y compilar de nuevo el traductor son:

1. Realizar las modificaciones necesarias en el fichero 'traductor.jj', este es el fichero que tiene la especificación del lenguaje en JavaCC
2. Una vez realizados los cambios, compilar el fichero'traductor.jj':

```
R:\TFC-DATOS\javacc-4.0\bin > javacc traductor.jj
```

   - Compilar siempre en la carpeta donde se encuentre 'javacc.bat'
   - Esta compilación generará las clases en Java necesarias en el directorio destino indicado en el archivo 'traductor.jj' en la primera línea

```
Output_directory="R:\\TFC-DATOS\\javacc-4.0\\traductor";
```

3. Ir a la carpeta de destino Output_directory :

```
cd R:\\TFC-DATOS\\javacc-4.0\\traductor"
```

4. Ahora solo tenemos que compilar la clase java principal del traductor: 'Traductor.java'

```
R:\TFC-DATOS\javacc-4.0\traductor > javac Traductor.java
```

G.2. Empaquetado

REQUISITO: Tener en el classpath las rutas donde se encuentra "jar.exe"

Dependiendo de la versión de Java, normalmente se encuentran en:

- java.exe:

```
C:\Program Files\Java\jdk1.6.0_16\bin
```

1. Crear el archivo jar con las clases ya compiladas del traductor:

```
R:\TFC-DATOS\javacc-4.0\traductor >
jar -cf traductor.jar *.class
```

2. Abrir "traductor.jar" con un programa de compresión, por ejemplo WinRAR y poner el texto del siguiente cuadro en "MANIFEST.M":

```
Manifest-Version: 1.0
Main-Class: Traductor
```

3. Por último, poner las carpetas: 'excepciones' y 'tablasimboloscompilador' en el raíz del fichero traductor.jar

Ya está empaquetado el programa y listo para ejecutar:

```
R:\TFC-DATOS\javacc-4.0\traductor >
java -jar Traductor.jar 1 1 < fuente.txt
```

Referencias

[1] E. Roanes-Lozano, A. Hernando, L. Laita, and E. Roanes-Macías, "A groebner bases-based approach to backward reasoning in rule based expert systems," *Annals of Mathematics and Artificial Intelligence*, vol. 56, pp. 297–311, 2009. 10.1007/s10472-009-9147-4.

[2] J. Alonso and E. Briales, "Lógicas polivalentes y bases de gröbner.," *In: C. Martin, ed.: Actas del V Congreso de Lenguajes Naturales y Lenguajes Formales. University of Seville, Seville*, pp. 307–315, 1995.

[3] J. Chazarain, A. Riscos, J. A. Alonso, and E. Briales, "Multi-valued logic and gröbner bases with applications to modal logic," *J. Symb. Comput.*, vol. 11, pp. 181–194, April 1991.

[4] J. Hsiang, "Refutational theorem proving using term-rewriting systems," *Artificial Intelligence*, vol. 25, no. 3, pp. 255 – 300, 1985.

[5] D. Kapur and P. Narendran, "An equational approach to theorem proving in first-order predicate calculus," in *Proceedings of the 9th international joint conference on Artificial intelligence - Volume 2*, (San Francisco, CA, USA), pp. 1146–1153, Morgan Kaufmann Publishers Inc., 1985.

[6] E. Roanes-Lozano, L. M. Laita, and E. Roanes-Macías, "A polynomial model for multivalued logics with a touch of algebraic geometry and computer algebra," *Mathematics and Computers in Simulation*, vol. 45, pp. 83–99, 1998.

[7] A. Capaini and G. Niesi, "Cocoa user's manual. dept. mathematics, univ. of genova, 1996." Available at `http://cocoa.dima.unige.it`.

[8] D. Perkinson, "Cocoa 4.0 online help (electronic file acompanying cocoa v.4.0)."

[9] W. Decker, G.-M. Greuel, G. Pfister, and H. Schönemann, "SINGULAR 3-1-1 — A computer algebra system for polynomial computations. 2010." Available at `http://www.singular.uni-kl.de`.

[10] M. Brickenstein and A. Dreyer, "Polybori: A framework for gr'obner-basis computations with boolean polynomials," *Journal of Symbolic Computatione*, vol. 44, no. 9, pp. 1326–1345, 2009.

[11] L. Laita, E. Roanes-Lozano, V. Maojo, L. de Ledesma, and L. Laita, "An expert system for managing medical appropriateness criteria based on computer algebra techniques," *Computers and Mathematics with Applications*, vol. 51, no. 5, pp. 473–481, 2000.

[12] M. Lourdes Jimenez, J. M. Santamaría, R. Barchino, L. Laita, L. M. Laita, L. A. González, and A. Asenjo, "Knowledge representation for diagnosis of care problems through an expert system: Model of the auto-care deficit situations," *Expert System with Applications*, vol. 34, pp. 2847–2857, 2008.

[13] C. Pérez-Carretero, L. M. Laita, E. Roanes-Lozano, L. Lázaro, J. González-Cajal, and L. Laita, "A logic and computer algebra-based expert system for diagnosis of anorexia," *Mathematics and Computers in Simulation*, vol. 58, pp. 183–202, February 2002.

[14] C. Rodríguez-Solano, L. M. Laita, E. Roanes-Lozano, L. López-Corral, and L. Laita, "A computational system for diagnosis of depressive situations," *Expert Systems with Applications*, vol. 31, no. 1, pp. 47 – 55, 2006.

[15] M. Minsk, "A framework for representing knowledge," *MIT-AI Laboratory Memo*, vol. 306, 1974.

[16] A. Hernando, "A new algebraic model for implementing expert systems represented under the 'concept-attribute-value' paradigm," *Mathematics and Computers in Simulation*, vol. In Press, Corrected Proof, pp.–, 2010.

[17] B. Buchberger, "Bruno buchberger's phd thesis 1965: An algorithm for finding the basis elements of the residue class ring of a zero dimensional polynomial ideal," *J. Symb. Comput.*, vol. 41, pp. 475–511, March 2006.

[18] D. A. Cox, J. Little, and D. O'Shea, *Ideals, Varieties, and Algorithms. An Introduction to Computational Algebraic Geometry and Commutative Algebra*. Undergraduate Texts in Mathematics, Springer, 2007.

[19] "National comprenhensive cancer network. nccn clinical practice guidelines in oncology v.2.2010.." Available at `http://www.nccn.org`.

[20] ISO/IEC, "14977:1996 information technology – syntactic metalanguage – extended bnf." Available at `http://www.iso.org/iso/catalogue_detail.htm?csnumber=26153`.

[21] S. Microsystems, "Javacc: a parser/scanner generator for java." Available at `https://javacc.dev.java.net/` .

[22] B. Buchberger, "Gröbner bases: A short introduction for systems theorists," in *Computer Aided Systems Theory — EUROCAST 2001* (R. Moreno-Díaz, B. Buchberger, and J. Luis Freire, eds.), vol. 2178 of *Lecture Notes in Computer Science*, pp. 1–19, Springer Berlin / Heidelberg, 2001. 10.1007/3-540-45654-6_1.