Especificación formal de bases de datos con un demostrador automático de teoremas

Juan J. Arrabal, José A. Alonso, Delia Balbontín y Javier Herrera
Facultad de Informática y Estadistica
Universidad de Sevilla
arrabal@sevaxu.cica.es

1 Introducción

El uso de tecnología de programación declarativa en el desarrollo de aplicaciones de bases de datos [5, 8] permite formalizar el concepto de base de datos en un lenguaje único para expresar de forma uniforme datos, programas, vistas, consultas y restricciones de integridad, lo que facilita el estudio de los procesos de consulta y la satisfacción de las restricciones de integridad. El uso de la unificación posibilita emplear estructuras de datos parcialmente definidas, invertibilidad de las relaciones y computación deductiva, de modo que es posible encontrar la sustitución que hace cumplir un objetivo bajo la teoría especificada.

Un problema fundamental en el inicio de una aplicación de bases de datos es la realización del modelo conceptual, que describe las informaciones, restricciones y requerimientos del sistema [2]. El modelo conceptual sirve de punto de partida para la definición del sistema y es esencialmente declarativo e independiente de la implantación efectiva del sistema.

Varios investigadores han dedicado atención al problema de especificar formalmente el esquema conceptual, aplicando ideas del campo de especificación algebraica junto con técnicas clásicas de bases de datos [3, 4, 6], programación lógica [7, 11], lógico-ecuacional [10, 11] y lógico-ecuacional con restricciones [1]. Así ha quedado asentado que un formalismo a emplear en la tarea de especificaciones formales debe ofrecer al menos las siguientes facilidades

1. Funciones, igualdad y otras relaciones que permitan especificar las operaciones de actualización y consulta.
2. Mecanismos de resolución y simplificación de restricciones.
3. Un tratamiento de la negación que permita expresar y deducir la falsedad de la igualdad y otras relaciones.

La demostración automática de teoremas [13, 14] proporciona, junto a esas facilidades, lenguaje clausal no restringido a cláusulas Horn, negación de predicados y relaciones, resolución generalizada, tratamiento eficiente de la igualdad, posibilidades de reescritura de términos, retención de información y diversidad de estrategias que permiten dirigir el proceso de deducción.

En el presente trabajo se plantea el problema de la especificación del modelo conceptual como teoría lógica de primer orden con igualdad y negación, de forma tal que la propia especificación sea adecuada como entrada de un programa de deducción automática. Como tal, hemos elegido el programa OTTER [9] que, junto a las facilidades antes enumeradas, reune la capacidad de ser utilizado al modo de un lenguaje de programación funcional [12].

Centraremos la atención en el ejemplo de la *agencia de empleo* introducido en [10] para ilustrar diferentes técnicas de especificación formal de una aplicación de bases de datos. En [1] se trata este mismo ejemplo para mostrar que la programación lógica con restricciones es capaz de proporcionar especificaciones ejecutables. En nuestro caso, nos mostrará que la demostración automática de teoremas proporciona, a través del proceso de deducción, además de la animación de especificaciones, la capacidad de explicar qué enunciados de la especificación intervienen en el procesado de cada consulta o transación del sistema.

2 Demostración automática de teoremas con OTTER

El lenguaje clausal, familiar en el campo de la programación lógica, es el modo básico de representación para el programa OTTER de demostración automática. En esta sección exponemos algunas notaciones y convenciones en el lenguaje clausal que difieren de lo habitual en programación lógica y también, las reglas de razonamiento que emplearemos en las secciones siguientes.

Las variables son elegidas entre los términos que comienzan con una letra minúscula comprendida desde u hasta z. Cada variable está implicitamente en el alcance de un cuantificador universal, y cada variable es relevante sólo para la cláusula en que ocurre. Por convención, las constantes y funciones se escribes en minúsculas y los predicados comenzando en mayúscula. El predicado de igualdad = está predefinido en el programa. La negación se representa por - y la disyunción por | . La existencia se representa por constantes y funciones. A diferencia con la programación lógica, el operador lógico **y** nunca aparece en una cláusula. No hay límite al número de literales positivos que aparecen en una cláusula, lo que permite gran flexibilidad expresiva al lenguaje. El siguiente ejemplo ilustra este estilo de escritura de cláusulas.

abuelo(X,Y):-progenitor(X,Z),progenitor(Z,Y).

Abuelo(x,y)|-Progenitor(x,z)|-Progenitor(z,y).

(x=0)|(x=1)|Mayor_que(x,1).

La primera cláusula está en la notación PROLOG y las restantes en notación clausal. La última es un ejemplo de uso de la igualdad.

La negación de un teorema puede ser representada con cláusulas cada una de las cuales contiene sólo literales positivos. En OTTER esa cláusula pude ser un objetivo, a diferencia de lo que ocurre en programación lógica.

La regla de inferencia *hiperresolución* considera una cláusula no positiva (el *núcleo*), intenta unificar simultáneamente cada uno de sus literales negativos con un literal en una cláusula positiva (un *satélite*) y, si tiene éxito, produce una cláusula positiva como conclusión (posiblemente la cláusula vacía). La regla de inferencia UR–*resolución* considera una cláusula no unitaria (el *núcleo*), intenta unificar cada uno de sus literales, salvo uno, con una cláusula unitaria (un *satélite*) y, si tiene éxito, produce como conclusión una cláusula unitaria.

La canonización en la escritura de términos es posible gracias al mecanismo de *demodulación* . Uno puede, por ejemplo, querer reemplazar uniformemente $-(-(t))$ por t para cualquier término t, o reemplazar $r(s+t)$ por $rs+rt$ para cualesquiera términos r, s y t. Tal reemplazamiento de términos lo realiza la paramodulación según algún conjunto de reglas de reescritura llamadas *demoduladores*. Las demoduladores pueden ser incluidos entre las cláusulas de entrada al programa o instruir a éste para que encuentre los demoduladores potencialmente útiles. Para los dos ejemplos anteriores, se pueden usar los siguientes demoduladores

$$(-(-(x)) = x)$$

y

$$(r(s+t) = rs+rt)$$

respectivamente.

Cada cláusula obtenida durante el proceso de deducción es retenida si no hay otra cláusula mas general que contiene toda su información, entonces ésta *subsume* a la primera, que es descartada.

Las cláusulas objetivo, aquellas sobre las cuales se centra la deducción, forman el *conjunto soporte* inicial, al que se le van añadiendo las cláusulas deducidas. Sólo se deducirán cláusulas que tengan un antecedente en una cláusula del conjunto soporte.

OTTER reconoce algunas funciones especiales y símbolos de predicado ($SUM, $LT, $EQ, ...) como símbolos evaluables. Puede emplearse aritmética entera, comparación léxica, evaluación booleana y expresiones condicionales que serán evaluadas durante demodulación y durante hiperresolución. Si, por ejemplo, demodulación encuentra un término $\$SUM(i_1, i_2)$, donde i_1 y i_2 son enteros, entonces el término es reescrito en i_3, la suma de i_1 e i_2, como si estuviera presente el demodulador ($\$SUM(i_1, i_2) = i_3$). Si, por ejemplo, hiperresolución encuentra el literal negativo -$\$LT(t_1, t_2)$, entonces t_1 y t_2 son demodulados; si los resultados son, respectivamente, enteros i_1 e i_2, con $i_1 < i_2$, entonces el literal es descartado como si estuviera presente la cláusula unitaria $\$LT(t_1, t_2)$.

Cualquier literal cuyo símbolo de predicado comience con $ans, $Ans o $ANS es un literal *answer*. El principal uso de esos literales es grabar, durante una búsqueda por refutación, las instanciaciones de las variables en las cláusulas de entrada. Las reglas de inferencia insertan en las cláusulas hijo las instancias apropiadas de cualquier literal *answer* presente en las cláusulas padre.

3 Esquema conceptual

El ejemplo de la *agencia de empleo* que formalizaremos a continuación, se especifica informalmente de la siguiente manera [1]:

La base de datos se inicializa a un *estado* vacío (denotado por *nil*). En un estado dado, las *personas* en paro pueden *solicitar* empleo y las *empresas* pueden $ofrecer$ un número de plazas, *contratar* personas en paro y despedir *empleados*. En el sistema se imponen las siguientes restricciones de integridad:

- Una persona puede solicitar empleo una sóla vez, adquiriendo la condición de *parado*. Esta condición se pierde si la persona es contratada por una empresa, en cuyo caso pasa a ser *empleado*. Si es despedido, recupera la condición de parado.
- Una empresa puede ofrecer plazas varias veces. El número de plazas ofrecidas se acumula para definir el número de plazas libres de cada emplesa. Este número disminuye con cada contratación y aumenta con los despidos.
- Sólo es posible contratar a personas con la condición de parado y sólo por empresas que tengan plazas libres.

Dentro del lenguaje de cláusulas con igualdad incorporada y predicados evaluables en tiempo de resolución de OTTER, se puede dar la siguiente especificación formal.

```
list(usable).

% Definicion de Estado(u) (de la base de datos)
```

```
Estado(nil).
- Estado(u) | - Persona(x) | -Ingresa(x,u) | Estado(ingresa(x,u)).
- Estado(u) | - Empresa(v) | - Oferta(v,y,u) | Estado(oferta(v,y,u)).
- Estado(u) | - Persona(x) | - Empresa(v) | - Despide(x,v,u)
            | Estado(despide(x,v,u)).
- Estado(u) | - Persona(x) | - Empresa(v) |  -Contrata(x,v,u)
            | Estado(contrata(x,v,u)).

% Clausulas  para Ingresa(Persona,Estado) (en la lista del desempleo)

-Persona(x) | -Estado(u)| -Empresa(v)| -Contrata(x,v,u)
            | -Ingresa(x,contrata(x,v,u)).
-Persona(x) | -Estado(u)| -Empresa(v)| -Despide(x,v,u)
            |  Ingresa(x,despide(x,v,u)).

% EsParado(Persona,Estado)

- Persona(x) | - Estado(u) | - Ingresa(x,u)
             | EsParado(x,ingresa(x,u)).
- Persona(x) | -Persona(z) | (x = z) | - Estado(u) | - Ingresa(z,u)
             | - EsParado(x,u) | EsParado(x,ingresa(z,u)).
-Persona(x) | - Estado(u) | - Empresa(v) | - EsParado(x,u)
            | - Oferta(v,y,u) | EsParado(x,oferta(v,y,u)).
-Persona(x) | - Persona(z) | (x = z) | - Estado(u) | -Empresa(v)
            | - EsParado(x,u) | - Despide(z,v,u)
            | EsParado(x,despide(z,v,u)).
-Persona(x) | - Persona(z) | (x = z) | - Estado(u) | -Empresa(v)
            | - EsParado(x,u) | - Contrata(z,v,u)
            | EsParado(x,contrata(z,v,u)).
-Persona(x) | - Estado(u) | -Empresa(v)
            | - EsParado(x,u) | - Contrata(x,v,u)
            | - EsParado(x,contrata(x,v,u)).

% EsEmpleado(Persona,Estado)

- Persona(x) | -Estado(u) | - Empresa(v) | -Contrata(x,v,u)
             | EsEmpleado(x,contrata(x,v,u)).
-Persona(x) | - Persona(z) | (x = z) | - Estado(u) | -Empresa(v)
            | - EsEmpleado(x,u) | -Contrata(z,v,u)
            | EsEmpleado(x,contrata(z,v,u)).
-Persona(x) | - Persona(z) | (x = z) | - Estado(u) | -Empresa(v)
            | - EsEmpleado(x,u) | -Despide(z,v,u)
            | EsEmpleado(x,despide(z,v,u)).
-Persona(x) | -Persona(z) | (x = z) | - Estado(u) | - EsEmpleado(x,u)
            | - Ingresa(z,u) | EsEmpleado(x,ingresa(z,u)).
-Persona(x) | - Estado(u) | - Empresa(v) | - EsEmpleado(x,u)
            | - Oferta(v,y,u) | EsEmpleado(x,oferta(v,y,u)).

% Trabajapara(Persona,Empresa,Estado)

-Persona(x) | - Persona(z) | (x = z) | - Estado(u) | -Empresa(v)
```

```
            | - Ingresa(z,u) | - Trabajapara(x,v,u)
            | Trabajapara(x,v,ingresa(z,u)).
-Persona(x) | - Estado(u) | -Empresa(v1) | -Empresa(v2)
            | - Oferta(v1,y,u) | - Trabajapara(x,v2,u)
            | Trabajapara(x,v2,oferta(v1,y,u)).
-Persona(x) | - Persona(z) | (x = z) | - Estado(u)
            | -Empresa(v1) | -Empresa(v2)
            | - Contrata(z,v1,u) | - Trabajapara(x,v2,u)
            | Trabajapara(x,v2,contrata(z,v1,u)).
-Persona(x) | - Estado(u) | -Empresa(v)
    | - Contrata(x,v,u) | Trabajapara(x,v,contrata(x,v,u)).
-Persona(x) | - Persona(z) | (x = z) | - Estado(u)
            | -Empresa(v1) | -Empresa(v2)
            | - Despide(z,v1,u) | - Trabajapara(x,v2,u)
            | Trabajapara(x,v2,despide(z,v1,u)).
-Persona(x) | - Estado(u) | -Empresa(v) | - Despide(x,v,u)
            | - Trabajapara(x,v,despide(x,v,u)).

% Tieneplazas(Empresa,Estado)

-Empresa(v) | - Estado(u) | $EQ(plazaslibres(v,u), 0)
            | Tieneplazas(v,u).
-Empresa(v) | - Estado(u) | $NE(plazaslibres(v,u), 0)
            | - Tieneplazas(v,u).
-Empresa(v) | - Estado(u) | - Persona(x) | - Ingresa(x,u)
            | - Tieneplazas(v,u) | Tieneplazas(v,ingresa(x,u)).
-Empresa(v) | - Estado(u) | - Oferta(v,y,u) | Tieneplazas(v,oferta(v,y,u)).
-Empresa(v) | - Empresa(v1)| (v = v1) | - Estado(u) | - Oferta(v1,y,u)
            | - Tieneplazas(v,u) | Tieneplazas(v,oferta(v1,y,u)).
-Empresa(v) | - Estado(u) | - Persona(x) | - $GT(plazaslibres(v,u),1)
            | - Contrata(x,v,u) | Tieneplazas(v,contrata(x,v,u)).
-Empresa(v) | - Empresa(v1) | (v = v1) | - Estado(u) | - Persona(x)
            | - Contrata(x,v1,u) | - Tieneplazas(v,u)
            | Tieneplazas(v,contrata(x,v1,u)).
-Empresa(v) | - Estado(u) | - Persona(x) | - Despide(x,v,u)
            | Tieneplazas(v,despide(x,v,u)).
-Empresa(v) | - Empresa(v1) | (v = v1) | - Estado(u) | - Persona(x)
            | - Despide(x,v1,u) | - Tieneplazas(v,u)
            | Tieneplazas(v,despide(x,v1,u)).

% plazaslibres(Empresa,Estado) = numero entero

-Empresa(v) | ( plazaslibres(v,nil) = 0 ).
-Empresa(v) | - Estado(u) | - Persona(x) | -Ingresa(x,u)
            |  (plazaslibres(v,ingresa(x,u)) = plazaslibres(v,u)).
-Empresa(v) | - Estado(u) | -Oferta(v,y,u)
     | ( plazaslibres(v,oferta(v,y,u)) = $SUM(plazaslibres(v,u),y)).
-Empresa(v) | -Empresa(v1) | (v = v1) | - Estado(u) | -Oferta(v1,y,u)
            | (plazaslibres(v,oferta(v1,y,u)) = plazaslibres(v,u)).
-Empresa(v) | - Estado(u) | - Persona(x) | -Contrata(x,v,u)
     | (plazaslibres(v,contrata(x,v,u)) = $DIFF(plazaslibres(v,u), 1)).
```

```
-Empresa(v) | Empresa(v1) | (v = v1) | - Estado(u) | - Persona(x)
     | -Contrata(x,v1,u)
     | (plazaslibres(v,contrata(x,v1,u)) = plazaslibres(v,u)).
-Empresa(v) | - Estado(u) | - Persona(x) | -Despide(x,v,u)
     | (plazaslibres(v,despide(x,v,u)) = $SUM(plazaslibres(v,u), 1)).
-Empresa(v) | Empresa(v1) | (v = v1) | - Estado(u) | - Persona(x)
     | -Despide(x,v1,u)
     | (plazaslibres(v,despide(x,v1,u)) = plazaslibres(v,u)).

% Recolector para  plazaslibres().

(plazaslibres(v,u) !=x) | $LT(x,0) | Plazaslibres(v,u,x).

% Clausulas para Oferta(Empresa, numero, Estado)

 -Estado(u)| -Empresa(v)| $GT(y,0)| -Oferta(v,y,u).

% Clausulas para Contrata(Persona, Empresa, Estado)

-Persona(x) | -Empresa(v)| -Contrata(x,v,nil).
-Persona(x) | -Estado(u)| -Empresa(v)| $GT(plazaslibres(v,u), 0)
            | -Contrata(x,v,u).
-Persona(x) | -Estado(u)| -Empresa(v) | EsParado(x,u)
            | - Contrata(x,v,u).

% Clausulas para Despide(Persona,Empresa,Estado).

-Persona(x) | -Empresa(v)| -Despide(x,v,nil).
-Persona(x) | -Estado(u) | -Empresa(v)| -Despide(x,v,u)
            | -EsParado(x,u).
-Persona(x) | -Estado(u) | -Empresa(v)| -Despide(x,v,u)
            | Trabajapara(x,v,u).

% Axioma reflexivo de igualdad.
(x = x).
end_of_list.
```

4 Verificación del esquema conceptual

El esquema conceptual especificado puede ser interrogado exhaustivamente para comprobar si un estado puede ser alcanzado desde un estado inicial despues de un conjunto de actualizaciones, qué condiciones son ciertas o no y bajo qué suposiciones será dicho estado alcanzado. A continuación se presentan algunos ejemplos que ilustran el modo de ejecución. Las reglas de razonamiento usadas son las de hiperresolución, UR–resolución y demodulación. En cada ejemplo se especifica el enunciado informal de la pregunta, el conjunto soporte empleado y un conjunto de demostraciones que la responde.

Pregunta 1: ¿Bajo qué condiciones se alcanzará un estado en que la empresa v tiene plazas libres después de que la empresa e1 ofrece 2 plazas y luego la empresa e2 ofrece x plazas?

```
list(sos).
```

```
Empresa(e1).
Empresa(e2).
(e1 != e2).
Oferta(e1,2,nil).
Oferta(e2,x,oferta(e1,2,nil)).
% negacion del objetivo:
-Tieneplazas(v,oferta(e2,x,oferta(e1,2,nil))) | $ANS(v).
end_of_list.

---------------- PROOF ----------------

1 [] Empresa(e1).
2 [] Empresa(e2).
4 [] Oferta(e1,2,nil).
5 [] Oferta(e2,x,oferta(e1,2,nil)).
6 [] -Tieneplazas(v,oferta(e2,x,oferta(e1,2,nil)))|$ANS(v).
7 [] Estado(nil).
9 [] -Estado(u)| -Empresa(v)| -Oferta(v,y,u)|Estado(oferta(v,y,u)).
34 [] -Empresa(v)| -Estado(u)| -Oferta(v,y,u)|Tieneplazas(v,oferta(v,y,u)).
65 [hyper,4,9,7,1] Estado(oferta(e1,2,nil)).
66 [ur,6,34,2,5] $ANS(e2)| -Estado(oferta(e1,2,nil)).
67 [binary,66,65] $ANS(e2).

---------------- PROOF ----------------

1 [] Empresa(e1).
2 [] Empresa(e2).
3 [] e1!=e2.
4 [] Oferta(e1,2,nil).
5 [] Oferta(e2,x,oferta(e1,2,nil)).
6 [] -Tieneplazas(v,oferta(e2,x,oferta(e1,2,nil)))|$ANS(v).
7 [] Estado(nil).
9 [] -Estado(u)| -Empresa(v)| -Oferta(v,y,u)|Estado(oferta(v,y,u)).
34 [] -Empresa(v)| -Estado(u)| -Oferta(v,y,u)|Tieneplazas(v,oferta(v,y,u)).
35 [] -Empresa(v)| -Empresa(v1)|v=v1| -Estado(u)| -Oferta(v1,y,u)
       | -Tieneplazas(v,u)|Tieneplazas(v,oferta(v1,y,u)).
64 [hyper,4,34,1,7] Tieneplazas(e1,oferta(e1,2,nil)).
65 [hyper,4,9,7,1] Estado(oferta(e1,2,nil)).
82 [ur,65,35,2,3,5,64,6] -Empresa(e1)|$ANS(e1).
83 [binary,82,1] $ANS(e1).
```

Respuesta: Cuando la empresa v es $e2$ y cuando es $e1$.

Pregunta 2: ¿Se alcanzará un estado en que alguien es desempleado si a pide empleo, luego b y luego c?.

```
list(sos).
Persona(a).
Persona(b).
Persona(c).
Ingresa(a,nil).
Ingresa(b,ingresa(a,nil)).
```

```
Ingresa(c,ingresa(b,ingresa(a,nil))).
% negacion del objetivo:
-EsParado(x,ingresa(c,ingresa(b,ingresa(a,nil)))) | $ANS(x).
end_of_list.

---------------- PROOF ----------------

1 [] Persona(a).
2 [] Persona(b).
3 [] Persona(c).
4 [] Ingresa(a,nil).
5 [] Ingresa(b,ingresa(a,nil)).
6 [] Ingresa(c,ingresa(b,ingresa(a,nil))).
7 [] -EsParado(x,ingresa(c,ingresa(b,ingresa(a,nil))))|$ANS(x).
8 [] Estado(nil).
9 [] -Estado(u)| -Persona(x)| -Ingresa(x,u)|Estado(ingresa(x,u)).
15 [] -Persona(x)| -Estado(u)| -Ingresa(x,u)|EsParado(x,ingresa(x,u)).
59 [hyper,4,9,8,1] Estado(ingresa(a,nil)).
61 [hyper,5,9,59,2] Estado(ingresa(b,ingresa(a,nil))).
63 [hyper,6,15,3,61] EsParado(c,ingresa(c,ingresa(b,ingresa(a,nil)))).
64 [binary,63,7] $ANS(c).

---------------- PROOF ----------------

1 [] Persona(a).
2 [] Persona(b).
3 [] Persona(c).
4 [] Ingresa(a,nil).
5 [] Ingresa(b,ingresa(a,nil)).
6 [] Ingresa(c,ingresa(b,ingresa(a,nil))).
7 [] -EsParado(x,ingresa(c,ingresa(b,ingresa(a,nil))))|$ANS(x).
8 [] Estado(nil).
9 [] -Estado(u)| -Persona(x)| -Ingresa(x,u)|Estado(ingresa(x,u)).
15 [] -Persona(x)| -Estado(u)| -Ingresa(x,u)|EsParado(x,ingresa(x,u)).
16 [] -Persona(x)| -Persona(z)|x=z| -Estado(u)| -Ingresa(z,u)
     | -EsParado(x,u)|EsParado(x,ingresa(z,u)).
59 [hyper,4,9,8,1] Estado(ingresa(a,nil)).
60 [hyper,5,15,2,59] EsParado(b,ingresa(b,ingresa(a,nil))).
61 [hyper,5,9,59,2] Estado(ingresa(b,ingresa(a,nil))).
63 [hyper,6,15,3,61] EsParado(c,ingresa(c,ingresa(b,ingresa(a,nil)))).
66 [hyper,60,16,2,3,61,6] c=b|EsParado(b,ingresa(c,ingresa(b,ingresa(a,nil)))).
70,69 [hyper,66,7] c=b|$ANS(b).
73 [back_demod,63,demod,70,70] EsParado(b,ingresa(b,ingresa(b,
       ingresa(a,nil)))).
74 [back_demod,7,demod,70] -EsParado(x,ingresa(b,ingresa(b,
       ingresa(a,nil))))|$ANS(x).
75 [binary,74,73] $ANS(b).

---------------- PROOF ----------------

1 [] Persona(a).
```

```
2 [] Persona(b).
3 [] Persona(c).
4 [] Ingresa(a,nil).
5 [] Ingresa(b,ingresa(a,nil)).
6 [] Ingresa(c,ingresa(b,ingresa(a,nil))).
7 [] -EsParado(x,ingresa(c,ingresa(b,ingresa(a,nil))))|$ANS(x).
8 [] Estado(nil).
9 [] -Estado(u)| -Persona(x)| -Ingresa(x,u)|Estado(ingresa(x,u)).
15 [] -Persona(x)| -Estado(u)| -Ingresa(x,u)|EsParado(x,ingresa(x,u)).
16 [] -Persona(x)| -Persona(z)|x=z| -Estado(u)| -Ingresa(z,u)
      | -EsParado(x,u)|EsParado(x,ingresa(z,u)).
58 [hyper,4,15,1,8] EsParado(a,ingresa(a,nil)).
59 [hyper,4,9,8,1] Estado(ingresa(a,nil)).
60 [hyper,5,15,2,59] EsParado(b,ingresa(b,ingresa(a,nil))).
61 [hyper,5,9,59,2] Estado(ingresa(b,ingresa(a,nil))).
62 [hyper,58,16,1,2,59,5] b=a|EsParado(a,ingresa(b,ingresa(a,nil))).
63 [hyper,6,15,3,61] EsParado(c,ingresa(c,ingresa(b,ingresa(a,nil)))).
66 [hyper,60,16,2,3,61,6] c=b|EsParado(b,ingresa(c,ingresa(b,ingresa(a,nil)))).
68 [hyper,62,16,1,3,61,6] b=a|c=a|EsParado(a,
      ingresa(c,ingresa(b,ingresa(a,nil)))).
70,69 [hyper,66,7] c=b|$ANS(b).
71 [back_demod,68,demod,70,70] b=a|EsParado(a,
      ingresa(b,ingresa(b,ingresa(a,nil)))).
73 [back_demod,63,demod,70,70] EsParado(b,
      ingresa(b,ingresa(b,ingresa(a,nil)))).
74 [back_demod,7,demod,70] -EsParado(x,
      ingresa(b,ingresa(b,ingresa(a,nil))))|$ANS(x).
79,78 [hyper,71,74] b=a|$ANS(a).
81 [back_demod,74,demod,79,79] -EsParado(x,ingresa(a,
      ingresa(a,ingresa(a,nil))))|$ANS(x).
82 [back_demod,73,demod,79,79,79] EsParado(a,
      ingresa(a,ingresa(a,ingresa(a,nil)))).
83 [binary,82,81] $ANS(a).
```

Respuesta: Cuando dicha persona es a, b o c.

Pregunta 3: ¿Cuántas plazas libres tiene la empresa $e1$, si oferta 2 plazas, contrata al parado p y luego oferta otras 3 plazas?.

```
list(sos).
Persona(p).
Empresa(e1).
Ingresa(p,nil).
Oferta(e1,2,ingresa(p,nil)).
Contrata(p,e1,oferta(e1,2,ingresa(p,nil))).
Oferta(e1,3,contrata(p,e1,oferta(e1,2,ingresa(p,nil)))).
%a = oferta(e1,3,contrata(p,e1,oferta(e1,2,ingresa(p,nil)))).
%Estado(a).
% negacion del objetivo:
-Plazaslibres(e1,oferta(e1,3,contrata(p,e1,oferta(e1,2,ingresa(p,nil)))),x)
  | $ANS(x).
end_of_list.
```

```
---------------- PROOF ----------------

1 [] Persona(p).
2 [] Empresa(e1).
3 [] Ingresa(p,nil).
4 [] Oferta(e1,2,ingresa(p,nil)).
5 [] Contrata(p,e1,oferta(e1,2,ingresa(p,nil))).
6 [] Oferta(e1,3,contrata(p,e1,oferta(e1,2,ingresa(p,nil)))).
7 [] -Plazaslibres(e1,oferta(e1,3,contrata(p,e1,
     oferta(e1,2,ingresa(p,nil)))),x)|$ANS(x).
8 [] Estado(nil).
9 [] -Estado(u)| -Persona(x)| -Ingresa(x,u)|Estado(ingresa(x,u)).
10 [] -Estado(u)| -Empresa(v)| -Oferta(v,y,u)|Estado(oferta(v,y,u)).
12 [] -Estado(u)| -Persona(x)| -Empresa(v)| -Contrata(x,v,u)
     |Estado(contrata(x,v,u)).
41 [] -Empresa(v)|plazaslibres(v,nil)=0.
42 [] -Empresa(v)| -Estado(u)| -Persona(x)| -Ingresa(x,u)
     |plazaslibres(v,ingresa(x,u))=plazaslibres(v,u).
43 [] -Empresa(v)| -Estado(u)| -Oferta(v,y,u)
     |plazaslibres(v,oferta(v,y,u))=$SUM(plazaslibres(v,u),y).
45 [] -Empresa(v)| -Estado(u)| -Persona(x)| -Contrata(x,v,u)
     |plazaslibres(v,contrata(x,v,u))=$DIFF(plazaslibres(v,u),1).
49 [] -Empresa(v)| -Estado(u)|plazaslibres(v,u)!=x|$LT(x,0)
     |Plazaslibres(v,u,x).
57 [] x=x.
60,59 [hyper,2,41] plazaslibres(e1,nil)=0.
63,62 [hyper,3,42,2,8,1,demod,60] plazaslibres(e1,ingresa(p,nil))=0.
65 [hyper,3,9,8,1] Estado(ingresa(p,nil)).
68,67 [hyper,4,43,2,65,demod,63]
   plazaslibres(e1,oferta(e1,2,ingresa(p,nil)))=2.
71 [hyper,4,10,65,2] Estado(oferta(e1,2,ingresa(p,nil))).
74,73 [hyper,5,45,2,71,1,demod,68]
   plazaslibres(e1,contrata(p,e1,oferta(e1,2,ingresa(p,nil))))=1.
78 [hyper,5,12,71,1,2] Estado(contrata(p,e1,oferta(e1,2,ingresa(p,nil)))).
81,80 [hyper,6,43,2,78,demod,74]
   plazaslibres(e1,oferta(e1,3,contrata(p,e1,oferta(e1,2,ingresa(p,nil)))))=4.
84 [hyper,6,10,78,2]
   Estado(oferta(e1,3,contrata(p,e1,oferta(e1,2,ingresa(p,nil))))).
86 [hyper,84,49,2,57,demod,81,81]
   Plazaslibres(e1,oferta(e1,3,contrata(p,e1,oferta(e1,2,ingresa(p,nil)))),4).
87 [binary,86,7] $ANS(4).
```

Respuesta: Tiene 4 plazas libres.

5 Conclusiones

En este trabajo se ha introducido la idea de describir el esquema conceptual de una base de datos como una teoría formal en lógica de primer orden con igualdad y predicados aritméticos. El lenguaje clausal empleado es un medio expresivo y sencillo para realizar una especificación detallada del sistema.

Usando las reglas de inferencia de hiperresolución, UR–resolución y paramodulación se puede probar en dicha teoría enunciados que recojen el sentido de consultas complejas sobre la base de datos; la instanciación de las variables en el momento de finalizar la prueba, dan la respuesta a la pregunta.

La animación de especificaciones basada en razonamiento automático que se ha propuesto aquí, aporta al diseñador de bases de datos una facilidad extra de depuración: cuando el sistema da una respuesta erronea, la demostración correspondiente señala explicitamente el conjunto de cláusulas responsables del comportamiento indeseable en el sistema.

References

[1] M. Alpuente. *El Lenguaje CLP(H/E): Una aproximación basada en restricciones a la integración de la programación lógica y ecuacional.* Tesis Dr. Univ. de Valencia, 1991.

[2] ANSI/X3/SPARC Study Group, Data Management Systems, *Framework Report on Database Management Systems*,AFIPS Press, 1978.

[3] H.D. Ehrich, H.J. Kreowski y H. Weber. Algebraic Specification Scheme for Database Systems. *Proc. 4th Conf. on Very Large Data Bases VLDB'78.* Berlin, 1978.

[4] H.D. Ehrich, K. Drosten y M. Gogolla. Towards an Algebraic Semantics for Database Spacification, Knowledge and Data. *Proc. IFIP WG 2.6 Working Conference on Database Semantics.* Albufeira, Portugal, 1986. North-Holland, Amsterdam, 1986.

[5] H. Gallaire, J. Minker y J.M. Nicolas. *Advances in Database Theory.* Plenum Press, New York, 1984.

[6] M. Gogolla. Algebraization and Integrity constraints for an Extended Entity-Relationship Approach. *Proc Int'l Joint Conf. on Theory and Practice of Software Development TAPSOFT'89, Lecture Notes in Computer Science.* 351, 259–273. Springer–Verlag, Berlin, 1989.

[7] M. Gustafsson, T. Karlsson y J. Bubenko. A declarative approach to Conceptual Information Modelling. *Information Systems Design metodologies: A comparative review*, 93–142. North–Holland, Amsterdam, 1982.

[8] J.W. Lloyd. *Foundations of logic programming.* Springer–Verlag, Berlin, 1987.

[9] W.W. McCune. OTTER 2.0 Users Guide, Technical Report ANL–90/9, Argonne National Lab., Argonne, Ill., 1990.

[10] P. Veloso, J. Castilho y A. Furtado. Systematic Derivation of Complementary Specifications. *Proc. 7th Int'l Conf. on Very Large Data Bases VLDB'81*, Cannes, 409–421. IEEE Computer Society Press, 1981.

[11] P. Veloso y A. Furtado. Towards simpler and yet complete formal specifications. *Information systems: Theoretical and Formal aspects.* North–Holland, Amsterdam, 1985.

[12] S. Winker y L. Woss. Procedure Implementation through Demodulation and Related Tricks. *Proceedings of the Sixth Conference on Automated Deduction*, Lecture Notes in Comp. Sci., 138, Springer–Verlag, New York, 1982.

[13] L. Woss, R. Overbeek, E. Lusk y J. Boyle. *Automated Reasoning: Introduction and Aplications.* Prentice–Hall, Enlewood Cliffs, New York, 1984.

[14] L. Woss y W.W. McCune. Authomated Theorem Proving and Logic Programming: A Natural Symbiosis. J. Logic Programming 1991, 11, 1–53.