DES: un recurso para el aprendizaje de bases de datos deductivas

Fernando Sáenz Pérez
Dept. Ingeniería del Software e Inteligencia Artificial
Universidad Complutense de Madrid
Facultad de Informática, c/ Profesor José García Santesmases, s/n, 28040 Madrid
fernan@sip.ucm.es

Resumen

En este artículo se presenta DES (Datalog Educational System), un sistema con el que se pueden transmitir los conceptos fundamentales de las bases de datos deductivas con Datalog como lenguaje de acceso a datos. Además, dado que está implementado en Prolog, los estudiantes que conozcan este lenguaje pueden estudiar su implementación en cursos más avanzados. El motivo de su desarrollo pareció evidente en su momento al no existir un sistema de uso fácil y multiplataforma. El sistema incorpora recursión no lineal (a diferencia de SQL) y negación estratificada. Se puede usar tanto desde un intérprete Prolog (por lo tanto, sobre cualquier plataforma hardware/software que admita tal intérprete) como desde ejecutables para Windows, Linux y Unix. El sistema se distribuye desde Sourceforge bajo licencia GPL, y su código es abierto y gratuito. Ha suscitado interés en la comunidad docente e investigadora internacional desde la publicación de su primera versión en el año 2003, y se publican nuevas versiones anualmente.

1. Introducción

La intersección de las bases de datos, la lógica y la inteligencia artificial ha dado lugar a las bases de datos deductivas. Una base de datos deductiva es un sistema que incluye mecanismos para definir reglas (deductivas) que pueden inferir o deducir información adicional de los hechos almacenados en la base de datos. Las reglas se especifican en un lenguaje declarativo y un motor de inferencia (el mecanismo de deducción) es capaz de deducir nuevos hechos de la base de datos. El modelo usado para las bases de datos deductivas está estrechamente relacionado con el modelo de datos relacional y en particular con el formalismo de cálculo relacional de dominios. Está también relacionado con el lenguaje Prolog y, principalmente, con el lenguaje Datalog, un subconjunto de Prolog que evita el uso de estructuras potencialmente infinitas.

El álgebra relacional se ha mostrado insuficiente en la expresión de consultas a bases de datos. Un defecto importante es la ausencia de recursión, que impide expresar consultas sencillas como es el cierre transitivo de un grafo. Este tipo de consultas pueden expresarse en la lógica de predicados de primer orden. No obstante, a partir del estándar SQL:1999 para SQL (el lenguaje *de facto* de bases de datos relacionales) se incluyó una forma limitada de recursión: la recursión lineal.

El área de las bases de datos deductivas y en particular el procesamiento recursivo de consultas se reveló muy activo en 1984 con el inicio de tres proyectos importantes, dos en Estados Unidos y otro en Europa. Los proyectos Nail!, en Stanford, LDL, en Austin y EKS de ECRC han producido contribuciones importantes y la construcción de prototipos simples. En la Universidad de Wisconsin se desarrolló Coral [11] a finales de los 80 (véase también el apartado 2).

En este artículo se presenta un sistema que se puede usar como recurso docente para la enseñanza de las bases de datos deductivas que disponen de Datalog como lenguaje de consulta. Estas enseñanzas están orientadas a las asignaturas de planes de estudios relativas a bases de datos avanzadas e ingeniería del conocimiento. En el siguiente apartado se comentan los sistemas relacionados. En el apartado 3 se listan sus características y limitaciones. El sistema se describe en el apartado 4 y se plantean algunos ejemplos de uso en la enseñanza en el 5. En el apartado 6 se comenta su aplicabilidad y utilidad en el contexto internacional y, finalmente, el apartado 7 recoge las conclusiones y posible trabajo futuro.

2. Sistemas relacionados

La comunidad científica ha dedicado un enorme esfuerzo en la investigación de bases de datos deductivas [13] (suscitando muchos talleres y conferencias) y ha dado lugar a varios sistemas. Sin embargo, ninguno de ellos proporciona un entorno fácil, amigable y multiplataforma cuyo objeto sea la introducción de este tema de estudio a los alumnos. No obstante, se pueden comentar algunos de los sistemas de bases de datos deductivas más representativos.

El sistema de bases de datos deductivas LOLA [23] está basado en un lenguaje de cláusulas declarativas que da soporte a la recursión, los términos compuestos, negación, agregación y predicados predefinidos. El motor deductivo está basado en el paradigma de evaluación ascendente con operaciones relacionales. Está disponible mediante una interfaz para clientes Web.

El proyecto LDL en MCC originó el prototipo LDL++ [21], un sistema de bases de datos deductivas con características como negación tanto estratificada como no estratificada, términos de tipo conjunto y agregados. Actualmente se puede usar via Internet con un cliente con capacidades Java.

XSB [12] (http://xsb.sourceforge.net/) es un sistema Prolog extendido que se puede usar para aplicaciones de bases de datos deductivas. Se caracteriza por implementar una semántica bien fundada para reglas con literales negativos en su cuerpo e implementa mecanismos de tablas de extensión. Funciona sobre los sistemas operativos Unix/Linux y Windows. Datalog++ [16] es una fachada para el sistema de bases de datos deductivas XSB.

Coral [11] es un sistema deductivo con un lenguaje de consulta declarativo que da soporte a cláusulas generales de Horn aumentadas con términos compuestos, agrupación por conjuntos, agregación, negación y relaciones con tuplas que contienen variables cuantificadas universalmente. Sólo funciona sobre plataformas Unix. Existe también una versión que permite características de la programación orientada a objetos: Coral++ [15].

FLORID (F-LOgic Reasoning In Databases) [8] es un sistema de bases de datos deductivas orientadas a objetos que emplea F-Logic como lenguaje de consulta y de definición de datos. Con el interés creciente por los datos semiestructurados, FLORID se ha extendido para tratar este tipo de datos en el contexto de la integración de datos en la Web semántica.

El proyecto NAIL! proporcionó un prototipo con negación estratificada, negación bien fundada y negación estratificada modularmente. Posteriormente se añadió el lenguaje Glue, que consiste esencialmente en reglas lógicas con instrucciones SQL incorporadas en un lenguaje imperativo convencional [10,7]. El enfoque de la combinación resulta similar a Coral, que usa C++. No funciona sobre plataformas Windows.

Otro sistema de bases de datos deductivas que sigue esta combinación de lenguajes declarativos e imperativos es Rock&Roll [2].

ADITI 2 [19] es la versión actual de un sistema de bases de datos deductivas que usa el lenguaje de programación lógico-funcional Mercury. No funciona sobre plataformas Windows y se ha detenido su desarrollo.

El único sistema de bases de datos deductivas comercial ha sido SDS (Smart Data System) y su lenguaje de consulta declarativo DECLARE (Declarative Reasoning) [9], con soporte para la negación estratificada y para los conjuntos.

No obstante, el mayor impacto de las bases de datos deductivas en el ámbito comercial quizás haya sido la incorporación de la recursión lineal en la versión SQL del sistema relacional DB2 de IBM con técnicas de implementación basadas en los conjuntos mágicos [3].

3. Características del sistema DES

Como ventajas del sistema se pueden enumerar:

- Sistema multiplataforma. Se ejecuta en la mayoría de los sistemas operativos actuales, bien sea a partir de su código fuente Prolog o mediante los ejecutables proporcionados.
- Recursión no lineal. Admite múltiples llamadas recursivas en el cuerpo de una regla, incluyendo llamadas recursivas con negación (aunque la negación en sí no puede aparecer en un ciclo; véase el siguiente punto).
- Negación estratificada [18]. La negación estratificada significa *grosso modo* que un objetivo negado no puede llamarse a sí mismo en ningún camino de cómputo con objeto de asegurar la corrección de la respuesta y así evitar

situaciones como la conocida paradoja de Russell. La negación puede aparecer en uno o varios literales negados del cuerpo de las reglas. El sistema implementa un algoritmo ascendente guiado por el objetivo y con cómputos descendentes que calcula de forma segura las respuestas.
- Vistas transitorias. Permiten emitir consultas conjuntivas al vuelo, es decir, desde el inductor de comandos y sin necesidad de crear una relación *ex profeso* para ello (véase el apartado 4.4).
- Implementación con técnicas de tablas de extensión [6,17] que mejoran significativamente el rendimiento del sistema con respecto a técnicas ascendentes ingenuas.

Algunas de las limitaciones del sistema que se pueden enumerar incluyen:

- Agregados. El sistema no dispone de funciones de agregación, que se demuestran muy útiles en la práctica. Su implementación no es sencilla, aunque está prevista.
- Aritmética. Aún no se ha implementado la posibilidad de incluir primitivas aritméticas, aunque es una labor sencilla e incluida en el trabajo futuro.
- La interfaz de usuario del sistema es un intérprete de comandos que acepta tanto consultas Datalog como comandos del sistema. Sin embargo, se debería ampliar esta interfaz a una interfaz gráfica de usuario (véase también el apartado de conclusiones).
- Depuración. Actualmente no hay herramientas para la depuración de programas Datalog, pero se están investigando actualmente [4].
- Incapacidad de representar términos compuestos. Los términos compuestos pueden introducir problemas de no terminación. La terminación es una característica que se exige *de facto* a los sistemas de bases de datos, y se podría asegurar limitando la profundidad de los términos a una cota finita. Una correspondencia entre constantes y un conjunto finito de términos construidos podría dar lugar a una implementación sencilla.

4. Descripción del sistema DES

Dado que Datalog se derivó de Prolog, se han adoptado prácticamente todos los convenios sintácticos de Prolog para la escritura de programas Datalog. Sin embargo, los comandos son diferentes a los que están acostumbrados los programadores de Prolog (véase el apartado 4.2).

4.1. Instalación e inicio

El sistema educativo DES se encuentra en su página de distribución (http://des.sourceforge.net). El sistema se puede usar a partir de su código fuente Prolog sobre cualquiera de los siguientes sistemas Prolog: GNU Prolog [5], Ciao Prolog [1], Sicstus Prolog [14] y SWI Prolog [20]. No obstante, el método recomendado es instalar uno de los ejecutables para Windows, Linux o Unix. En esta página de distribución se incluye el manual de usuario con instrucciones más detalladas sobre la instalación del sistema. No obstante, elegir el método recomendado supone, para la instalación, descomprimir simplemente el archivo de distribución y, para su inicio, ejecutar un fichero.

4.2. Uso del sistema

Al iniciar el sistema se muestra lo ilustrado en la figura 1. La última línea (**DES>**) es el inductor de Datalog, en el que se pueden escribir comandos o consultas Datalog. Si el sistema se ejecuta desde un intérprete Prolog y en algún momento se sale del sistema por algún error se puede escribir **des.** en el inductor de Prolog para continuar.

```
*********************************************************
*                                                       *
* DES: Datalog Educational System v.1.2.0               *
*                                                       *
* - Stratified Negation                                 *
* - Full recursion                                      *
* - Noncompound terms                                   *
*                                                       *
* Type "/help" for commands                             *
* Type "des." if you get out of DES                     *
*   from a Prolog interpreter                           *
*                                                       *
*                          Fernando Sáenz (c) 2004-2007 *
*                                              DISIA UCM *
*            Please send comments, questions, etc. to: *
*                                    fernan@sip.ucm.es *
*                               Visit the Web site at: *
*                            http://des.sourceforge.net/ *
*********************************************************

DES>
```

Figura 1. El sistema DES

El modo habitual de uso del sistema es crear mediante cualquier editor de texto en el sistema operativo un fichero con extensión **.dl** para guardar el programa Datalog que se desarrolle. Una vez se haya escrito un programa Datalog se puede cargar con el comando:

```
/consult NombreArchivo
```

Donde ***NombreArchivo*** es el nombre de archivo, como **relop.dl** (la extensión predeterminada **.dl** se puede omitir) que en este caso es un programa de ejemplo de la distribución y que muestra la versión Datalog de las operaciones relacionales básicas. Obsérvese que los comandos comienzan por una barra de división y la especificación de argumentos es similar a los intérpretes de comandos de un sistema operativo (con espacios separadores en lugar de paréntesis y comas).

Este archivo de la distribución alojado en la carpeta de ejemplos contiene un programa que reproduce las operaciones relacionales básicas mediante reglas Datalog. Contiene tres relaciones (**a**, **b** y **c**) que se usan como argumentos de las operaciones relacionales.

```
% Relations
a(a1).      b(b1).      c(a1,b2).
a(a2).      b(b2).      c(a1,a1).
a(a3).      b(a1).      c(a2,b2).

% Relational Operations

% pi(X)(c(X,Y))
projection(X) :- c(X,Y).

% sigma(X=a2)(a)
selection(X) :- a(X), X=a2.

% a X b
cartesian(X,Y) :- a(X), b(Y).

% a |x| c
join(X) :- a(X), c(X,Y).

% a U b
union(X) :- a(X).
union(X) :- b(X).

% a - b
difference(X) :- a(X), not(b(X)).
```

Los programas Datalog pueden incluir comentarios (indicados por el símbolo **%** y se extienden hasta el final de la línea).

Si este archivo está en el directorio de inicio del sistema, se podría escribir:

```
/consult relop
```

Una vez cargado el programa se puede emitir una consulta Datalog, como:

```
DES> projection(X)
```

que da como resultado:

```
{
  projection(a1),
  projection(a2)
}
```

El resultado de la evaluación de una consulta es el conjunto de hechos derivado de la consulta en el contexto del programa de forma tanto intensional (con las reglas que definen nuevas relaciones a partir de otras relaciones) como extensional (con las reglas especificadas como hechos). En este ejemplo, **projection(X)** se corresponde con la operación relacional proyección sobre el primer argumento de la relación **c**.

Para que el alumno comprenda el diferente modo de evaluar y entender las consultas Datalog con respecto a Prolog, se puede usar el comando:

```
DES> /prolog projection(X)
projection(a1)
? (type ; for more solutions, <Intro>
to continue) ;
projection(a1)
? (type ; for more solutions, <Intro>
to continue) ;
projection(a2)
? (type ; for more solutions, <Intro>
to continue) ;
no
```

Obsérvese la diferencia en la ejecución Datalog y la ejecución Prolog. El primero aporta el significado de la relación como un conjunto y el segundo busca soluciones, una a una, que satisfagan un objetivo; por ello se obtienen tres respuestas (con **projection(a1)** repetido debido a la definición de **c**).

Dado que DES está implementado con técnicas de tablas de extensión, se puede consultar el significado de las relaciones calculadas con el comando **/list_et**, que muestra todas las que se han calculado hasta el momento (incluyendo consultas anteriores). Además, muestra las últimas llamadas que se han producido en la última iteración de punto fijo, que puede ser útil si se desea estudiar su implementación:

```
Answers:
{
  projection(a1),
  projection(a2),
  c(a1,a1),
```

```
  c(a1,b2),
  c(a2,b2)
}
Calls:
{
  projection(_10771),
  c(_10766,_10767)
}
```

A menos que se emita una vista transitoria (véase el apartado 4.4) o el comando **/clear_et** (que borra el contenido de la tabla de extensión), se conservarán las respuestas calculadas.

4.3. Consultas

Una consulta es el nombre de una relación del programa con tantos argumentos como la aridad de la relación y que se escribe en el inductor DES. Cada uno de ellos puede ser una variable o una constante, pero no un término compuesto. Obsérvese que no se pueden escribir consultas conjuntivas, como **a(X), b(X)** para calcular la intersección de ambas relaciones. Sin embargo, esta situación se puede abordar desde dos puntos de vista: (1) escribir en el programa una regla **r(X) :- a(X), b(X)** (o insertarla con el programa cargado mediante el comando **/assert**) y emitir la consulta **r(X)**, y (2) usar vistas transitorias (véase el apartado 4.4).

La respuesta a una consulta con variables para todos sus argumentos resulta en el conjunto completo de hechos que definen su significado. Si una consulta contiene una constante en uno o más de sus argumentos, en el resultado se descartan los hechos que definen su significado pero que no casan con el patrón de la consulta. Ésta es una operación análoga a la selección del álgebra relacional, como, por ejemplo, **a(a3)** en el ejemplo anterior.

4.4. Vistas transitorias

En este sistema se ha introducido un concepto nuevo en los sistemas de consulta a bases de datos deductivas, las vistas transitorias, que permiten escribir consultas conjuntivas al vuelo. Una vista transitoria es una regla que se añade a la base de datos y su cabeza (parte izquierda) se considera como una consulta que se ejecuta. Después de su evaluación, la regla se elimina de la base de datos. Las vistas transitorias son útiles para emitir rápidamente consultas conjuntivas, como, por ejemplo, en:

```
DES> d(X) :- a(X), not(b(X))
```

que calcula la diferencia de conjuntos entre las relaciones **a** y **b**.

Obsérvese que la vista se calcula en el contexto del programa, por lo que si se tienen definidas más reglas para la misma relación de consulta (expresada en la cabeza de la vista), su evaluación devolverá el significado completo de esta relación considerando el resto de reglas del programa. Por ejemplo:

```
DES> a(X) :- b(X)
```

calcula la unión de conjuntos de las relaciones **a** y **b** en el contexto del programa **relop.dl**, dando como resultado:

```
{
  a(a1),
  a(a2),
  a(a3),
  a(b1),
  a(b2)
}
```

4.5. Reglas

Las reglas Datalog son similares a las cláusulas Prolog con las mismas limitaciones indicadas para las consultas. Estas reglas tienen la forma **cabeza :- cuerpo**, o simplemente **cabeza**, terminando ambas en un punto. Una cabeza es un término compuesto, y un cuerpo es una secuencia de objetivos separados por comas que pueden contener operadores predefinidos. Los objetivos pueden aparecer negados como **not(objetivo)**.

4.6. Programas

Los programas consisten en un conjunto de reglas en el que su orden no es importante. Pueden contener comentarios (**%**).

4.7. Detalles de la implementación

A diferencia de muchas implementaciones de Datalog, que usan una evaluación ascendente (*bottom-up*), este intérprete usa una evaluación ascendente guiada por una descendente (*top-down*). En lugar de calcular el significado del programa completo, calcula el significado de la consulta emitida usando tablas de extensión [6] como técnica de optimización.

5. Ejemplos

En este apartado se muestran algunos ejemplos ilustrativos del uso de DES en la enseñanza de Datalog y que ilustran algunas de sus características más relevantes (entre paréntesis se indica el nombre del fichero que se pueden encontrar en la carpeta **examples** de la distribución).

5.1. Caminos de un grafo (fichero paths.dl)

Este programa introduce el uso de la recursión en DES definiendo el grafo de la figura 2 y el conjunto de tuplas <origen, destino> tal que existe un camino desde el origen hasta el destino

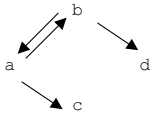

Figura 2. Caminos de un grafo

```
% Paths in a Graph
edge(a,b).      edge(a,c).
edge(b,a).      edge(b,d).

path(X,Y) :- path(X,Z), edge(Z,Y).
path(X,Y) :- edge(X,Y).
```

La consulta path(X,Y) devuelve:

```
{
  path(a,a),
  path(a,b),
  path(a,c),
  path(a,d),
  path(b,a),
  path(b,b),
  path(b,c),
  path(b,d)
}
```

que se interpreta como el cierre transitivo de la relación **path**.

5.2. Negación estratificada (fichero negation.dl)

DES asegura que la información negativa obtenida a partir de un programa con objetivos negados es correcta siempre que el programa admita una estratificación. El siguiente programa lo ilustra:

```
a :- not(b).
b :- c,d.
c :- b.
c.
```

La consulta **a** tiene éxito con significado **{a}**. Obsérvese que el significado de la consulta **not(a)** es el conjunto vacío.

Si se estudia la resolución de consultas Datalog en presencia de negación, los siguientes comandos resultan de utilidad:

```
DES> /pdg
Nodes: [d/0,a/0,b/0,c/0]
Arcs : [a/0-b/0,c/0+b/0,
        b/0+d/0,b/0+c/0]

DES> /strata
[(d/0,1),(a/0,2),(b/0,1),(c/0,1)]
```

El primer comando muestra el grafo de dependencias de predicados (relaciones) (véase, e.g., [22]) para el programa cargado. En primer lugar se listan los nodos en una lista cuyos elementos R son predicados con sus aridades de la forma relación/aridad. A continuación se muestran los arcos como una lista cuyos elementos son P+Q o P-Q, donde P y Q son nodos del grafo. Un arco P+Q significa que existe una regla tal que P es la relación para su cabeza y Q es la relación de uno de sus literales. Si el literal está negado, el arco es negativo, lo cual se expresa por P-Q. El grafo de este programa se ilustra en la figura 3.

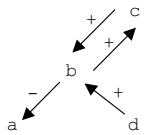

Figura 3. Grafo de dependencias de predicados (relaciones) para **negation.dl**

El segundo comando muestra el estrato asignado a cada relación. Los estratos se muestran como una lista de pares (R,S), donde R es una relación y S es su estrato. En este ejemplo todas las relaciones se encuentran en el estrato 1 salvo **a**, que se encuentra en el estrato 2. Esto significa que si se calcula el significado de **a**, es necesario tener calculado completamente el o los estratos inferiores (en este caso sólo el estrato 1) y sólo para las relaciones que intervengan para calcular el significado de **a**.

Para ilustrar que sólo se calculan las relaciones necesarias, considérese la consulta **b** para el mismo programa. DES calcula el subgrafo de

dependencias para **b**, i.e., el grafo en el que aparecen todas las relaciones alcanzables a partir de **b**; después se calcula de nuevo una estratificación. Obsérvese que el sistema informa de los estratos calculados (siempre que la salida informativa esté activada, que es el comportamiento predeterminado y que se puede modificar con los comandos **/verbose** y **/noverbose**):

```
DES> a
Info: Computing by stratum of [b].
{
  a
}
DES> b
{
}
```

Para la consulta **a**, el sistema informa de que **b** se calcula previamente (usando la técnica de optimización basada en tablas de extensión), mientras que no es necesario para la consulta **b** recurrir al algoritmo de cómputo por estratos.

5.3. Operadores

Todos los operadores son infijos salvo la negación. Los infijos se deben entender como tests en lugar de relaciones para asegurar finitud en la respuesta. Esto significa que la lectura declarativa de las reglas con estos operadores no es clara. Por ejemplo, una vez cargado el programa **relop.dl** y al añadir la siguiente regla (con el comando **/assert**):

```
less(X,Y) :- X<Y, c(X,Y).
```

la evaluación de la consulta **less(X,Y)** devuelve un significado vacío, pero si se permuta el orden de los objetivos en la regla, se obtiene:

```
{
  less(a1, b2),
  less(a2, b2)
}
```

Por lo tanto, no se aseguran respuestas correctas para los programas que usen primitivas sin sus argumentos cerrados (i.e., sin variables) dado que en general el orden de los objetivos que incluyan estos operadores afecta a la semántica. Es una consecuencia de las reglas inseguras [22], las cuales se pueden caracterizar sintácticamente aunque la versión actual del sistema no advierte de su presencia.

6. Utilidad del sistema

Según lo que conocemos, DES ha sido reconocido de utilidad en la comunidad internacional y aplicado en diferentes cursos de bases de datos y sistemas basados en conocimiento (EE.UU. y Europa). También se ha evaluado en un proyecto europeo y aplicado en el ámbito de la investigación. Está listado como un producto de interés por el grupo SIGMOD (Special Interest Group on Management of Data) de la sociedad ACM. Se ha demostrado como un sistema muy demandado e incluido en la lista de los diez sistemas más activos de Prolog en sourceforge.net, el sitio de referencia para la distribución de aplicaciones de código abierto, donde se han contabilizado en el último año más de 180 descargas mensuales de nuestro sistema.

7. Conclusiones

Nuestra experiencia al usar este recurso en la asignatura "Bases de datos y sistemas de información", junto con la realimentación por parte de usuarios (alumnos, profesores, investigadores y profesionales) de la comunidad internacional, ha permitido ajustarlo y mejorarlo para su aplicación en el entorno docente. Los resultados han sido muy satisfactorios (corroborado por colegas internacionales). No obstante, aún quedan aspectos importantes y pendientes de su implementación, tales como la integración de aritmética, los agregados, un entorno gráfico de desarrollo y herramientas de depuración. Markus Triska, moderador del grupo comp.lang.prolog, ha desarrollado un entorno Emacs para DES. Actualmente estamos desarrollando un entorno de desarrollo configurable (para DES en particular) denominado ACIDE (véase sourceforge). Finalmente, investigamos la depuración declarativa de programas Datalog [4].

Agradecimientos

El autor desea agradecer al grupo Clip por proporcionar su sistema Ciao Prolog gratuito y en particular a F. Bueno por su ayuda en la adaptación de DES. También a Jan Wielemaker y Daniel Diaz por proporcionar sus sistemas Prolog gratuitos, así como a Mats Carlsson por resolver rápidamente los aspectos de implementación en Sicstus, y a todos ellos por permitir usar sus listas de distribución como medio de difusión del sistema. Final-

mente agradecer la realimentación de todos los usuarios de DES y a los revisores de este trabajo.

Referencias

[1] F. Bueno, D. Cabeza, M. Carro, M. Hermenegildo, P. López–García, and G. Puebla. “The Ciao Prolog system. Reference manual”, School of Computer Science, Technical University of Madrid (UPM), 1997.

[2] M.L. Barja, N.W. Paton, A. Fernandes, M.H. Williams, A. Dinn, “An Effective Deductive Object–Oriented Database Through Language Integration”, In Proc. of the 20th VLDB Conference, 1994.

[3] C. Beeri and R. Ramakrishnan, "On the Power of Magic", Journal of Logic Programming, 10(3,4):255-299, 1991.

[4] R. Caballero, Y. García-Ruiz, and F. Sáenz-Pérez, “Towards a Set Oriented Calculus for Logic Programming”, PROLE 2006, pp. 41-50, Barcelona, 2006.

[5] D. Diaz, GNU Prolog, http://www.gnu.org/software/prolog.

[6] S.W. Dietrich, “Extension Tables: Memo Relations in Logic Programming”, IV IEEE Symposium on Logic Programming, 1987.

[7] M. Derr, S. Morishita, and G. Phipps, “Design and Implementation of the Glue–NAIL Database System”, In Proc. of the ACM SIGMOD International Conference on Management of Data, pp. 147–167, 1993.

[8] M. Kifer, G. Lausen, J. Wu, "Logical Foundations of Object Oriented and Frame Based Languages", Journal of the ACM, vol. 42, p. 741-843, 1995.

[9] W. Kiessling, H. Schmidt, W. Strauss, and G. Dünzinger, “DECLARE and SDS: Early Efforts to Commercialize Deductive Database Technology”, VLDB Journal, 3, 1994.

[10] G. Phipps, M. A. Derr, and K.A. Ross, “Glue–NAIL!: A Deductive Database System”. In Proc. of the ACM SIGMOD Conf. on Management of Data, pp. 308–317, 1991.

[11] R. Ramakrishnan, D. Srivastava, S. Sudarshan, and P. Seshadri, “Implementation of the Coral Database System”, In Proc. of the ACM SIGMOD Conference on Management of Data, 1993.

[12] P. Rao, Konstantinos F. Sagonas, Terrance Swift, David Scott Warren, and Juliana Freire, “XSB: A System for Efficiently Computing WFS", Logic Programming and Non–monotonic Reasoning, 1997.

[13] R. Ramakrishnan and J.D Ullman, “A Survey of Research on Deductive Database Systems”, Journal of Logic Programming, 23(2): 125–149, 1995.

[14] SICS, Sicstus Prolog, http://www.sics.se/sicstus.

[15] D. Srivastava, R. Ramakrishnan, S. Sudarshan, and P. Seshadri, “Coral++: Adding Object–Orientation to a Logic Database Language”, Proceedings of the International Conference on Very Large Databases, 1993.

[16] Z. Tang, "Datalog++: An Object-Oriented Front-End For The Xsb Deductive Database Management System", http://citeseer.ist.psu.edu/tang99datalog.html.

[17] H. Tamaki and T. Sato, “OLD Resolution with Tabulation”, Proceedings of ICLP’86, Lecture Notes on Computer Science 225, Springer–Verlag, 1986.

[18] J.D. Ullman. Database and Knowledge-Base Systems, Computer Science Press, 1995.

[19] J. Vaghani, K. Ramamohanarao, D.B. Kemp, Z. Somogyi, and P.J. Stuckey, “Design Overview of the Aditi Deductive Database System”, In Proc. of the 7th Intl. Conf. on Data Engineering, pp. 240–247, 1991.

[20] J. Wielemaker, SWI Prolog http://www.swi-prolog.org.

[21] C. Zaniolo, N. Arni, and K. Ong, “Negation and Aggregates in Recursive Rules: the LDL++ Approach”, In Proc. Intl. Conf. on Deductive and Object Oriented DBs, 1993.

[22] C. Zaniolo, S. Ceri, C. Faloutsos, T.T. Snodgrass, V.S. Subrahmanian, and R. Zicari, "Advanced Database Systems", Morgan Kauffmann Publishers, 1997.

[23] U. Zukowski and B. Freitag, “The Deductive Database System LOLA”, In: J. Dix and U. Furbach and A. Nerode (Eds.). Logic Programming and Nonmonotonic Reasoning. LNAI 1265, pp. 375–386. Springer, 1997.