Datalog Educational System V1.1

User's Manual

Technical Report SIP 139-04

Fernando Sáenz Pérez
Departamento de Sistemas Informáticos y Programación
Universidad Complutense de Madrid
4/3/2004

[1] **http://www.fsf.org/**

Contents

1. Introduction

The Datalog Educational System (DES) is a free Prolog-based implementation of a basic deductive database with stratified negation which uses Datalog as a query language. The system is implemented on top of Prolog and you can use it from a Prolog interpreter running both on Windows and Unix/Linux. Moreover, both Windows and Unix executables are also provided.

We have developed it aiming to have a simple, interactive, multiplatform, and affordable system (not necessarily efficient) for students, so that they can get the fundamental concepts behind a deductive database with Datalog as a query language. In addition, since it is implemented on top of Prolog and students are assumed to know Prolog programming, they can study its implementation. Other known systems are not fully suited due to the absence of some characteristics DES does offer for our educational purposes (See section 7).

DES 1.1 is the current implementation, which enjoys full recursive evaluation with memoization techniques and stratified negation. It does not deal with compound terms, although they may come in future versions.

1.1 Deductive Databases

The intersection of databases, logic, and artificial intelligence delivered deductive databases. Deductive database systems are database management systems built around a logical model of data, and their query languages allows to express logical queries. The relational language SQL is a limited form of logical expression, and deductive database systems are advanced forms of relational systems.

A deductive database is a system which includes procedures for defining deductive rules which can infer information in addition to the facts loaded in the database. The logic model for deductive databases is closely related to the relational model and, in particular, with the domain relational calculus. Their query languages are related with the Prolog language and, mainly, with Datalog, a Prolog subset without complex terms (in order to avoid infinite structures).

The relational algebra has been shown to be inefficient for expressing database queries. A main defect is the lack of recursion, which does not allow to express recursive definitions as the transitive closure of a graph.

Origins of deductive databases can be found in automatic theorem proving and, later, in logic programming. Minker [Mink87] suggests that Green and Raphael [GR68] were the pioneers in discovering the relation between theorem proving and deduction in databases. They developed several question–answer systems using a version of the Robinson resolution principle [Robi65], showing that deduction can be systematically performed in a database environment. Other pioneer systems were MRPPS [MN82], DEDUCE–2 [Chan78] and DADM [KT81].

1.2 Referring to DES

Please use the following BiBTeX entry for referring to this system:

```
@techreport{des-user-manual-tr,
  author = {F. S\'aenz-P\'erez},
```

```
  title = {Datalog Educational System V1.1. User's Manual},
  institution = {Faculty of Computer Science, UCM},
  year = 2004,
  number = {139-04},
  note = {Available from http://www.fdi.ucm.es/profesor/fernan/DES/}
 }
}
```

1.3 Release Notes

Version 1.1 of DES adds to previous version (1.0):

- Full recursion
- Memoization techniques
- Gathering of undefined facts under non stratified programs (incomplete algorithm)
- Several new commands:
    - /listing name/arity. Lists Datalog rules matching the pattern
    - /retractall head. Deletes all Datalog rules matching head
    - /list_et. Lists contents of the extension table
    - /list_et name/arity. Lists contents of the extension table matching the pattern
    - /clear_et. Clears the extension table
    - /builtins. lists builtin operators
    - /cd path. Sets the current directory
    - /cd. Sets the current directory to the directory where DES was started from
    - /pwd. Displays the current directory
    - /ls. Displays the contents of the current directory
    - /ls path. Displays the contents of the given absolute or relative directory
    - [filenames]. Consults a list of Datalog files abolishing previous rules
    - [+filenames]. Consults a list of Datalog files keeping previous rules
    - /shell command. Submits a command to the operating system shell
- Cosmetic changes:
    - Commands start with a slash
    - Command arguments are no longer enclosed in brackets
    - Both commands and queries may end with a dot
- Fixed bugs:
    - Primitives fail adequately when they should do it, instead of exiting from the interpreter

2. Installation

2.1 Downloading DES

You can download the system from the DES web page via the URL: **http://www.fdi.ucm.es/profesor/fernan/DES/**

There, you can find a source distribution for several Prolog interpreters and operating systems, and executable distributions for Windows and Unix.

2.1.1 Source Distribution

Under the source distribution, there are several versions depending on the Prolog interpreter you select to run DES: Ciao Prolog [BCC97], GNU Prolog [Diaz], Sicstus Prolog [Sicstus], and SWI Prolog [Wiele]. However, with minor changes to a small selected piece of code (found in the file **`des1.pl`**), it is likely to run on any other Prolog system and operating system it is installed, since the core (found in the file **`des.pl`**) was implemented following standard Prolog (See section 6.3 for porting to unsupported systems). We have tested DES under several Prolog systems (Ciao Prolog 1.8#2, GNU Prolog 1.2.16, Sicstus Prolog 3.11.0, and SWI–Prolog 5.2.10), and several operating systems (MS Windows 98 and later, Solaris, and Linux).

The source distribution comes in a single archive file containing the following:

- **des.pl**. Contains the core of DES
- **des1.pl**. Contains particular code for the selected Prolog system
- **systems/{ciao,gnu,sicstus,swi}**. Contains the same two previous files for all of the supported Prolog systems (these directories can be erased if desired, they are included only for reference)
- **doc/manual.pdf**. This manual
- **examples/*.dl** Example files which will be discussed in section 3.4
- **license/license** A verbatim copy of the GNU Public License for this distribution

2.1.2 Executable Distribution

2.1.2.1 Windows

From the same above URL you can download a Windows executable distribution in a single archive file containing the following:

- **des.exe**. Console executable file
- **deswin.exe**. Windows executable file
- **des.pl**. Contains the core of DES
- **des1.pl**. Contains Sicstus dependent code
- **systems/{ciao,gnu,sicstus,swi}**. Contains the same two previous files for all of the supported Prolog systems (these directories can be erased if desired, they are included only for reference)

- **main.sav**. Saved state of DES
- ***.dll**. DLL libraries for the runtime system
- **doc/manual.pdf**. This manual
- **examples/*.dl** Example files which will be discussed in section 3.4
- **license/license** A verbatim copy of the GNU Public License for this distribution
- **sp311/*.dl** Directory containing Prolog libraries

2.1.2.2 Unix

From the same above URL you can download a Unix executable distribution in a single archive file containing the following:

- **des**. Console executable file. It may require to set the execution permission
- **des.pl**. Contains the core of DES
- **des1.pl**. Contains Sicstus dependent code
- **systems/{ciao,gnu,sicstus,swi}**. Contains the same two previous files for all of the supported Prolog systems (these directories can be erased if desired, they are included only for reference)
- **des.sav**. Saved state of DES
- **doc/manual.pdf**. This manual
- **examples/*.dl** Example files which will be discussed in section 3.4
- **license/license** A verbatim copy of the GNU Public License for this distribution

2.2 Installing DES

Unpack the distribution archive file into the directory you want to install DES, which will be referred to as the distribution directory from now on. This allows you to run the system, whether you have a Prolog interpreter or not (in this latter case, you have to run the system either on MS Windows or SunOS).

Although there is no need for further setup and you can go directly to section 3.1, you also can configure the way the system starts for commodity. In this way, you can follow two routes depending on the operating system.

2.2.1 MS Windows

2.2.1.1 Executable Distribution

Simply create a shortcut in the desktop for executing the executable of your choice: des.exe or deswin.exe. The former is a console-based executable, whereas the latter is a windows-based executable. Both have been generated under Sicstus Prolog, so that all Sicstus notes in the rest of this document also apply to these executables.

2.2.1.2 Source Distribution

Perform the following steps:

1. Create a shortcut in the desktop for running the Prolog interpreter.
2. Modify the start directory in the Properties dialog box of the shortcut to the installation directory for DES. This allows the system to consult the needed files at startup.
3. Append the following options to the Prolog executable path, depending on the Prolog interpreter you use:
   (a) Ciao Prolog: **`-l ciaorc`**
   (b) GNU Prolog: **`--entry-goal ['des.pl']`**
   (c) Sicstus Prolog: **`-l des.pl`**
   (d) SWI Prolog: **`-g "[des]"`**

Another alternative is to write a batch file similar to the script file described in the next section.

2.2.2 Unix/Linux

2.2.2.1 Executable Distribution

You can create a script or an alias for executing the file **`des`** at the distribution root. This executable has been generated under Sicstus Prolog, so that all Sicstus notes in the rest of this document also apply to these executables.

2.2.2.2 Source Distribution

You can write a script for starting DES according to the selected Prolog interpreter, as follows:

(a) Ciao Prolog:
`$CIAO –l ciaorc`

Provided that **`$CIAO`** is the variable which holds the absolute filename of the Ciao Prolog executable.

(b) GNU Prolog:
`$GNU --entry-goal ['des.pl']`

Provided that **`$GNU`** is the variable which holds the absolute filename of the GNU Prolog executable.

(c) Sicstus Prolog:
`$SICSTUS –l des.pl`

Provided that **`$SICSTUS`** is the variable which holds the absolute filename of the Sicstus Prolog executable.

(d) SWI Prolog:
`$SWI -g "[des]"`

Provided that **`$SWI`** is the variable which holds the absolute filename of the SWI Prolog executable.

3. Using DES

Since DES has been written with Prolog, we have adopted almost all the Prolog syntax conventions for writing Datalog programs (the reader is assumed to have basic knowledge about Prolog). Commands are somewhat different for Prolog programmers as they are accustomed to (See section 4). Also, exceptions are noted when necessary.

3.1 Starting DES

Besides the methods described in the previous section, you can start DES from a Prolog interpreter, firstly changing to the distribution directory, and then with:

```
?- [des].
```

Followed by:

```
?- start.
```

Whichever method you use to start DES (a script, batch file, or shortcut), you get the following:

```
*********************************************************
*                                                       *
* DES: Datalog Educational System v.1.1                 *
*                                                       *
* - Stratified Negation                                 *
* - Full recursion                                      *
* - Noncompound terms                                   *
*                                                       *
* Type "/help" for commands                             *
* Type "des." if you get out of DES                     *
*                                                       *
*                              Fernando Sáenz (c) 2004 *
*                                               SIP UCM *
*            Please send comments, questions, etc. to: *
*                                      fernan@sip.ucm.es *
*                                 Visit the Web site at: *
*            http://www.fdi.ucm.es/profesor/fernan/DES/ *
*********************************************************

DES>
```

This last line (**DES>**) is the DES prompt, which allows you to write commands or Datalog queries. If an error leads to an exit from DES and you have started from a Prolog interpreter, then you can write **des.** at the Prolog prompt to continue.

3.2 Writing and Running Datalog Programs

The common way of using the system is to write Datalog program files (with default extension **.dl**) and consulting them before submitting queries. Another alternative is to assert program rules from the command prompt. Following the first alternative, you write the program in a text file, and then you use the following command in order to consult the Datalog program[2]:

DES> /consult *Filename*

[2] See section 4 for more details about commands.

Where ***FileName*** is the name of the file, as **family.dl** (the default extension la extensión **.dl** can be omitted). If the file is located in the system directory, you can consult the file with:

```
DES> /consult family.dl
```

or simply:

```
DES> /consult family
```

Otherwise, when the file is located at another path, you can firstly change to the new path using the command **/cd *Path***, where ***Path*** is the new directory (relative or absolute). Assuming that we are in the system directory and we have installed the distribution **at c:\des1.1**, we can do the following:

```
DES> /cd examples

Current directory is:
  c:/des1.1/examples/
```

Alternatively, you can directly consult the file with relative or absolute paths:

```
DES> /consult examples/family.dl
DES> /consult C:/des1.1/examples/family.dl
```

Assume now that we have the program file **a.dl** with the following contents (also found in the examples directory):

```
a(a1).
a(a2).
a(a3).
```

From the Datalog prompt, the following commands and queries may be submitted:

```
DES> /consult a

Consulting a...
a(a1)
a(a2)
a(a3)


DES> /listing

a(a1)
a(a2)
a(a3)

yes

DES> /assert a(a4)

yes
```

```
DES> /listing

a(a1)
a(a2)
a(a3)
a(a4)

yes

DES> a(a3)

{
  a(a3)
}
yes

DES> a(a5).

{
}
no

DES> a(X).

{
  a(a1),
  a(a2),
  a(a3),
  a(a4)
}
yes

DES> /prolog a(X)

a(a1)

? (type ; for more solutions, <Intro> to continue) ;

a(a2)

? (type ; for more solutions, <Intro> to continue) ;

a(a3)

? (type ; for more solutions, <Intro> to continue) ;

a(a4)

? (type ; for more solutions, <Intro> to continue) ;

no
```

This last command allows to note the basic difference between Datalog and Prolog execution. The former gives the whole meaning of the relation **a** with the query **a(X)**, whereas the latter search for solutions that satisfy the goal **a(X)**. A meaning of a relation is the set of facts inferred both extensionally and intensionally from the program.

3.2.1 Syntax

DES syntax comes mainly from Prolog:

- Numbers. Integers and decimal numbers are allowed. A number is decimal whenever the number contains a dot (**.**) between two digits. No provision is made for floating-point numbers up to now. The range depends on the Prolog platform being used. Negative numbers are identified by a preceding minus (**-**), as usual.

  Examples of numbers are **1**, **1.1**, and **-1.0**.

  Note that **-1.**, **+1**, and **.1** are not valid numbers.

- Atoms. Atoms are identifiers used to build the Herbrand universe. An atom is a sequence of characters written in any of the following forms:
  - Any sequence of alphanumeric characters (including _), starting with a lowercase letter
  - Any sequence of alphanumeric characters delimited by single quotes.

  Examples of atoms are **foo**, **foo_foo**, and **'X'**.

- Constants. A constant is either a number (integer or decimal) or an atom.
- Variables. Variables are written with alphanumeric characters, and alternatively start with uppercase or with an underscore (_).

  Examples of variables are: **X**, **_X**, **_variable**.

- Terms. Terms can be:
  - Noncompound. Variables or constants.
  - Compound. As in Prolog, they are composed of a functor followed by a comma-separated list of arguments enclosed between brackets. A functor obeys the same syntax rules as atoms. An argument can be a noncompound term.

  Some built-in operators are written infix, as relational operators, equality and disequality (See section 5).

  Examples of compound terms are: **r(p)** , **p(X,Y)**, and **X > Y**.

- Goals. Goals obey the same syntax rules as compound terms. In addition, a goal can take the form **not(*Term*)**, where ***Term*** is a compound term. Goals can appear in rule bodies and queries.

3.2.2 DES Queries

A query is the name of a relation with as many arguments as the arity of the relation, following the same syntax as compound terms. These arguments can be variables or atoms. Complex data structures are not allowed. Note that you cannot write conjunctive queries, as **a(X), b(X)**. This does not imply loss of generality because you can write a rule **r(X) :- a(X), b(X)** and submit the query **r(X)**. Built-in operators (listed in section 5) can be used in queries whenever their arguments are ground, i.e., they are constants.

You can type in queries (as well as the commands described in section 4) at the DES system prompt. The answer to a query is the set of facts matching the query. A query with variables for all the arguments of the queried relation gives the whole set of facts defining the relations, as the query **a(X)** in the previous example. If a query contains an atom in an argument position, it means that the query processing will select the facts from the meaning of the relation such that the argument position matches with the atom (i.e., analogous to a select relational operation). This is the case of the query **a(a3)** in the example at the beginning of section 3.2.

3.2.3 DES Rules

DES rules are similar to Prolog rules with the same restrictions found in queries. DES rules have the form **head :- body**, or simply **head**. Both end with a dot. A DES head follows the same syntax as a compound term. A DES body contains a comma-separated sequence of goals which may contain predefined operators as listed in section 5. Goals can be positive or negative. A negated goal is expressed as **not(goal)**.

3.3 Getting Help

You can get useful information with the following commands:

- **/help.** Shows the list of available commands, which are explained in section 4.
- **/builtins.** Shows the list of predefined operators, which are explained in section 5.

Also, visit the URL for last information:
http://www.fdi.ucm.es/profesor/fernan/DES/

3.4 Examples

The DES distribution contains the directory **examples** which shows several features of the system.

3.4.1 Relational Operations (file relop.dl)

This (classic) program is intended to show how to mimic the basic relational operations with Datalog rules. It contains three relations (**a**, **b**, and **c**), which are used as arguments of relational operations.

```
% Relations
a(a1).
a(a2).
a(a3).

b(b1).
b(b2).
b(a1).
```

```
c(a1,b2).
c(a1,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| b
join(X) :- a(X), b(X).

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

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

Once the program is consulted, you can query it by, for example:

```
DES> projection(X)

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

The result of a query is the meaning of the view, i.e., the fact set for the query derived from the program whether intensionally or extensionally. In the above example, **`projection(X)`** corresponds to the projection of the first argument of relation **`c`**.

3.4.2 Paths in a Graph (file `paths.dl`)

This program[3] introduces the use of recursion in DES by defining the graph in Figure 1 and the set of tuples <origin, destination> such that there is a path from origin to destination.

[3] Adapted from [TS86].

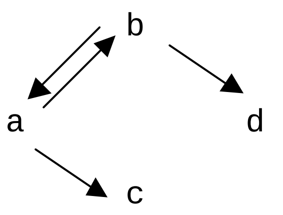

Figure 1. Paths in a Graph

```
% 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).
```

The query **path(X,Y)** yields the following answer:

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

3.4.3 Family Tree (file family.dl)

This (yet another classic) program defines the family tree shown in Figure 2, the set of tuples <parent, child> such that parent is a parent of child (the relation parent), the set of tuples <ancestor, descendant> such that ancestor is an ancestor of descendant (the relation ancestor), the set of tuples <father,child> such father is the father of child, and the set of tuples <mother,child> such mother is the mother of child.

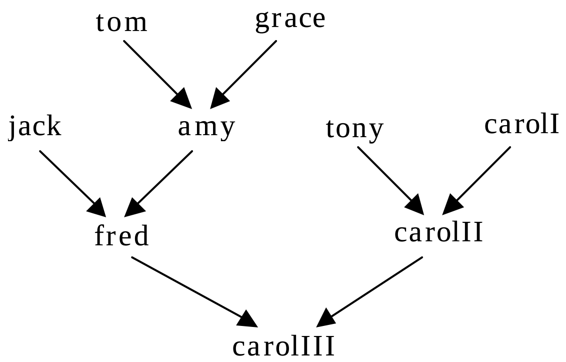

Figure 2. Family Tree

```
father(tom,amy).
father(jack,fred).
father(tony,carolII).
father(fred,carolIII).
mother(graceI,amy).
mother(amy,fred).
mother(carolI,carolII).
mother(carolII,carolIII).

parent(X,Y) :- father(X,Y).
parent(X,Y) :- mother(X,Y).

ancestor(X,Y) :- parent(X,Y).
ancestor(X,Y) :- parent(X,Z), ancestor(Z,Y).
```

The query **ancestor(tom,X)** yields the following answer (that is, it computes the set of descendants of **tom**):

```
{
  ancestor(tom,amy),
  ancestor(tom,carolIII),
  ancestor(tom,fred)
}
```

3.4.4 Basic Recursion Problem (file recursion.dl)

This example is intended to show that queries involving recursive predicates do terminate thanks to DES fixpoint semantics, by contrast with Prolog's usual SLD resolution.

```
p(0).
p(X) :- p(X).
p(1).
```

The query **p(X)** returns the inferred facts from the program irrespective of the apparent infinite recursion in the second rule. (Note that the Prolog goal **p(1)** does not terminate. You can check it out with /**prolog p(1)**).

3.4.5 Transitive Closure (file tranclosure.dl)

With this example, we show a possible use of mutual recursion by means of a program that defines the transitive closure of the relations p and q.

```
p(a,b).
p(c,d).
q(b,c).
q(d,e).
pqs(X,Y) :- p(X,Y).
pqs(X,Y) :- q(X,Y).
pqs(X,Y) :- pqs(X,Z),p(Z,Y).
pqs(X,Y) :- pqs(X,Z),q(Z,Y).
```

The query **pqs(X,Y)** returns the whole set of inferred facts that model the transitive closure.

3.4.6 Mutual Recursion (file `mutrecursion.dl`)

The following program shows a basic example about mutual recursion:

```
p(a).
p(b).
q(c).
q(d).
p(X) :- q(X).
q(X) :- p(X).
```

Submitting the goal **p(X)**, we get:

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

which is the same set of values for arguments for the query **q(X)**. The file **mrtc.dl** is a combination of this example and that of the previous section.

3.4.7 Stratified Negation (file `negation.dl`)

DES ensures that negative information can be gathered from a program with negated goals provided that a restricted form of negation is used: stratified negation. This broadly means that negation is not involved in a recursive computation path, although it can use recursive rules. The following program[4] illustrates this point:

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

The query **a** succeeds with the meaning **{a}**. Observe also that **not(a)** does not succeed, i.e., its meaning is the empty set.

3.4.8 Paradoxes (file `russell.dl`)

When negation is used, we can find paradoxes, such as the Russell's paradox (the barber in a town shaves every person who does not shave himself) shown in the next example (please note that this example is not stratified and, in general, we cannot ensure completeness for non stratified negated programs):

```
shaves(barber,M) :- man(M), not(shaves(M,M)).
man(barber).
man(mayor).
```

[4] Adapted from [RSSWF97].

If we submit the query **shaves(X,Y)**, we get the positive facts as well as a set of undefined inferred information (in our example, whether the barber shaves himself), as follows:

```
DES> shaves(X,Y).
{
  shaves(barber,mayor)
}
yes

Undefined:
{
  shaves(barber,barber)
}
```

If we look at the extension table contents by submitting the **et** command, we get:

```
man(barber).
man(mayor).
not(shaves(mayor,mayor)).
shaves(barber,mayor).
```

We can see that, in particular, we have proved additional negative information (the mayor does not shaves himself) and that no information is given for the undefined facts. The current implementation uses an incomplete algorithm for finding such undefined facts. We can see this incompleteness by adding the following rule:

```
shaved(M) :- shaves(barber,M).
```

The query **shaved(M)** returns:

```
{
  shaved(barber),
  shaved(mayor)
}
yes

Undefined:
{
  shaves(barber,barber)
}
```

That is, the system is unable to prove that **shaved(barber)** is undefined (it proves that the fact is positive but it is unable to prove that the fact is also negative). Future versions may overcome this limitation by allowing more flexible forms of stratification.

The basic paradox **p:-not(p)** can be found in the file paradox.dl, whose model is undefined as you can test with the query **p**.

3.4.9 Farmer-Wolf-Goat-Cabbage Puzzle (file `puzzle.dl`)

This example[5] shows the classic Farmer–Wolf–Goat–Cabbage puzzle (also Missionaries and Cannibals as another rewritten form). The farmer, wolf, goat, and cabbage are all on the north shore of a river and the problem is to transfer them to the south shore. The farmer has a boat which he can row taking at most one passenger at a time. The goat cannot be left with the wolf unless the farmer is present. The cabbage, which counts as a passenger, cannot be left with the goat unless the farmer is present. The following program models the solution to this puzzle. The relation **`state/4`** defines the valid states under the specification (i.e., those situations in which there is no danger for any of the characters in our story; a state in which the goat is left alone with the cabbage may result in an eaten cabbage) and imposes that there is a previous valid state from which we depart from. The arguments of this relation are intended to represent (from left to right) the position (north –**`n`**– or south –**`s`**– shore) of the farmer, wolf, goat, and cabbage. We use the relation **`safe/4`** to verify that a given configuration of positions is valid. The relation **`opp/2`** simply states that north is the opposite shore of south and viceversa.

```
% Initial state
state(n,n,n,n).
% Farmer takes Wolf
state(X,X,U,V) :-
  safe(X,X,U,V),
  opp(X,X1),
  state(X1,X1,U,V).
% Farmer takes Goat
state(X,Y,X,V) :-
  safe(X,Y,X,V),
  opp(X,X1),
  state(X1,Y,X1,V).
% Farmer takes Cabbage
state(X,Y,U,X) :-
  safe(X,Y,U,X),
  opp(X,X1),
  state(X1,Y,U,X1).
% Farmer goes by himself
state(X,Y,U,V) :-
  safe(X,Y,U,V),
  opp(X,X1),
  state(X1,Y,U,V).

% Opposite shores (n/s)
opp(n,s).
opp(s,n).

% Farmer is with Goat
safe(X,Y,X,V).
% Farmer is not with Goat
safe(X,X,X1,X) :- opp(X,X1).
```

[5] Adapted from [Wagner87].

If we submit the query **state(s,s,s,s)**, we get the expected result:

```
{
  state(s,s,s,s)
}
yes
```

That is, the system has proved that there is a serial of transfers between shores which finally end with the asked configuration (this problem is not modeled to show this serial, although it could be). If we ask for the extension table contents regarding the relation **state/4** (with the command **/list_et state/4**), we get:

```
{
  state(n,n,n,n),
  state(n,n,n,s),
  state(n,n,s,n),
  state(n,s,n,n),
  state(n,s,n,s),
  state(s,n,s,n),
  state(s,n,s,s),
  state(s,s,n,s),
  state(s,s,s,n),
  state(s,s,s,s)
}
```

This is the complete set of valid states which includes all of the valid paths from **state(n,n,n,n)** to **state(s,s,s,s)**. However, the order of states to reach the latter is not given, but we can find it by observing this relation, i.e.:

```
state(n,n,n,n) ® Farmer takes Goat to south shore ®
state(s,n,s,n) ® Farmer returns to north shore ®
state(n,n,s,n) ® Farmer takes Wolf to south shore ®
state(s,s,s,n) ® Farmer takes Goat to north shore ®
state(n,s,n,n) ® Farmer takes Cabbage to south shore ®
state(s,s,n,s) ® Farmer returns to north shore ®
state(n,s,n,s) ® Farmer takes Goat to south shore ®
state(s,s,s,s)   Final safe state
```

Observe that there is two states in the relation **state/4** which do not form part of the previous path:

```
state(s,n,s,s)
state(n,n,n,s)
```

These states come from another possible path[6]:

```
state(n,n,n,n) ® Farmer takes Goat to south shore ®
state(s,n,s,n) ® Farmer returns to north shore ®
state(n,n,s,n) ® Farmer takes Cabbage to south shore ®
state(s,n,s,s) ® Farmer takes Goat to north shore ®
state(n,n,n,s) ® Farmer takes Wolf to south shore ®
state(s,s,s,n) ® Farmer takes Goat to north shore ®
state(s,s,n,s) ® Farmer returns to north shore ®
state(n,s,n,s) ® Farmer takes Goat to south shore ®
state(s,s,s,s)   Final safe state
```

[6] Remember that the system returns *all* of the possible solutions.

Now, let's turn our attention to the query **state(X,Y,U,V)**. If we empty the extension table with **/clear_et** or we submit this query from an empty extension table (as when we consult a program), we may expect to get all the possible states as before, but we only get:

```
{
  state(n,n,n,n)
}
yes
```

However, this is a reasonable answer given the above program. Note that, by contrast with previous examples, we have a non ground fact:

```
% Farmer is with Goat
safe(X,Y,X,V).
```

What is the meaning of this rule? The relation **safe/4** is true whenever the first and third arguments are the same, whatever the arguments are. This means that we can find an infinite ground set representing the meaning of **safe/4**. The answer set for this relation is represented with ground and non ground facts, as follows:

```
{
  safe(_8432,_8431,_8432,_8433),
  safe(n,n,s,n),
  safe(s,s,n,s)
}
```

The meaning of this relation is not completely defined with ground facts, which are needed to also complete the meaning of **state/4**. If we want to get a finite meaning for relations we are ought to write programs with only ground facts. Therefore, it turns out to be reasonable for this example to restrict the relation **safe/4** to only the possible values its arguments can take; in other words, to define finite domains (types) for them. So, we rewrite this relation as:

```
% Farmer is with Goat
safe(X,Y,X,V) :- shore(X), shore(Y), shore(V).
% Farmer is not with Goat
safe(X,X,X1,X) :- opp(X,X1).
```

Where the new relation **shore/1** is intended to represent the two possible values for an argument of type **shore**.

```
% Possible shores (n/s)
shore(n).
shore(s).
```

In order to derive the complete intended meaning for a query, we have to rewrite the entire program such that all of the relation arguments have finite domains. In our example, each rule for state in the program have to be rewritten. For instance:

```
% Farmer takes Wolf
state(X,X,U,V) :-
  shore(U),shore(V),
  safe(X,X,U,V),
  opp(X,X1),
```

```
  state(X1,X1,U,V).
```

We have added domain information to the two arguments we do not know their domains. The first two arguments have known domains because the goal **opp(X,X1)**. (A systematic straightforward approach is to define domains for all of the relation arguments, though less efficient.)

Back to the meaning of **safe/4**, now we get, as expected:

```
{
  safe(n,n,n,n),
  safe(n,n,n,s),
  safe(n,n,s,n),
  safe(n,s,n,n),
  safe(n,s,n,s),
  safe(s,n,s,n),
  safe(s,n,s,s),
  safe(s,s,n,s),
  safe(s,s,s,n),
  safe(s,s,s,s)
}
```

That is, we get the intended intensional meaning in the former answer set as extensional information.

The file **typedpuzzle.dl** contains the rewritten program which yields the intended complete meaning to the query **state(X,Y,U,V)**.

4. Commands

The input at the prompt (i.e., commands or queries) must be written in a line (i.e., without carriage returns, although it can be broken by the DES console due to space limitations) and can end with an optional dot.

Commands are issued by preceding the command with a slash (/) at the DES command prompt. An argument for a command is not enclosed between brackets, it simply appears separated by one or more blanks. Ending dots are considered as part of the argument wherever it can expected. For instance, **/cd ..** behaves as **/cd ...** (this command changes the working directory to the parent directory). In this last case, the final dot is not considered as part of the argument. The command **/ls .** shows the contents of the working directory, whereas **/ls ..** shows the contents of the parent directory (which behaves as **/ls ...**). Filenames and directories can be specified with relative or absolute names. There is no need of enclosing such names between separators. For instance, file or directory names can contain blanks (for Windows users) and you do not need to use double quotes.

When consulting Datalog files, filename resolution works as follows:

- If the given filename ends with **.dl**, DES tries to load the file with this (absolute or relative) filename.
- If the given filename does not end with **.dl**, DES firstly tries to load a file with **.dl** appended to the end of the filename. If such a file is not found, it tries to load the file with the given filename.

In command arguments, when applicable, you can use relative or absolute pathnames. In general, you can use a slash (**/**) as a directory delimiter, but sometimes (depending on the platform) you can also use the backslash (****).

See section 0 for information about DES queries.

4.1 Rule Database Commands

- **/consult *FileName***
  Loads the Datalog program found in the file ***Filename***, discarding rules already loaded. The extension table is emptied. The default extension **.dl** for Datalog programs can be omitted.
  Examples:
  Assuming we are on the distribution directory, we can write:
  DES> /consult examples/mutrecursion
  which behaves the same as the following:
  DES> /consult examples/mutrecursion.dl
  DES> /consult ./examples/mutrecursion
  DES> /consult c:/des1.1/examples/mutrecursion.dl
  This last command assumes that the distribution directory is **c:/des1.1**.
  Synonyms: **/c**.
- **/[*FileNames*]**
  Loads the Datalog programs found in the list **[*Filenames*]**, discarding rules already loaded. The extension table is emptied. Arguments in the list are comma–separated.
  Examples:
  Assuming we are on the examples distribution directory, we can write:
  DES> /[mutrecursion,family]
  See also **/consult *Filename***.
- **/reconsult *FileName***
  Loads a Datalog program found in the file ***Filename***, keeping rules already loaded. The extension table is emptied.
  See also **/consult *Filename***.
  Synonyms: **/r**.
- **/[+*FileNames*]**
  Loads the Datalog programs found in the comma–sepparated list **[Filenames]**, keeping rules already loaded. The extension table is emptied.
  See also **/[*Filenames*]**.
- **/assert *Head:-Body***
  Adds a Datalog rule. Rule order is irrelevant for Datalog computation. The extension table is emptied.
- **/retract *Head:-Body***
  Deletes a Datalog rule. The extension table is emptied.
- **/retractall *Head***
  Deletes all Datalog rules whose head unifies with ***Head***. The extension table is emptied.
- **/abolish**
  Deletes all the loaded rules. The extension table is emptied.
- **/listing**
  Lists loaded rules.
- **/listing *Name/Arity***

Lists loaded rules matching the pattern ***Name/Arity***.
See also **/listing**.

4.2 Extension Table Commands

- **/list_et**
  Lists the contents of the extension table in alphabetical order. First, answers are displayed, then calls.
- **/list_et *Name/Arity***
  Lists the contents of the extension table matching the pattern ***Name/Arity***.
  See also **/list_et**.
- **/clear_et**
  Deletes the contents of the extension table.

4.3 File System Commands

- **/cd *Path***
  Sets the current directory to ***Path***
- **/cd**
  Sets the current directory to the directory where DES was started from
- **/pwd**
  Displays the absolute filename for the current directory.
- **/ls**
  Displays the contents of the current directory in alphabetical order. First, files are displayed, then directories.
- **/ls *Path***
  Displays the contents of the given path in alphabetical order. It behaves as **/ls**.

4.4 Help Commands

- **/help**
  Shows the help.
  Synonyms: **/h**.
- **/builtins**
  Lists predefined operators.

4.5 Miscellanea

- **/prolog *Goal***
  Triggers Prolog's SLD resolution for the goal ***Goal***.
- **/halt**
  Quits the system.
  Synonyms: **/q**, **/quit**, **/e**, **/exit**.
- **/shell *Command***
  Submits ***Command*** to the operating system shell.
  Notes for platform specific issues:
  - Windows users:
    command.exe is the shell for Windows 98, whereas **cmd.exe** is for Windows NT/2000/2003.
  - Ciao users:
    The environment variable **SHELL** must be set to the required shell.
  - Sicstus users:

Under Windows, if the environment variable **SHELL** is defined, it is expected to name a Unix like shell, which will be invoked with the option **-c Command**. If SHELL is not defined, the shell named by **COMSPEC** will be invoked with the option **/C Command**.

- Windows and Unix executable users:
  The same note for Sicstus is applied.

Synonyms: **/s**.

5. Operators

All operators are infix but negation. Also, they are not cached with the memoization technique but negation.

Some infix operators need ground arguments since they are not constraints, but test predicates. This means that the declarative reading of rules is not full. For instance, given the following rule:

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

the program file **relop.dl**, and the query **less(X,Y)**, we get no solutions, whereas if we rewrite the above rule as:

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

and submit the same query, we get:

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

Therefore, we do not ensure sound answers for programs containing primitives with nonground arguments, since, as seen in such cases, goal ordering affects semantics. However, a constraint solver could be used to overcome this limitation.

Next, we list the available operators.

- **=**
  Tests syntactic equality between noncompound terms (variables, atoms, or numbers). It also performs unification when variables are involved.
- **\=**
  Tests syntactic disequality between noncompound terms (variables, atoms, or numbers). Its declarative reading is sound whenever its arguments are ground; otherwise, problems as stated at the beginning of this section may arise. It always fails if at least one of its arguments is a variable.
- **>**
  Tests whether its left argument is greater than its right argument. Its declarative reading is sound whenever its arguments are ground; otherwise, problems as stated at the beginning of this section may arise. It always fails if at least one of its arguments is a variable. Numbers are compared in terms of their arithmetical value; other terms are compared in standard order.
- **>=**
  Tests whether its left argument is greater or equal than its right argument. Its declarative reading is sound whenever its arguments are ground; otherwise,

problems as stated at the beginning of this section may arise. It always fails if at least one of its arguments is a variable. Numbers are compared in terms of their arithmetical value; other terms are compared in standard order.

- **<**
  Tests whether its left argument is less than its right argument. Its declarative reading is sound whenever its arguments are ground; otherwise, problems as stated at the beginning of this section may arise. It always fails if at least one of its arguments is a variable. Numbers are compared in terms of their arithmetical value; other terms are compared in standard order.
- **=<**
  Tests whether its left argument is less or equal than its right argument. Its declarative reading is sound whenever its arguments are ground; otherwise, problems as stated at the beginning of this section may arise. It always fails if at least one of its arguments is a variable. Numbers are compared in terms of their arithmetical value; other terms are compared in standard order.
- **not(*Relation*)**
  Stratified negation. It stands for the complement of the relation ***Relation*** under the meaning of the program.

6. Notes about the Implementation of DES

DES is implemented with the seminar ideas found in [Wagner87, TS86], that deal with termination issues of Prolog programs. These ideas have been used in the deductive database community. Our implementation uses extension tables for achieving a top–down driven bottom–up approach. In its current form, it can be seen as an extension of the work in [Wagner87] in the sense that we deal with negation and undefined (although incomplete) information. In addition, the implementation follows a different approach: instead of translating rules, we interpret them (this may prove useful for a straightforward implementation of debugging).

DES does not pretend to be an efficient system but a system capable of showing the nice aspects of the more powerful form of logic we can find in Datalog systems wrt. relational database systems.

6.1 Tabling[7]

DES uses an extension table which stores answers to goals previously computed, as well as their calls. For the ease of the introduction, we assume an answer table and a call table to store answers and calls, respectively. Answers may be positive or negative, that is, if a call to positive goal **p** succeeds, then the fact **p** is added as an answer to the answer table; if a negated goal **not(p)** succeeds, then the fact **not(p)** is added. Calls are also added to the call table whenever they are performed. This allows us to detect that a call has been previously performed and we can use the results in the extension table (if any). The algorithm which implements this idea can be sketched as follows:

[7] For a complementary understanding of this section, the reader is advised to read [Wagner87].

First, test whether there is a previous call that subsumes[8] the current call. There are two possibilities: 1) there is such a previous call: then, use the result in the answer table if any. It is possible that there is no such a result (for instance, when computing the goal p in the program **`p :- p`**) and we cannot derive any information. 2) otherwise, process the new call knowing that there is no call or answer to this call in the extension table. So, firstly store the current call and then, solve the goal with the program rules (recursively applying this algorithm). Once the goal has been solved (if succeeded), store the computed answer if there is no any previous answer subsuming the current one (note that, through recursion, we can deliver new answers for the same call). This so–called memoization process is implemented with the predicate **`memo/1`** in the **`des.pl`** file of the distribution, and will also be referred to as a memo function in the rest of this manual.

Negative facts are produced when a negative goal is proved by means of negation as failure (closed world assumption). In this situation, a goal as **`not(p)`** which succeeds produces the fact **`not(p)`** which is added to the answer table, just the same as proving a positive goal.

Primitive operators are not tabled for efficiency reasons without any limitation for the system.

6.2 Finding Stable Models

The tabling mechanism is insufficient in itself for computing all of the possible answers to a query. The rationale behind this comes from the fact that the computed information is not complete when solving a given goal, because it can use incomplete information from the goals in its defining rules (these goals can be mutually recursive). Therefore, we have to ensure that we produce all the possible information by finding a fixpoint of the memo function. First, the call table is emptied in order to allow the system to try to obtain new answers for a given call, preserving the previous computed answers. Then, the memo function is applied, possibly providing new answers. If the answer table remains the same as before after this last memo function application, we are done. Otherwise, the memo function is reapplied as many times as needed until we find a stable answer table (with no changes in the answer table). The answer table contains the stable model of the query (plus perhaps other stable models for the relations used in the computation of the given query).

The fixpoint is found in finite time because the memo function is monotonic in the sense that we only add new entries each time it is called while keeping the old ones. Repeatedly applying the memo function to the answer table produce a finite answer table since the number of new facts that can be derived from a Datalog program is finite (recall that there are no complex terms such as $s^k(z)$). On the one hand, the number of positive facts which can be inferred are finite because there is a finite number of ground facts which can be used in a given proof, and proofs have finite depth provided that tabling prevents recomputations of older nodes in the proof tree. On the other hand, the number of negative facts which can be inferred is also finite because they are proved using negation as failure.

[8] A term T1 subsumes a term T2 if T1 is "more general" than T2 and both terms are unifiable. Eg: **`p(X,Y)`** subsumes **`p(a,Z)`**, **`p(X,Y)`** subsumes **`p(U,V)`**, **`p(X,Y)`** subsumes **`p(U,U)`**, but **`p(U,U)`** neither subsumes **`p(a,b)`**, nor **`p(X,Y)`**.

(Failures are always finite because they are proved trying to get a success.) Finally, there are facts that cannot be proved to be true or false because of recursion. These cases are detected by the tabling mechanism which prevent infinite recursion such as in **`p :- p`**.

It is also possible that both a positive and a negative fact have been inferred for a given call. Then, an undefined fact replaces the contradictory information. The implementation simply removes the contradictory facts and informs about the undefinedness.

6.3 Porting to Unsupported Systems

DES is implemented with two Prolog files: des.pl, and des1.pl. The former contains the common predicates for all of the platforms (both Prolog interpreters and operating systems), and the latter contains Prolog system specific code, which vary from a system to another. Adapting the predicates found there should not pose problems, provided the Prolog interpreter and operating system features some basic characteristics (mainly about the file system commands). If you plan to port DES to other systems not described here, you will have to modify the system specific Prolog file to suit your system. If so, and if you want to figure as one of the system contributors, please send an e–mail message with the code and reference information to: **`fernan@sip.ucm.es`**, accepting that your contribution will be under the GNU General Public License (See appendix for details.)

6.4 Differences among Platforms

Ciao, SWI, and Sicstus Prolog implementations use a sort which eliminates duplicates whereas GNU Prolog implementation does not.

In its current version, the Ciao system forces to use some directives for using several basic Prolog primitives. This can only be done by writing them in the core file (des.pl) of the system, making it not compatible with other platforms. This is why the core file for Ciao has some preliminary directives not found in the core file shared by the rest of the platforms. Future Ciao versions may overcome this problem.

7. Related Work

There has been a high amount of work around deductive databases [RU95] (its interest delivered many workshops and conferences for this subject) which dealt to several systems. However, to the best of our knowledge, there is no system oriented to introducing deductive databases to students, but we can comment some representative deductive database systems.

The LOLA [ZF97] deductive database system is based on a declarative clause language supporting recursion, complex terms, negation, aggregation, built–in predicates. The deductive engine is based on the paradigm of bottom–up evaluation with relational operations. It is available via a web browser interface.

The LDL project at MCC that lead to the LDL++ prototype [ZAO93], a deductive database system with features as stratified and nonstratified negation, set terms, and aggregates. It can be currently used through Internet using a Java–enabled client.

XSB [RSSWF97] (**http://xsb.sourceforge.net/**) is an extended Prolog system that can be used for deductive database applications. It enjoys a well–founded semantics for rules with negative literals in rule bodies and implements tabling mechanisms. It runs both on Unix/Linux and Windows operating systems.

Coral [RSSS93] is a deductive system with a declarative query language that supports general Horn clauses augmented with complex terms, set–grouping, aggregation, negation, and relations with tuples that contain (universally quantified) variables. It only runs under Unix platforms. There is also a version which allows object–oriented features, called Coral++ [SRSS93].

The NAIL! project delivered a prototype with stratified negation, well–founded negation, and modularity stratified negation. Later, it added the language Glue, which is essentially single logical rules, with SQL statements wrapped in an imperative conventional language [PDR91, DMP93]. The approach of combining two languages is similar to the aforementioned Coral, which uses C++. It does not run on Windows platforms.

Another deductive database following this combination of declarative and imperative languages is Rock&Roll [BPFWD94].

The only commercial oriented deductive database system has been the Smart Data System (SDS) and its declarative query language Declarative Reasoning (DECLARE) [KSSD94], with support for stratified negation and sets.

ADITI 2 [VRK+91] is the current version of a deductive database system which uses the logic/functional programming language Mercury. It does not (and probably will never) run on Windows platforms.

8. Future Enhancements

The following list suggests some points to address in order to enhance DES:

- Aggregated functions (count, sum, avg, ...).
- Complete algorithm for finding undefined information.
- Constraint solver for sound computation of primitives.
- Database integration (relational, object–oriented). We propose a system which can use both SQL and Datalog as query languages of current high–widely used relational systems as Microsoft Access, Oracle, and MySQL. We think that the simple ideas found in the relational model makes it quite adequate for current business applications, but relational systems can be enhanced by allowing SQL to be complemented with the more powerful Datalog database language.
- Debugger.
- Integrated development environment under Windows. We are interested in this platform due to most of our students do feel more comfortable with Windows systems.

9. Acknowledgements

The author whishes to thank the Clip group for providing their free Ciao system, and in particular to F. Bueno for his help in porting DES to the Ciao system. Also thanks to J. Wielemaker and D. Diaz for providing their free Prolog systems.

Appendix A. GNU General Public License

Version 2, June 1991

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this

License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

one line to give the program's name and an idea of what it does.

Copyright (C) *yyyy name of author*

This program is free software; you can redistribute it and/or

modify it under the terms of the GNU General Public License

as published by the Free Software Foundation; either version 2
of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

Gnomovision version 69, Copyright (C) *year name of author*

Gnomovision comes with ABSOLUTELY NO WARRANTY; for details

type `show w'. This is free software, and you are welcome

to redistribute it under certain conditions; type `show c'

for details.

The hypothetical commands `show w' and `show c' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than `show w' and `show c'; they could even be mouse-clicks or menu items--whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright

interest in the program `Gnomovision'

(which makes passes at compilers) written

by James Hacker.

signature of Ty Coon, 1 April 1989

Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License.

Bibliography

[BCC97] 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. **http://www.clip.dia.fi.upm.es**.

[BPFWD94] 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 20^{th} VLDB Conference, 1994.

[Chan78] C.L. Chang, "Deduce 2: Further Investigations of Deduction in Relational Databases", H. Gallaire and J. Minker (eds.), Logic and Databases, Plenum Press, 1978.

[Diaz] D. Diaz, **http://www.gnu.org/software/prolog**.

[DMP93] 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.

[GR68] C.C. Green and B. Raphael, "The Use of Theorem–Proving Techniques in Question–Answering Systems", Proceedings of the 23^{rd} ACM National Conference, Washington D.C., 1968.

[KSSD94] W. Kiessling, H. Schmidt, W. Strauss, and G. Dünzinger, "DECLARE and SDS: Early Efforts to Commercialize Deductive Database Technology", VLDB Journal, 3, pp. 211–243, 1994.

[KT81] C. Kellogg and L. Travis, "Reasoning with Data in a Deductively Augmented Data Management System", H. Gallaire, J. Minker, and J. Nicolas (eds.), Advances in Data Base Theory, Volume 1, Plenum Press, 1981.

[Mink87] J. Minker, "Perspectives in Deductive Databases", Technical Report CS–TR–1799, University of Maryland at College Park, March 1987.

[MN82] J. Minker and J.–M. Nicolas, "On Recursive Axioms in Deductive Databases, Information Systems", 16(4):670–702, 1991.

[PDR91] G. Phipps, M. A. Derr, and K.A. Ross, "Glue–NAIL!: A Deductive Database System". In Proc. of the ACM SIGMOD Conference on Management of Data, pp. 308–317, 1991.

[Robi65] J.A. Robinson, "A Machine–Oriented Logic Based on the Resolution Principle", Journal of the ACM, 12:23–41, 1965.

[RSSS93] 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.

[RSSWF97] 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.

[RU95] R. Ramakrishnan and J.D Ullman, "A Survey of Research on Deductive Database Systems", Journal of Logic Programming, 23(2): 125–149, 1995.

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

[SRSS93] 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.

[TS86] H. Tamaki and T. Sato, "OLD Resolution with Tabulation", Proceedings of ICLP'86, Lecture Notes on Computer Science 225, Springer–Verlag, 1986.

[VRK+91] 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.

[Wagner87] S. Wagner, "Extension Tables: Memo Relations in Logic Programming", IV IEEE Symposium on Logic Programming, 1987.

[Wiele] J. Wielemaker, **http://www.swi-prolog.org**.

[ZAO93] 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 Databases, 1993.

[ZF97] 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.