PROLOG: UNA HERRAMIENTA IDEAL PARA EL DESARROLLO DE PROTOTIPOS?

Gustavo Arango Gaviria
Universidad de los Andes
Bogotá - *Colombia*

PROLOG : Una herramienta ideal para la investigación en la universidad (léase desarrollo de prototipos)
Su ilustración con una aplicación en bases de datos

0. GUIAS DE LECTURA.

El artículo se escribió pretendiendo tener varios tipos de lectores:

- quienes no conocen Prolog pero desean conocer algo de sus posibilidades (deben leer la descripción del sistema sin tratar ahondar en las explicaciones sobre las gramática ni en las cláusulas PROLOG que se presentan para implementar algunas de las características del sistema).
- quienes no conocen de teorías sobre analizadores que deben omitir todas las explicaciones técnicas al respecto.
- las personas familiarizadas con PROLOG, bases de datos y gramáticas a quienes todo debe interesarles.

I. Introducción, supuestos y motivaciones.

Se pretende ilustrar la tesis enunciada en el título a través de la descripción de un sistema de consulta en lenguaje natural para bases de datos definidas según el modelo entidad-relación.

Antes que nada es indispensable definir lo que es la investigación a nivel de informática en un departamento de sistemas donde ofrece - por el momento - exclusivamente un pro\ ma de pregrado.

Los profesores pueden solo dedicar una cuarta parte de su tiempo a proyectos de investigación puesto que gran parte de su trabajo está consumido por la docencia.
En la mayoría de los casos la generación de resultados proviene de los trabajos en tesis de grado que son componentes de proyectos dirigidos por profesores que trabajan en áreas de interés común definidas según sus especialidades.

Supuestos básicos:

a. Por la estructura misma de la universidad como generadora de conocimientos tenemos que la manera más eficaz de concretar los resultados de investigación es por medio de la generación de maquetas o prototipos.
Con esta forma de funcionar se logra :
- mostrar y difundir entre profesionales e industriales el alcance real de los proyectos de investigación que se llevan a cabo.
- no exigir un gasto exagerado de recursos, ni un compromiso largo de la investigación en problemas donde el interés primordial es mostrar la factibilidad de buenas soluciones y no la obtención de productos finales.

Aceptemos entonces que el desarrollo de prototipos es el principal producto de la investigación.

b. Aun si admitimos la posibilidad de generación por la universidad de productos de software; una de las etapas a lograr lo más pronto posible en el ciclo de vida de un programa verdaderamente operacional(ver :comercializable) es justamente un prototipo.

La obtención de un prototipo permite la interacción con el usuario cuyo aporte es fundamental para identificar:

- las fallas y omisiones en la comunicación hombre-máquina
- las indefiniciones en el problema,
- las fallas en el diseño.

c. Dada la gran difusión de micro-computadores, cada vez es mayor el número de sus utilizadores.
El diferente nivel de capacitación de estos frente a las máquinas exige gran flexibilidad de las aplicaciones .

El objetivo que se busca es llegar a exigir lo mínimo del usuario por parte de la aplicación.
Lo ideal es que el lenguage de la aplicación se acerque al máximo al lenguaje del usuario y no lo contrario.
Una de las posibilidades es usar el lenguaje natural o en su defecto manejar la ilusión de comunicar en lenguaje natural utilizando un subconjunto de éste.

El propósito de la aplicación que se describe es el de ilustrar las posibilidades de PROLOG para la generación de prototipos de aplicaciones donde la comunicación con el usuario se hace buscando acercarse al máximo a su lenguaje

II. Descripción del sistema

El sistema presentado es el resultado de dos meses de investigación en el período de descanso en la docencia, su meta era explorar hasta donde se podía llegar en la investigación en

Prolog, creando un sistema de multiples aplicaciones con algunas características (flexibilidad, facilidad de comunicación, verificación de coherencia de datos) que lo hicieran llamativo a usuarios finales (aquellos que no requieren de grandes conocimientos en informática para usar una aplicación)

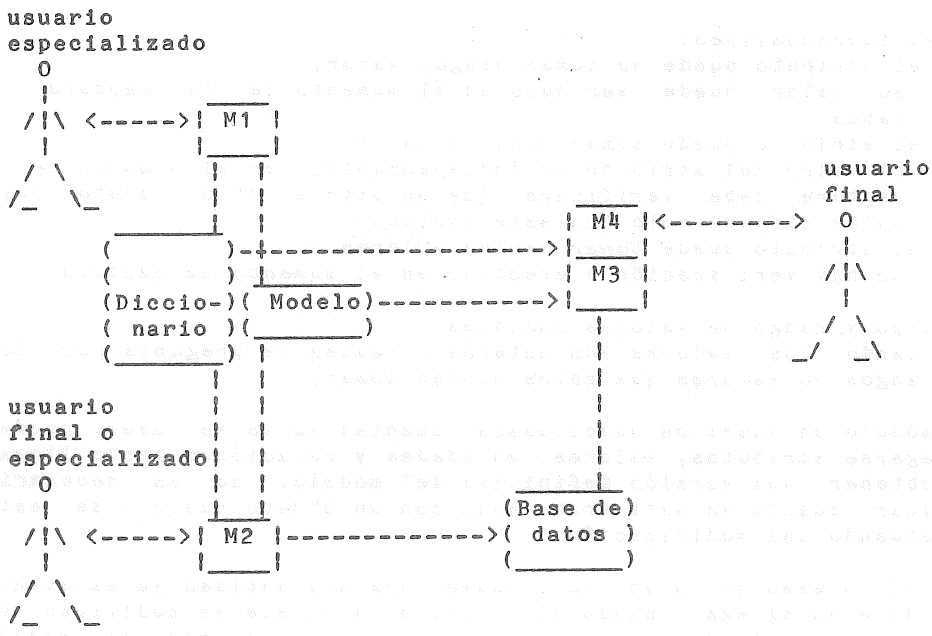
Se trata de un prototipo de un manejador de bases de datos relacionales que puede consultar el usuario final en un subconjunto del lenguaje natural.

Es importante hacer notar que el sistema que se describe necesita de un tiempo de implementación mucho mayor cuando se usa una herramienta clásica (un lenguaje de programación de alto nivel como Pascal o C).

El sistema comprende cuatro módulos:

- Definición del modelo entidad-relación (M1)
- Captura de los datos (M2)
- Sistema de consulta con base en predicados de Prolog (M3)
- Interfaz en lenguaje natural (M4)

DESCRIPCION GRAFICA DEL SISTEMA



LOS MODULOS:

- M1. Modelo entidad-relación
- M2. Captura de los datos
- M3. Sistema de consulta en Prolog
- M4. Interfaz en lenguaje natural

CONVENCIONES:

(ARCHIVO) |PROGRAMA|

A.Definición del modelo entidad-relación.(M1)

El sistema pide al usuario la descripción de cada uno de los objetos de trabajo : entidades, atributos o relaciones.
A medida que se obtienen nuevos nombres de objetos se actualiza un diccionario que permite la posibilidad de uso de sinóminos y pregunta por las formas en plural de cada item del diccionario

En forma interactiva el programa permite obtener:
- las entidades y sus atributos,
- las relaciones, relaciones que intervienen y sus atributos.

Para todo atributo se determina:

- si se trata de la clave o parte de ésta,

- su funcionalidad:
  el atributo puede no tomar ningun valor,
  (su valor puede ser nulo en el momento de la captura de datos)
  el atributo puede tomar solo un valor
  (el valor del atributo es indispensable y en el momento de la captura debe verificarse que no existe otro objeto cuyo valor difiere solo con este atributo)
  el atributo puede tomar varios valores
  (no hay verificación a efectuar en el momento de captura)

- tipo y rango de valores posibles:
  cuando los valores son enteros o reales se pregunta por los rangos de valores que estos pueden tomar.

El módulo en cuestión puede usarse cuantas veces se desee: pueden agregarse atributos, valores, entidades y relaciones hasta llegar a obtener una versión definitiva del modelo, no es necesario indicar cuándo se está trabajando con un objeto nuevo o se está efectuando una modificación.

vg: Al presentar un atributo nuevo para una entidad ya existente éste se agrega; cuando el atributo ya existe se modifican sus características sin que esto cause traumatismos al modelo existente.

La implementación del modelo entidad-relación es inmediata, cada objeto del modelo se representa como un hecho PROLOG.

Por ejemplo la información sobre una entidad se hace:

```
entidad(Código de su nombre, Número de atributos,
        Lista de los códigos de sus atributos llaves)
```

haciendo abstracción de los códigos tendríamos:

```
entidad(estudiante,4,[carnet])
```

B.Captura de datos.(M2)

Se pregunta al usuario cual es el nombre del objeto sobre el que quiere dar información.

Mediante menús se obtiene:

- para entidades:
  los valores debidamente validados (según se explica en el modulo anterior) de sus atributos.

- para relaciones:
  los valores validados de sus atributos,
  los valores validados de los atributos necesarios para caracterizar las entidades que intervienen.

(Se considera que los atributos que conforman la llave de una entidad identifican claramente; cuando no hay llaves es necesaria la información sobre los valores de todos los atributos)

Ejemplo:

si se definieron las entidades:

- estudiante
  .atributos :
    nombre, carnet, teléfono y programa
  .llave:
   carnet

- materia
  .atributos :
   nombre, código, departamento
  .llave:
   código

y la relación:

- vio
  entidades:
   estudiante, materia
  atributos:
   nota, semestre

entonces al hacerse la captura de datos sobre la relación se pregunta por las informaciones correspondientes a:
- carnet [llave de estudiante]
- código [llave de materia]
- nota y semestre (atributos de la relación)

Cuando un valor no es numérico, se codifica y se agrega a una tabla de los valores posibles de un atributo dado.
Así se permite cuando el usuario lo desee investigar sobre los valores posibles que puede tomar un atributo.
Esta opción no fue incorporada en el sistema pero su programación es inmediata usando el predicado setof (descrito en [CLO 81])

Ejemplo:

en la base de datos:
- fruta(limon)
- fruta(pera),
- fruta(uva),

pregunta:
- setof(X,fruta(X),L),

respuesta:
- L = [limon, pera, uva]

C. La consulta en PROLOG.(M3)

Se han considerado por el momento las preguntas de información directa sobre un objeto :

- para entidades:
  - preguntar por algunos o todos sus atributos,
  - con o sin restricciones sobre los valores de otros de sus atributos

ie: estudiantes de nombre Pedro Pérez
(respuesta: todos los atributos de estudiantes de nombre Pedro Pérez)

carnet de los estudiantes de nombre Pedro Pérez
(respuesta: solamente el valor del carnet)

teléfono y carnet de los estudiantes de ingenieria
(respuesta:carnet y teléfono de los matriculados en ingeniería)

- para relaciones:
  - preguntar por algunas de sus entidades o de sus atributos,
  - con o sin restricciones sobre los valores de algunas de sus entidades o de sus atributos

ie: estudiantes que vieron la materia 21117
(respuesta :carnet de los estudiantes que vieron la materia de código 21117 y todos los valores de los atributos de la relacion vio)

nota de las materias que vió el estudiante 8080800
(repuesta : código y nota de las materias que tienen como identificación para la entidad estudiante el carnet 8080800)

D. la interfaz en lenguaje natural.(M4)

Es quizá en este módulo donde se aplican las técnicas más avanzadas y donde se explotan mejor las características más potentes de PROLOG

La parte fundamental es un generador de analizadores sintáctico-sémantico que recibe la grámatica y produce un programa Prolog que será el que haga efectivo el análisis de una serie de tokens (formas normalizadas de las declinaciones-conjugaciones de las palabras en español) y produce el predicado PROLOG que corresponde a la pregunta del usuario.

El proceso de análisis tiene dos partes :

- Una general independiente de las consultas que requiere de un usuario especializado para que defina la grámatica en su lenguaje
(ésto es prácticamente lo único que debería modificarse para responder consultas en otros lenguajes)
Una vez definida la gramática debe utilizarse el módulo que la lee y genera el programa PROLOG que hace el análisis.
Esto debe repetirse cada vez que se cambie la gramática, pero la labor del usuario especializado se limita a escribir adicciones o modificaciones de las reglas de producción que la componen.

- El análisis de una consulta específica consta de :

  - análisis léxico que recibe la frase del usuario y produce la lista de símbolos (tokens) que sirve de entrada al módulo siguiente,

  - análisis sintáctico-semántico que genera el predicado PROLOG que sirve de consulta.

A continuación se describen en detalle las componentes de proceso de análisis; primero la gramática y sus particularidades, posteriormente un ejemplo de lo escrito para la aplicación y finalmente el proceso general de consulta.

1.La gramática.

Está compuesta por reglas de producción cuya forma general es:

no_terminal ---> lista de terminales, de no terminales y/o de cláusulas semánticas

a.No terminales.

Un no_terminal es un término Prolog con variables y átomos y se puede visualizar en el momento del análisis como un predicado PROLOG que se encargará de hacer un análisis parcial de la cadena de entrada.

b.Terminales.

Un terminal puede tomar varias formas:

i- [atomo]

el análisis tiene éxito si se identifica el símbolo analizado en la cadena de entrada con atomo y se pasa al siguiente símbolo

ie: [ser]

ii- [{atomo}]

es opcional que figure atomo como símbolo de entrada en el análisis en cuyo caso se avanza al símbolo siguiente

ie: [{sobre}]

iii- [objeto^^Variable]

objeto : corresponde a uno de los elementos del lenguaje en el modelo entidad-relación (entidad, atributo, relación, ...)

el análisis tiene éxito cuando el símbolo de entrada pertenece a la categoría especificada por medio de objeto, se pasa al token siguiente y se instancia Variable con el valor con que se codificó en el diccionario.

ie: [entidad^^Var]
tiene éxito si el símbolo de entrada es materia,

[atributo^^Var]
tiene éxito si el símbolo de entrada es nota

(el valor de Var corresponde a los códigos asignados respectivamente por el sistema a materia y nota)

iv- [[Pos,predicado]]

- predicado corresponde a un 'functor' PROLOG y a un predicado

- Pos permite ligar la cadena de entrada con el argumento del 'functor' predicado con cuya variable se instanciará, Pos es la posición de la variable dentro del predicado;

el análisis tiene éxito cuando el símbolo analizado o una sublista de la cadena de entrada satisface predicado.

Con esta forma de terminal se logra permitir un look-ahead pues se puede examinar un numero variable de símbolos en la cadena de entrada.

ie:[[5,atributo(estudiante,nombre,_,_,Val)]]
funciona si:
- atributo(estudiante,nombre,x,y,'pedro perez') está en la base de datos
- pedro, perez son los siguientes símbolos a analizar en la cadena de entrada

c.Acciones semánticas

Se trata simplemente de predicados PROLOG definidos por el usuario para obtener resultados adicionales al análisis.
Como estos predicados deben diferenciarse de los no_terminales de la gramática se introduce un nuevo operador para indicar el cambio de interpretación.
Se utiliza el operador @ para introducir los predicados de PROLOG.

ejemplo:

las siguientes reglas permiten analizar y contestar preguntas sobre la hora actual:

```
responda_hora ---> tiempo(Hora,Minuto),
                   @ write('hora:'),write(Hora),
                     write('minuto:'),write(Minuto).

tiempo(Hora,Minuto) ---> [que], [hora] ,[[es]], [[?]]
                         @ time(time(Hora,Minuto,_,_)).
```

el predicado time(X) se responde consultando el reloj del sistema y tiene éxito instanciando X con el 'functor':
time(Valor_hora,Valor_minutos,
Valor_segundos,Valor_centesimas_de_segundo)

2.Ejemplo de parte de la gramática utilizada en la aplicación.

```
/* identificacion del símbolo distinguido */
    disting(sdist,4).

    sdist(Cad,Obj,Lval,Lres) --->
                    [[ sobre ]],
                    det_cad(Cad,Obj,Lval,Lres).

/* producción que identifica una pregunta sobre una entidad */
    det_cad(entidad,Enti,[Latval],[Latr]) --->
            atri(Latr),  /*  no terminal que identifica
                             restricciones sobre valores de atributos */
            [entidad^^Enti],   /* identificación de una entidad */
            busq_atr_val(Enti,Latval).  /*  no terminal que identifica
                                            los atributos sobre los que
                                             se desea informacion */

/* producciones que identifican el valor de un atributo */
    val_clav(Ent,Atri,Val0) --->
            [de], [atributo^^Atri],      /* identificación de un atributo */
            mire_val(Ent,Atri,Val,Val0).
    val_clav(Ent,Atri,Val) ---> mire_val(Ent,Atri,Val).
             /*  identificacion  de un  valor  de  atributo
                 cuando se omite su nombre */

    mire_val(Entidad,Atri,Val0) --->
       [[4, verifique_val_pos(atributo,Entidad,Atri,Val,Val0)]].
     /* identificación y validación del valor del atributo */

/* produciones que identifican una lista de valores*/
    lista_val_clav(Entres,[[Atr|Claval]|Lres]) --->
        [','],
        val_clav(Entres,Atr,Claval),/* identificación  de un valor
                                       de un atributo */
        lista_val_clav(Entres,Lres).

    lista_val_clav(Entres,[[Atr|Claval]]) --->
        [y], val_clav(Entres,Atr,Claval).

    lista_val_clav(Entres,[]) ---> lambda.
```

la cláusula:

```
disting(sdist,4)
```

permite identificar el símbolo distinguido de la gramática y su número de argumentos como 'functor' PROLOG,
esta información será aprovechada por el generador de analizadores para ceder el control al predicado indicado.

3.El análisis de una consulta.

a. El analizador léxico.

Se recibe una frase que se procesa convirtiendola en una lista de tokens. Todo blanco en la frase se considera un separador de símbolos.
Las palabras originales de la frase sufren transformaciones en la lista de tokens:

- hay una lista de palabras cuyo aporte semántico se considera irrelevante; toda palabra que pertenezca a esta lista se elimina de la lista de símbolos.

- se aprovechan las indicaciones dadas por el usuario al definir su modelo para transformar el plural o las formas conjugadas de los verbos y obtener una forma normal

  ie: estudiantes pasa a ser estudiante
  vieron se transforma en vio

b. El análisis semántico.

La lista de símbolos producida por el analizador léxico se usa como entrada al programa PROLOG generado a partir de la gramática por el generador de analizadores.

III. Ventajas de PROLOG en la aplicación.

A. Facilidades en el análisis.

1. No es necesario escribir un programa especial para leer la grámatica.

Las reglas de producción pueden leerse como términos PROLOG.
Es suficiente usar la facilidad de definición de operadores que tiene el lenguaje.
Usando el predicado op (ver [ARI 84], [CLO 81])

Es necesario definir :
- ^^ y ---> como operadores infijos,
- @ y { como operadores prefijos,
- } como operador sufijo.

2. La gramática es muy fácilmente modificable

Cualquier opción nueva necesita solamente de la adición, y modificación de las reglas de producción de la gramática.
El generador de analizadores se encarga del trabajo y por su potencia al incluir la posibilidad de look-ahead y de utilización de predicados PROLOG es una herramienta que no debe ser modificada.

Es necesario anotar aquí una de las características de PROLOG que puede pasar desapercibida:

la posibilidad de generar por medio de un programa PROLOG otro al cúal pueda cedersele el control

esto se logra mediante el uso de los predicados call, functor arg y =.. que no son necesariamente parte de todas las implementaciones existentes del lenguaje como la versión de TURBO-PROLOG (ver [TUR 86])

R.Facilidades de generalización.

La aplicación que trabaja sobre el modelo entidad-relacion es general en cuanto al tipo de objetos que el usuario puede definir en su base de datos.

Es en cuanto al idioma escogido para hacer las preguntas que hay que efectuar modificaciones sobre el programa.

Prolog permite manejar las palabras redundantes, las palabras del metalenguaje, y los sinónimos en forma de tablas lo que los hace fáciles de identificar y de modificar.

A continuación se presentan algunas de estas tablas en la aplicación:

```
/* tabla con algunas de las palabras redundantes al hacer
 preguntas en espanol */

 sino(informacion).          sino(el).
 sino(aparecer).             sino(dar).
 sino(lo).                   sino(haya).
 sino(decir).                sino(cuyo).
 sino(saber).                sino(conocer).
 sino(mostrar).              sino(indicar).

/* tabla de formas normales de palabras del lenguaje usado para
  hacer preguntas en espanol */

 sinon_l(saber,[sabe,saben,sabes,sepas,sepan,sepa]).
 sinon_l(conocer,[conoce,conoces,conocen,conozca]).
 sinon_l(listar,[liste,listeme,listenos]).
 sinon_l(lo,[aquello,aquella,aquellas,aquellos]).
 sinon_l(el,[los,la,las]).
 sinon_l(dar,[deme,denos]).
 sinon_l(decir,[diga,digame,diganos]).
 sinon_l(indicar,[indiqueme,indiquenos,indique]).
 sinon_l(informacion,[informaciones]).
 sinon_l(mostrar,[muestreme,muestrenos,muestre]).
 sinon_l(ser,[es,son]).
 sinon_l(cuyo,[cuya,cuyas,cuyos]).
```

No se pensó en lo mismo con los mensajes al usuario y es entonces necesario inspeccionar todo el código para cambiar de idioma en la comunicación que parte del sistema hacia el usuario.

Sinembargo la declarativiдad de prolog permite facilmente solucionar este problema para facilitar su modificacion:

ie: write('desea dar informacion sobre'), write(X)
donde X es un elemento del lenguaje del modelo entidad-relación

puede transformarse en:
write_meta(prompt_info),write_meta(X)

el predicado write_meta se puede definir:

```
write_meta(Cad):- tabla_prompt_(Cad,Cad1), !, write(Cad1).
write_meta(Cad):- write(Cad).
```

como parte de tabla_prompt, se tendría el hecho:

```
tabla_prompt(prompt_info,'desea dar informacion sobre').
```

La introducción de la tabla localiza en un solo sitio los cambios que habrían de hacerse, a nivel de preguntas del sistema, cuando se cambia de idioma.

C.Facilidad para encontrar la respuesta a una pregunta.

La idea del módulo de respuesta se resume en:

- identificar el objeto sobre el que se pregunta y por ende su representación PROLOG
- identificar las características que debe cumplir este objeto; los valores que deben tomar algunos de sus atributos. Esto se logra con una lista de posición y valor para cada uno de los atributos.
- construir un patrón de respuesta que se unificará - haciendo uso de la unificación de PROLOG - con los hechos correspondientes en la base de datos.

Los argumentos de entrada utilizados por este módulo son obtenidos por el programa de análisis. Como se verá a continuación la programación del patrón de respuesta es muy simple y su uso es general.

El predicado siguente permite construirlo :

```
/* patron_resp(Objeto,Argumentos,Lista_restricciones,Patrón)
    Objeto  :  nombre en la base de datos del objeto por el que se
               pregunta
    Argumentos : Número de elementos del objeto en cuestión

    Lista_de_restricciones : lista de parejas (posición, valor)
        posición: sitio donde se representa el atributo
                  restringido
        valor: el valor que el atributo debe tomar

    Patrón : Patrón de Respuesta a buscar en la base de datos  */

 patron_resp(Obj,Arg,L,P):-
             functor(P,Obj,Arg),
             llene_functor(L,P).

 llene_functor([],P):-!.
 llene_functor([[Pos|Valor]|Resto_L],P):-
                          arg(Pos,P,Valor),
                          llene_functor(Resto_L,P).
```

(los predicados arg y functor son los standard PROLOG)

D.Facilidades de consulta.

Cuando se hace un proceso repetitivo donde no es necesario guardar información sobre lo que pasa en cada una de las iteraciones, PROLOG permite un esquema de trabajo muy simple y efectivo.
Este esquema se usa para generar las repetidas preguntas del usuario.

Las cláusulas PROLOG que siguen ilustran la forma de realizar la consulta:

```
preguntas :- repeat,
       write('escriba su pregunta'),nl,
       analice,  /* predicado que se ocupa de todos
                    los pormenores de una consulta
                    solo se satisface una vez */
       mire_siga.

mire_siga:- write('hay mas preguntas ?'),
            escriba_s_n, !, fail.
mire_siga:- !.

escriba_s_n:- write('conteste:(s/n)'),
              get(Z),
              low_case(Z,Z1), /* transforma un caracter mayuscula
                                 en minuscula todo otro caracter
                                 lo deja igual */
              mire_resp(Z1),!.

/* clausulas para manejar la contestacion s/n */
 mire_resp(Z):- name(s,[Z]),!.
 mire_resp(Z):- name(n,[Z]),!, fail.
 mire_resp(_):- escriba_s_n. /* repite el proceso en caso de
                                respuesta no contemplada */
```

Como puede verse el predicado mire_resp:

- falla cuando el usuario responde con 'n'. En este caso mire_siga tendrá éxito y el predicado preguntas terminará correctamente.

- tiene éxito cuando el usuario responde con 's' lo que genera un nuevo intento de contestación; como repeat siempre puede recontestarse, se genera nuevamente el proceso.

La combinación repeat-fail es recomendada por muchos de los implementadores de PROLOG ([ART 84],[EXP 83], [TUR 86]) porqué permite un ahorro importante de espacio y es muy sencilla de utilizar para describir procesos iterativos cuyas iteraciones son independientes entre sí.

IV. MEJORAS Y LIMITACIONES.

Es claro que la aplicación descrita dista mucho de un producto terminado pero sirve para conocer pronto qué tan lejos se puede llegar, y para ilustrar algunas de las facilidades y técnicas para escribir en PROLOG procesos que no estamos acostumbrados a usar por la falta de un mecanismo de unificación o de re_ensayo (vg: la combinación repeat-fail)

Otros de las limitaciones del sistema presentado son :

- una gramática que solo contempla algunas pocas construcciones sintácticas del español
  (fácilmente corregible aumentandola)

- solo en algunos casos particulares se pueden usar correctamente las locuciones y grupos de palabras como unidades que constituyen un solo símbolo :

  - acerca de
    debe tomarse como 'acerca de' sinónimo de 'sobre'
    y no como los dos símbolos: acerca, de

  - atributo de relación
    no debe ser una serie de tres símbolos sino un objeto del lenguaje del modelo entidad-relación que es en este caso el atributo de una relación.

  (aquí la corrección se obtiene cambiando el módulo de análisis léxico, e incluyendo una tabla con las palabras del lenguaje usado para hacer preguntas y para describir los objetos que conforman el modelo del modelo entidad-relación)

El reto y la importancia de PROLOG se centran en que el paso del prototipo esbozado - aquel que contiene todas las mejoras presentadas a lo largo de este artículo - al prototipo implementado es rápido y sencillo.
La implementación efectiva del nuevo prototipo se hizo en una semana, la aplicación ganó en generalidad y en rapidez de análisis.
Es conveniente, a la luz de lo presentado, evaluar la tesis inicial. El prototipo de una aplicación no trivial se obtuvo con rapidez, las pruebas y modificaciones de éste permitieron efectuar la construcción de uno nuevo y en general el proceso de desarrollo se agilizó.

A pesar de los muy buenos resultados es importante hacer notar que no se trabajaron problemas fundamentales de paquetes de software como son el manejo de gran cantidad de información, y las facilidades en manejo de archivos.

Los ejemplos nos muestran la efectividad de lenguaje PROLOG en campos como el manejo de símbolos, la consulta de bases de datos y la generación de interfaces hombre-máquina.

BIBLIOGRAFIA

[ARI 84] Arity/Prolog interpreter Version 3.2
Arity Corporation.

[CLO 81] W. F. Clocksin y C. S. Mellish
Programming in Prolog,
Springer-Verlag, 1981

[EXP 83] MS-DOS Prolog Version 2.1
Expert Systems Ltd.
Oxford, U,K.

[TUR 86] Turbo-Prolog Owner's Handbook
Borland