Utilización de acertijos lógicos como ejercicios motivadores para la enseñanza de la programación lógica

Jorge A. Peri y Daniela Lis Godoy

Universidad Nacional de Luján. Departamento de Ciencias Básicas. División Estadística y Sistemas. Casilla de Correo 291. (6700) Luján, Bs. As. República Argentina. E-mail: jperi@mail.unlu.edu.ar

Universidad Nacional de La Pampa. Facultad de Ingeniería. Calle 9 esq. 110. General Pico. La Pampa. República Argentina.

Se presentan algunas experiencias desarrolladas con la utilización de acertijos lógicos como ejercicios en las cátedras de programación lógica. La inclusión de esta problemática se desarrolló a partir de la intención de ampliar el conjunto de problemas que tradicionalmente se usan en los trabajos prácticos en este paradigma. En los mismos se desarrollan ejemplos que implican búsquedas y navegación en árboles y en grafos en general. Para ampliar este espectro, se recurrió a dos estrategias: por un lado se propone la redefinición lógica de problemas que tradicionalmente se resuelven dentro del paradigma procedural, y por el otro se incluyó la resolución automatizada o semiautomatizada de acertijos. En este trabajo se presentan algunos ejercicios motivadores utilizados conforme a la segunda estrategia.

En general, un acertijo lógico consiste en un tipo de problema, que se presenta con un enunciado en el cual se describe una situación, pero en forma incompleta, con datos fragmentarios y pistas indirectas. La solución de el problema consiste en obtener todos los datos que describen la situación aludida. Esta solución suele obtenerse mediante un proceso de deducción y eliminación. En programación lógica, es posible representar los datos disponibles mediante hechos y reglas, de la misma forma es posible representar la cadena de deducciones necesaria para obtener la solución. Esto permite automatizar total o parcialmente el proceso de encontrar la solución. Los trabajos prácticos basados en acertijos han demostrado poseer un efecto motivador en los estudiantes, desde adolescentes hasta alumnos de los últimos años de la carrera de sistemas, entre los últimos se ha llegado a niveles de dificultad que constituyen verdaderos desafíos intelectuales.

Palabras clave: programación lógica, acertijos, educación, trabajo práctico

Introducción

La enseñanza de la programación está fundamentalmente basada en la resolución, por parte de los alumnos, de problemas propuestos por la cátedra; por ello las guías de trabajos prácticos constituyen un recurso didáctico esencial, y de su calidad depende el éxito en el logro de los objetivos educativos. En la elaboración de una guía, la observación de dos detalles contribuye a garantizar la eficacia de este recurso: a) respetar un orden de dificultad gradualmente creciente, y b) presentar ejercicios que resulten motivadores para los alumnos. En una edición anterior de este Ateneo, se ha hecho notar la necesidad de incorporar ejercitación motivadora, en particular en la enseñanza de la recursión y estructuras repetitivas [4]. Allí se afirmó que sería deseable contar con enunciados que complementen los ejemplos clásicos de los libros de programación, y se presentaron algunos ejercicios novedosos que incluyen ejemplos de recursión en estructuras arbóreas, backtracking y recursión cruzada. En el mismo trabajo se hizo notar que el conjunto de ejercicios sobre recursión que se encuentran en los textos más usuales, resulta demasiado canónico e insuficiente. Hemos observado esta misma deficiencia en general en los textos de programación lógica, en donde se utilizan ejemplos clásicos que implican búsquedas y navegación en árboles y grafos en general.

En un trabajo posterior [10] se señalaron las complicaciones que plantea la enseñanza de la recursividad, donde se hizo referencia a la dificultad que presentan los alumnos para aprender a usar un recurso alternativo de otro en el que ya habían adquirido alguna habilidad. Allí se hizo notar que los alumnos en general no comprenden muy bien por qué deben utilizar planteos recursivos, cuando pueden encontrar soluciones satisfactorias resolviendo el problema en forma iterativa. Para superar este problema, se recurrió a una selección de ejemplos que no pueden ser resueltos por los alumnos en forma iterativa, en el nivel de desarrollo en que se encuentran cuando cursan esa asignatura. Un análogo de esta inercia intelectual puede observarse en alumnos que se enfrentan con la necesidad de introducirse en el paradigma lógico luego de haber adquirido práctica en la programación procedural. Algunas experiencias realizadas con la enseñanza del paradigma lógico [2, 12, 13], sugieren que las dificultades de los estudiantes se producen al tratar de diseñar la solución del problema en términos procedurales, para luego intentar traducirla al paradigma lógico.

En la búsqueda de ejercicios que permitan superar estas dificultades, se encontraron dos estrategias poco o nada explotadas por los textos clásicos:

a) Se les propuso a los alumnos la resolución de una clase de problemas, para los cuales la programación lógica aparece en principio como una herramienta inadecuada.

b) Se enfrentó a los alumnos con un tipo de problemas extraño al que ellos están acostumbrados a tratar, y para los cuales se hace muy difícil pensar una solución procedural.

Dentro de la primer estrategia, se han elaborado guías de trabajos prácticos que incluyen colecciones de problemas que, si bien tienen soluciones clásicas dentro del paradigma procedural [11], se transforman en verdaderos desafíos intelectuales cuando se pretende resolverlos dentro del paradigma lógico. Resultaron particularmente motivadores los ejercicios que incluyen ordenamientos y búsquedas, u operaciones con matrices, como puede ser la resolución de sistemas lineales. La necesidad de definir las matrices en forma de listas y la ausencia de índices obliga a aguzar la imaginación para encontrar el equivalente lógico del algoritmo. En un nivel mayor de complejidad, algunos alumnos han logrado redefinir algoritmos de clasificación no supervisada [5], y problemas geométricos en espacios n-dimensionales, del tipo de la búsqueda del punto más cercano o la obtención del cerco convexo.

Con respecto a la segunda estrategia, se encontró que en los acertijos lógicos o "juegos de ingenio" existe un material potencialmente interesante para la elaboración de trabajos prácticos

de programación lógica. Por una parte, en general los alumnos de informática se sienten atraídos por este tipo de desafíos intelectuales, y por la otra, el diseño de la solución resulta mucho más sencillo mediante un enfoque descriptivo. Para el dictado de las asignaturas, se recopiló una serie de acertijos [1, 6, 7, 14], y se publicó un cuadernillo [9], con un conjunto escogido de problemas resueltos y explicados, además de una serie de ejercicios propuestos; entre éstos últimos están los ejemplos aquí desarrollados.

Durante el dictado de las clases hemos comprobado que ambas estrategias permiten explorar enfoques novedosos, motivadores, y que presentan a los alumnos interesantes desafíos intelectuales.

Destinatarios

La currícula de la carrera Licenciatura en Sistemas de Información de la Universidad Nacional de Luján cuenta con cuatro asignaturas en las que se aborda o aplica la programación lógica, ellas son:

- Métodos de Programación
- Programación Funcional y Lógica
- Inteligencia Artificial
- Sistemas Expertos

Asimismo, entre las actividades extracurriculares, se ha desarrollado una experiencia de capacitación de docentes en inteligencia artificial, en donde se han utilizado ejemplos sencillos de acertijos, comprobando también su efecto motivador en este nivel. Los autores también participamos en el dictado de la asignatura Programación III, -funcional y lógica-, correspondiente a la carrera de Analista Programador en Computación, que se dicta en la Facultad de Ingeniería de la Universidad Nacional de La Pampa, en la localidad de General Pico.

En la primera de estas asignaturas, Métodos de Programación, se introduce a los alumnos en la resolución de problemas en tres paradigmas, funcional, lógico y procedural. El nivel de este curso es introductorio, por lo que las prácticas sobre programación lógica alcanzan un bajo nivel de complejidad.

Es hacia el final de la carrera, durante el dictado de Programación Funcional y Lógica, donde se profundizan los conceptos, lenguajes y prácticas de este paradigma. Acertijos con diferentes niveles de complejidad fueron propuestos dentro de los trabajos prácticos de ambas asignaturas. En Programación III, se trabaja con un nivel intermedio de complejidad.

El primero de los acertijos que se presenta más adelante, ha sido escogido de entre aquellos con menores nivel de dificultad, que son los utilizados con los alumnos de Métodos de Programación. El segundo y el tercero, en cambio, tienen un mayor nivel de dificultad.

Acertijos

Un acertijo es un tipo de problema en el cual se describe una situación, pero a través de datos fragmentarios y pistas indirectas. Consta generalmente de una descripción general del problema, en donde se pueden identificar las dimensiones y los límites, y un conjunto de pistas, indicios y condiciones. Estos últimos comúnmente caen dentro de cuatro categorías:

A. Dar un dato concreto. Por ejemplo: *Luisa siempre viste de rojo*
B. Eliminaciones. Por ejemplo: *El nombre del señor López no es Juan*
C. Límites superiores e inferiores. Por ejemplo: *María compró la prenda más económica*

D. Pistas más sutiles. Por ejemplo: *A Pedro le desagrada el rock*

Si el problema ha sido bien diseñado y redactado, a partir de esta información, y siguiendo una cadena de deducciones lógicas, es posible llegar a una solución, que suele consistir en completar explícitamente todos los datos.

En otra clase de acertijos, el enunciado consiste en la descripción de una supuesta "tarea" a desarrollar como una sucesión de subtareas, y una o más restricciones, que ponen condiciones de validez a estas subtareas. Ejemplos clásicos de este tipo de acertijos, lo constituyen el problema del lobo, la oveja y la alfalfa, el de los misioneros y los caníbales, o el de los jarros de agua que es el que presentamos en este trabajo. Obsérvese que la "solución" de este tipo de problema consiste en la definición de un algoritmo, esto es, la secuencia de subtareas válidas que permiten ejecutar con éxito la tarea principal.

Ejercicios Propuestos

Se presentan a continuación ejemplos de ejercicios utilizados para esta experiencia. Se realiza además un análisis y descripción del juego de hechos y reglas Prolog, expresado en notación de Edimburgo, que al ser consultado resuelve el problema planteado.

En general, la base de conocimiento que permita la solución de un acertijo constará de:
a) Un conjunto de hechos que permitan definir a los sujetos que intervienen.
b) Un conjunto de hechos y reglas para describir las pistas, indicios y condiciones.
c) Una o más consultas, que permitan indagar a la base sobre todos o algunos de los datos que completan la descripción de la situación.

Las mujeres y sus trabajos

Clara, Luisa, María y Nélida son cuatro mujeres que aman sus trabajos. Ellas trabajan como diseñadora de moda, florista, jardinera y directora de orquesta. Cada mujer tiene un solo trabajo, y cada trabajo es ocupado por una sola mujer. Con las siguientes pistas, encontrar el trabajo realizado por cada mujer:

(a) Clara es violentamente alérgica a las plantas.
(b) Luisa y la florista comparten el departamento
(c) A María y Luisa les gusta solamente la música rock
(d) La jardinera, la diseñadora de modas y Nélida no se conocen entre sí.

Para comenzar es necesario identificar cuales son los sujetos que participan en el problema, en este caso son mujeres y ocupaciones. Esto se describe con los siguientes hechos:

```
mujer(clara).
mujer(luisa).
mujer(maria).
mujer(nelida).

ocupacion(diseñadora).
ocupacion (florista).
ocupacion (jardinera).
ocupacion (directora_de_orquesta).
```

A continuaciuón es necesario comenzar a analizar cada una de las pistas mencionadas en el planteo del problema, con el objetivo de expresarlas en forma de hechos o reglas. Puede observarse que en todos los casos se trata de eliminaciones, por lo tanto se introduce un predicado *no_es,* el cuál es verdadero cuando su primer sujeto es una mujer que no tiene la ocupación indicada por el segundo sujeto.

La pista (a) sugiere que Clara no puede tener un trabajo relacionado con plantas ya que es alérgica a ellas, por lo tanto se pueden inferir los siguientes hechos:

(1)

```
no_es(clara,florista).
no_es(clara,jardinera).
```

Si Luisa y la florista comparten el departamento, como dice la pista (b), no pueden ser la misma persona, de lo que se deduce:

```
no_es(luisa,florista).
```

La pista (c) permite obtener los siguientes hechos:

(2)

```
no_es(luisa,directora_de_orquesta).
no_es(maria, directora_de_orquesta).
```

es decir, ni Luisa ni María pueden ser directoras de orquesta ya que a ellas únicamente les gusta la música de rock.

Dos hechos más se obtienen de la última pista al suponer que si Nélida no conoce a la jardinera ni a la diseñadora de modas ella no puede tener tales ocupaciones:

```
no_es(nelida, jardinera).
no_es(nelida, diseñadora).
```

Los hechos obtenidos de las pistas (a) y (c) surgieron, en realidad, de haber hecho inferencias previas, es decir de relacionar lo que dice el enunciado con algo que era conocido de antemano. Con un nivel de complejidad un poco mayor, se puede lograr que estas inferencias se hagan automáticamente en el momento de la consulta. Para el primer caso por ejemplo, todo lo que dice la pista (a) es que Clara es alérgica a las plantas, lo cual es directamente representable en Prolog de la siguiente manera:

(3)

```
alergica(clara, plantas).
```

pero hay datos que se usaron implícitamente para obtener los hechos indicados en [4] y es que tanto la ocupación de jardinera como la de florista están relacionadas con las plantas:

(4)

```
relacionado(jardinera, plantas).
relacionado(florista, plantas).
```

y además una mujer no puede tener una ocupación que esté relacionada con algo a lo que es alérgica:

(5)

```
no_es(Mujer, Ocupacion) :-
  alergica(Mujer, X),
  relacionado(Ocupacion, X).
```

De manera que (1) puede ser reemplazado por (3), (4) y (5), obteniendo los mismos resultados. Sin embargo, la segunda opción es preferible por sobre la primera ya que representa con mayor fidelidad la información implícita en la pista (a).

En el mismo sentido la pista (c) da cuenta que Luisa y María sólo escuchan música de rock, es decir:

(6)

```
solo_escucha(luisa, rock).
solo_escucha(maria, rock).
```

dicho estilo musical es incompatible con la profesión de directora de orquesta:

(7)

```
incompatibles(directora_de_orquesta, rock).
```

y, por otro lado, una mujer que sólo escucha determinada música no puede tener una ocupación que sea incompatible con dicha música:

(8)

```
no_es(Mujer, Ocupacion) :-
  solo_escucha(Mujer,X),
  incompatibles(Ocupacion,X).
```

Una forma más deseable de representar la pista (c) se obtiene entonces al reemplazar (2) por (6), (7) y (8). Es posible hacer un análisis similar con las pistas b y d. Obsérvese que mediante la explicitación de estas inferencias puede aumentarse la complejidad de la base, y por lo tanto el nivel de dificultad del ejercicio. Podrían lograrse versiones mas complejas que la propuesta aquí, explicitando pistas más sutiles como *las mujeres aman sus trabajos*.

En este momento encontrar la solución al problema se reduce a obtener cuatro combinaciones mujer-ocupación diferentes. Donde el nombre de las mujeres debe ser un nombre válido, es decir debe estar declarado mediante el predicado *mujer*, al igual que las ocupaciones deben ser ocupaciones válidas, declaradas mediante el predicado *ocupación*.

Una pista más sutil dentro del enunciado del problema indicaba que c*ada mujer tiene un solo trabajo, y cada trabajo es ocupado por una sola mujer* por lo que las cuatro mujeres elegidas y las cuatro ocupaciones deben ser diferentes entre sí. Para garantizar esta unicidad se usan dos predicados, un predicado llamado *diferentes* que será verdadero cuando sus cuatro sujetos sean distintos:

```
diferentes(A, B, C, D) :-
  not(A == B),
  not(A == C),
  not(A == D),
  not(B == C),
  not(B == D),
  not(C == D).
```

y otro predicado respuesta, cuyo sujeto es una lista de pares mujer-ocupación, y que junto con *diferentes* garantice que no hay repeticiones:

```
respuesta([[M1,O1],[M2,O2],[M3,O3],[M4,O4]]) :-
  mujer(M1),
  mujer(M2),
  mujer(M3),
  mujer(M4),
```

```
    ocupacion(O1),
    ocupacion(O2),
    ocupacion(O3),
    ocupacion(O4),
    diferentes(M1,M2,M3,M4),
    diferentes(O1,O2,O3,O4),
    not(no_es(M1,O1)),
    not(no_es(M2, O2)),
    not(no_es(M3, O3)),
    not(no_es(M4, O4)).
```

A efectos de mostrar la solución, este último predicado puede ser consultado desde el intérprete Prolog de la siguiente manera:

```
?- respuesta(X).
X=[[clara,diseñadora],[luisa,jardinera],[maria,florista],[nelida,directora_de_orque
sta]]
Yes
```

Teniendo en cuenta que en un programa lógico se pueden identificar los análogos de las estructuras básicas de la programación procedural [3], puede observarse que este ejercicio no contiene descripciones explícitas de los correspondientes de estructuras alternativas ni repetitivas, quedando ambas enmascaradas por la definición múltiple de reglas y por el backtracking.

El siguiente ejemplo es un acertijo interesante [6], que conceptualmente posee un nivel de dificultad un poco superior al anterior, pero que puede resolverse con un juego de reglas sorprendentemente compacto:

Sobre líquidos venenosos

El Sr. Ido, el químico, tiene seis frascos llenos de líquidos coloreados. Hay uno de cada color: rojo, anaranjado, amarillo, verde, azul y violeta. El señor Ido sabe que algunos de esos líquidos son tóxicos, pero no recuerda cuales...

Sin embargo, sí recuerda algunos datos. En cada uno de los siguientes pares de frascos hay uno con veneno y otro no:

a) los frascos violeta y azul
b) los frascos rojo y amarillo
c) los frascos azul y anaranjado

El Sr. Ido recuerda también que en estos otros pares de frascos hay uno sin veneno:

d) el violeta y el amarillo
e) el rojo y el anaranjado
f) el verde y el azul

¡Ah! Casi lo olvido, añade el Sr. Ido, el líquido del frasco rojo no es venenoso. ¿Qué frascos tienen veneno?

De la misma forma que en el acertijo anterior se comienzan describiendo aquellos hechos que son conocidos. En este caso de lo que esta seguro el Sr. Ido es de que en los pares de frascos señalados en a, b y c uno contienen veneno y otro no, lo que puede escribirse en Prolog de la siguiente manera:

```
uno_toxico_otro_no (violeta,azul).
uno_toxico_otro_no (rojo,amarillo).
uno_toxico_otro_no (azul,anaranjado).
```

También tiene seguridad el Sr. Ido de que en los pares de frascos indicados en d, e y f hay uno sin veneno:

```
uno_no_toxico (violeta,amarillo).
uno_no_toxico (rojo,anaranjado).
uno_no_toxico (verde,azul).
```

Se utilizan dos predicados en la resolución de este acertijo, *no_toxico* y *tóxico*, *no_tóxico* es verdadero cuando su único sujeto es un frasco que no contiene veneno, en cambio *tóxico*, es verdadero cuando su sujeto es un frasco que sí contiene veneno.

Dado que se conocen pares en los que hay un frasco con veneno y otro no (a-c), se puede decir que un frasco es tóxico si pertenece a uno de esos pares junto con otro frasco que no lo es. En términos de Prolog:

```
toxico(X) :-
  (uno_toxico_otro_no (X,Y); uno_toxico_otro_no (Y,X)),
  no_toxico(Y).
```

Como también se conocen pares de frascos en los que hay uno sin veneno (d-f), se puede decir que un frasco no contiene veneno si pertenece a uno de esos pares y se puede demostrar que el frasco que lo acompaña sí es tóxico.

```
no_toxico(X) :-
  (uno_no_toxico (X,Y); uno_no_toxico (Y,X)),
  toxico(Y).
```

El Sr. Ido da un dato concreto al recordar que el frasco rojo no es venenoso:

```
no_toxico(rojo).
```

En este punto se cuenta con toda la información del enunciado expresado en forma de hechos y reglas en Prolog, de manera que para obtener la respuesta puede efectuarse la siguiente consulta a la base de datos:

```
?- findall(X,toxico(X),L).
L = [azul, amarillo]
Yes
```

donde *findall* es un predicado predefinido que es verdadero cuando la variable indicada como tercer sujeto (L), pueda unificarse con la lista de todas las instancias de la variable del primer sujeto (X), que hacen verdadero el predicado indicado en segundo lugar (toxico(X)). Es decir L es la lista de todos los frascos de los cuales se puede probar que son tóxicos.

A diferencia del problema anterior, aquí sí existen descripciones explícitas de análogos de la estructura alternativa, que son los "or" en las definiciones de "tóxico" y no_tóxico". También está explicitamente expresada la estructura repetitiva en la recursividad cruzada entre los predicados mencionados.

En el siguiente acertijo, la *solución* del mismo consiste en encontrar un algoritmo, es decir, cuál es la secuencia de tareas que permite llegar a un objetivo.

Los jarros de agua

Se dispone de dos jarros, uno con una capacidad de 3 litros, y otro con una capacidad de 4 litros. Ninguno de los dos tiene marcas que permitan medir cantidades que no sean las de sus propias capacidades.

Existe una canilla que permite llenar los jarros con agua y un sumidero donde se puede vaciar los mismos. El problema consiste en encontrar cuál es la secuencia de movimientos de llenado, vaciado y trasvase que permitan obtener exactamente dos litros de agua en el jarro de 4 litros.

El juego de reglas propuesto permite consultas acotadas a una cantidad de movimientos propuesta por el usuario, obviamente, el problema no tiene soluciones posibles para una cantidad de movimientos menor que un mínimo, que en este caso es seis.

Se comienza con el predicado *movida*, que relaciona los estados de los jarros antes y después de cada movimiento. La relación también abarca una lista con una oración donde se relata el tipo de movimiento hecho, estas oraciones formarán una lista con la respuesta, que se mostrará por pantalla. La proposición tiene la forma:

movida(A3, D3, A4, D4, Movimiento)

en la cuál los sujetos son:

A3: Nivel del jarro de 3 litros antes del movimiento.

D3: Nivel del jarro de 3 litros después del movimiento.

A4: Nivel del jarro de 4 litros antes del movimiento.

D4: Nivel del jarro de 4 litros después del movimiento.

Movimiento: Descripción del movimiento hecho.

Las tres operaciones posibles con cada jarro son: llenarlo, vaciarlo y pasar agua al otro jarro. Como cada una de estas tres operaciones se puede aplicar a cada uno de los dos jarros, existe un total de seis operaciones permitidas, que da lugar a cada una de las seis ocurrencias del predicado *movida*:

Al llenar el jarro de tres litros

```
Movida(A3, 3, X, X, [llenar_el_jarro_de_tres_litros]) :-
  A3 < 3.
```

donde se dice que para poder llenar el jarro de tres litros es condición que antes del llenado el jarro tenga menos de tres litros. En esta proposición también se dice que cualquiera que sea la cantidad de agua que hay en el jarro de tres litros antes de la operación, luego de la misma hay tres litros. Algo similar se describe para el llenado del
jarro de cuatro litros

```
movida(X, X, A4, 4, [llenar_el_jarro_de_cuatro_litros]) :-
  A4 < 4.
```

Con respecto al vaciado de los jarros, la condición es que no estén previamente vacíos, mientras que después de la operación su contenido es cero

```
movida(A3, 0, X, X, [vaciar_el_jarro_de_tres_litros]) :-
   A3 /= 0.

movida(X, X, A4, 0, [vaciar_el_jarro_de_cuatro_litros]) :-
   A4 /= 0.
```

Para el pasaje de agua de un jarro al otro, las reglas son un poco mas complejas. La relación *movida* es verdadera si existe un predicado que describa la acción de trasvasar, ya que *movida* sólo relaciona las cantidades antes y después del trasvase, sin interesar como se hace

```
movida(A3, D3, A4, D4, [pasar_lo_posible_del_de_4-l_al_de_3_l]) :-
  trasvase(A4, D4, 3, A3, D3).


movida(A3, D3, A4, D4, [pasar_lo_posible_del_de_3_l_al_de_4_l]) :-
  trasvase(A3, D3, 4, A4, D4).
```

El predicado TRASVASE relaciona cinco sujetos

trasvase(AE, PE, CD, AD, PD)

donde

AE: Nivel anterior al trasvase en el jarro del cual se extrae agua.

PE: Nivel posterior al trasvase en el jarro del cual se extrae agua.

CD: Capacidad del jarro destino.

AD: Nivel anterior al trasvase en el jarro destino.

PD: Nivel posterior al trasvase en el jarro destino.

La relación TRASVASE, describe en que casos es posible trasvasar, como se produce el trasvase, y como se modifican los valores en juego debido a la operación

```
trasvase(AE, PE, CD, AD, PD) :-
  AE /= 0,
  CD /= AD,
  L is CD - AD,
  minimo(L, AE, M),
  PE is AE - M,
  PD is AD + M.
```

Dicho en lenguaje natural, esta relación dice que se trasvasan los litros posibles desde un jarro, -que tenía "AE" litros y queda con "PE" litros-, hasta un jarro de capacidad "CD", -que tenía "AD" litros quedando con "PD" litros-. Para que este predicado sea verdadero, es necesario que se verifiquen las siguientes proposiciones:

- El jarro desde el cual se trasvasa agua no debe estar vacío antes de efectuar la operación

AE /= 0

- El jarro hacia el cual se trasvasa agua no debe estar lleno antes de efectuar la operación

CD /= AD

- Es necesario conocer cuanto espacio libre tiene el jarro hacia el cual se trasvasa agua

L is CD - AD

donde "L" es el espacio libre, ya que "CD" es la capacidad total y "AD" es la cantidad que tiene antes de la operación de trasvasado.

- La cantidad de agua trasvasada debe cumplir con dos condiciones: no debe ser mayor que la cantidad disponible en el jarro fuente, ni mayor que el espacio libre que tenga el jarro destino, las dos condiciones se cumplen si la cantidad de agua trasvasada es el mínimo entre estos dos valores

minimo(L, AE, M)

donde "L" es la cantidad que se trasvasa

- La cantidad de agua que queda en el jarro fuente es la que tiene antes del trasvase menos la cantidad trasvasada

PE is AE - M,

- La cantidad de agua que queda en el jarro destino es igual a la que tenía mas la trasvasada

PD is AD + M.

El predicado *minimo* relaciona dos números con el menor de ellos, existiendo tres situaciones posibles: igualdad, el primero menor y el segundo menor

```
minimo(X, X, X).

minimo(X, Y, X) :-
  X < Y.

minimo(X, Y, Y) :-
  Y < X.
```

Una vez definidas estas reglas referidas a las condiciones que debe cumplir el algoritmo solución, se puede plantear una regla que es verdadera cuando existe solución para un número dado de operaciones. Se propone una regla de la forma

prueba(CA3, CA4, N, MAX, [X|Y])

donde los predicados son

CA3: Cantidad de agua contenida en el jarro de tres litros.

CA4: Cantidad de agua contenida en el jarro de cuatro litros.

N : Número de operación

MAX: Número máximo de movidas permitidas

[y|z]: [X|Y]: Lista de soluciones

Esta relación recursiva tiene un caso embrionario, tal que si llega a él se está frente a una de las posibles soluciones

```
prueba(X, 2, Y, Y, []) :- !.
```

Para que se cumpla esta regla, el jarro de cuatro litros debe tener dos litros de agua y el número de movimiento tiene que ser igual al número de movimientos permitidos. La solución es válida solamente si se la obtiene con el mismo número de movidas. Esta es una condición implícita que se logra al poner iguales el tercer y cuarto sujetos, la misma hace que se reporten solamente las soluciones que son válidas para cada nodo particular de búsqueda.

El caso general

```
prueba(CA3, CA4, N, MAX, [X|Y])
```

se cumple cuando el número de movida es menor que el número máximo posible, lo que quiere decir que hace falta hacer mas operaciones. Cada invocación recursiva de esta regla corresponde a un movimiento u operación. Esta proposición es verdadera cuando efectivamente el número de movimiento es menor que el permitido

```
N < MAX.
```

cuando es posible realizar una movida descripta en "X", después de la cuál los jarros de tres y cuatro litros quedan con las cantidades "D3" y "D4" respectivamente

```
movida(CA3, D3, CA4, D4, X)
```

y que además es posible realizar otra movida con los nuevos niveles de los jarros, y cuyo número de movida será el siguiente del actual

```
N2 is N + 1
prueba(D3, _, D4, _, N2)
```

Juntando todas las condiciones anteriores en una regla, queda definido el caso general de *prueba:*

```
Prueba(CA3, CA4, N, MAX, [X|Y]) :-
  N < MAX,
  N2 is N + 1,
  Movida(CA3, D3, CA4, D4, X),
  Prueba(D3, D4, N2, MAX, Y).
```

Esta base puede consultarse con *prueba*, teniendo en cuenta que al comienzo los dos jarros están vacíos

```
?-prueba(0, 0, 0, 6, X).
X =[[llenar_el_jarro_de_tres_litros]
[pasar_lo_posible_del_de_3_l_al_de_4_l]
[llenar_el_jarro_de_tres_litros]
[pasar_lo_posible_del_de_3_l_al_de_4_l]
```

[vaciar_el_jarro_de_cuatro_litros]
[pasar_lo_posible_del_de_3_l_al_de_4_l]]

los renglones anteriores describen el algoritmo que constituye una de las posibles soluciones al problema.

Conclusiones

Se han mostrado algunos ejemplos, que en nuestros cursos nos demostraron el potencial de los acertijos lógicos como motivadores para la enseñanza de la programación lógica. La resolución de acertijos lógicos en este paradigma, ha sido usada en trabajos prácticos correspondientes a asignaturas de distintos niveles, que abarcan desde un curso para docentes de la EGB y polimodal, -sin experiencia previa en programación-, hasta asignaturas destinadas a alumnos de los últimos años de la carrera de Sistemas de Información. En todos los casos se ha comprobado su efecto motivador. Es necesario señalar que los resultados presentados son totalmente prácticos, y solo pretenden servir como un modesto aporte al bagaje de ejercicios que pueden usarse en trabajos prácticos de este paradigma descriptivo.

Referencias

[1] Brandreth, G. P.: *Acertijos modernos*. Selector Actualidad Editorial. México. 1990.

[2] Bottino, R.M.: Comparing different approaches to programming from an educational viewpoint. *Computer Education*. Vol 18, No 4, pp. 273-281, 1992

[3] Covington, M.A.: Expressing procedural algorithms in Prolog. *Research Report 01-0012*. University of Georgia. Athens. 1986.

[4] Chesñevar, C. I.: Algunas consideraciones y ejercicios motivadores para la enseñanza de la recursión. *Segundo Ateneo de Profesores Universitarios de Computación y Sistemas*.

[5] Ferrarotti, F.A.: Enfoque descriptivo del algoritmo Chain-Map de agrupación de clases. *27 JAIIO*. Buenos Aires. (1998)

[6] Holt, M.: *Matemáticas recreativas*. Ediciones Martinez Roca. Barcelona. 1988.

[7] Perelmán, Y.: *Matemáticas recreativas*. Editorial Mir. Moscú. 1988.

[8] Peri, J. A. Y Godoy, D. L.: Teaching artificial intelligence with Edulog: a pilot with High School Teachers. Presentado a: *The Third International Conference on the Learning Sciences (ICLS-98)*. Atlanta. 1998.

[9] Peri, J.: *Resolviendo acertijos con Prolog*. Fundación Encuentro. Luján. 1996.

[10] Rueda, S. y Castro, S.: Recursividad Esencial en la Resolución de Problemas. *Cuarto Ateneo de Profesores Universitarios de Computación y Sistemas*.

[11] Sedgewick, R.: *Algorithms in C++*. Addison-Wesley. Massachusetts. (1992).

[12] van Someren, M.: What's wrong? Understanding beginners problems with Prolog. *Instructional Science*. 19(415):257-282. 1990.

[13] van Someren, M.: Undestanding students: errors with Prolog unification. *Instructional Science*, 19(415):316-376, 1990.

[14] Wujec, T.: *Mental Gym*. Editorial Atlántida. Buenos Aires. 1994.