PROGRAMACION LOGICA BASADA EN ASERCIONES
Una herejía imperativa

por

JAIME A. BOHORQUEZ V.

Departamento de Sistemas y Computación.
Universidad de Los Andes
Aptdo Aéreo 4976
Bogotá
COLOMBIA

Resumen: Se propone una interpretación de la ejecución de un programa lógico en términos de un esquema de transición de estados. Con base en esta interpretación se define la noción de corrección total basada en especificaciones expresadas mediante aserciones, y se desarrolla un cálculo de corrección en el estilo de la lógica de Hoare [H69]. Se dan ejemplos ilustrativos.

Palabras claves: Programación lógica, aserciones, lógica de Hoare, corrección total, lenguaje PROLOG.

0. Introducción.

La programación lógica consiste en la interpretación de la lógica de predicados como un paradigma de programación (alternativo a la programación imperativa y a la programación funcional) para el cual cierto tipo de fórmulas (llamadas cláusulas) constituyen un lenguaje de programación.

El primer lenguaje lógico llamado PROLOG [C73], [R75] fué diseñado e implantado en 1972 y se basó en la interpretación procedimental de las cláusulas de Horn debida a Kowalski [K74]. Posteriormente se diseñó un compilador de PROLOG (escrito en PROLOG) que fué implantado por Warren, Pereira y Pereira [W77], mostrando que el compilador PROLOG ejecutaba programas lógicos tan eficientemente como LISP compilado.

En la programación lógica se expresa únicamente la lógica de los métodos de solución de los problemas, en contraste con los programas convencionales, los cuales mezclan la lógica del conocimiento usado en solucionar los problemas, con el control de los métodos que procesan la información. Por esta razón se afirma que los programas lógicos son fáciles de entender, verificar y cambiar y que serían especialmente adecuados para programadores principiantes y usuarios de bases de datos que no quieran saber de los detalles sobre el control necesario para ejecutar sus programas.

Sin embargo, hay un precio que pagar por el logro de las ventajas mencionadas anteriormente, y es a costa de la eficiencia del programa obtenido al tratar la lógica simplemente como un lenguaje de especificación sin tener en cuenta sus aspectos pragmáticos (ver [K79] cap. 5).

Para el programador profesional la eficiencia es muchas veces un factor de importancia para juzgar la bondad de un programa, lo que muchas veces implica una pérdida de claridad en la interpretación de las cláusulas que lo constituyen. Esto sin detrimento de la prioridad e importancia que tiene la corrección en el desarrollo de programas.

En consecuencia, a pesar de las cualidades autodescriptivas propias de la programación lógica, en ocasiones hacen necesarias técnicas suplementarias de documentación y verificación de ciertos programas lógicos cuya claridad ha sido sacrificada en aras de un desempeño eficiente. Tal situación se presenta especialmente cierto cuando los programas lógicos se implantan en una máquina secuencial.

Adicionalmente, muchas veces durante la labor misma del diseño de un programa lógico eficiente, el programador familiarizado con la programación imperativa encuentra que una solución sencilla y eficiente para el problema que tiene entre manos es, por ejemplo, la de un ciclo o proceso iterativo; debido al espíritu declarativo característico de la programación lógica, se ve entonces abocado a un bloqueo mental causado por el hecho de estar pensan-

do una solución propia del estilo imperativo de programar.

El propósito de este artículo es el de ayudar al programador de marras a salir de su problema mediante la interpretación de una clase propia de programas lógicos por medio de una semántica operacional basada en una máquina de transición de estados. Esta semántica permitirá el desarrollo de soluciones que se podrán interpretar como iterativas. Además, los programas pertenecientes a la clase mencionada serán suceptibles de documentación por medio de aserciones, y también de verificación mediante un cálculo axiomático inspirado en reglas de inferencia en el estilo de Hoare [H69].

La organización de este artículo será la siguiente: primeramente se presentará la sintaxis y nociones básicas de la lógica clausal. En la sección 2 se explican las nociones de sustitución, unificación, asociación y estado. La sección 3 da una interpretación procedimental de un conjunto de cláusulas de Horn como programa lógico. En la sección 4 se presenta el concepto de corrección de un procedimiento lógico basada en una noción de estado que corresponde a la colección de valores asociados a las variables del procedimiento. La sección 5 describe una semántica operacional que permite interpretar un subconjunto propio de programas lógicos como procedimientos iterativos. En la sección 6 se exhibe un calculo inferencial para demostrar la corrección de un procedimiento lógico. Finalmente en la sección 7 se presentan aplicaciones y ejemplos ilustrativos.

1. La Lógica Clausal.

Un identificador es una cadena finita de símbolos o caracteres alfanuméricos, no todos ellos numéricos.

Una variable se representa mediante un identificador que comience con una letra minúscula:

x a x4 an25

Una constante es cualquier número entero o identificador que no comienza con letra minúscula:

123 Abc 1a MB Persona Sumando

denotarán objetos resultantes de la conceptualización de un problema.

Un término es una variable, una constante o una expresión $f(t_1,...,t_m)$ $(m \geq 1)$ donde f es un identificador (que comienza con minúscula) llamado símbolo de función y los t_i son a su vez términos:

x 123 edad(Jorge) inverso(123)

Una fórmula atómica (o simplemente un átomo) es una expresión de

la forma $p(t_1,...,t_m)$ donde p es un identificador que corresponde a un símbolo de relación n-aria y $t_1,..,t_m$ es una secuencia posiblemente nula (m=0) de términos, usada para expresar relaciones entre términos:

invierta(x) concatene(cons(u,v),L,z)

Una <u>expresión</u> es un átomo o un término.

Una <u>expresión explicita</u> es cualquier expresión que no involucra variables.

Un <u>hecho</u> es un átomo, v.gr. $p(t_1,t_2,...,t_m)$. En este caso se dice que $p(t_1,...,t_m)$ es un hecho acerca de la relación p. Un hecho se interpreta como una afirmación o hipótesis, con todas sus variables cuantificadas universalmente, que puede ser usada para deducir otro hecho o contestar una pregunta.

Una <u>regla</u> consta de dos partes: un átomo llamado <u>cabeza</u> y una lista ordenada de átomos llamada <u>cola</u>. Notacionalmente, en la regla $B:- A_1,..,A_n$. B es la cabeza y $A_1,...,A_n$ es la cola. Si el símbolo asociado a la cabeza de la regla es p, se dice que es una regla acerca de (la relación) p.
Una regla con variables $x_1,..,x_k$ tiene la siguiente interpretación: para todos los valores de $x_1,...,x_k$ vale que lo afirmado por la cabeza de la regla es cierta si se puede establecer la verdad de cada una de las condiciones en la cola de la regla.
Procedimentalmente, una regla $B:- A_1,...,A_n$ se interpreta como un comando condicional o guardado para ejecutar la serie de llamadas de procedimiento $A_1,...,A_n$ cuya condición o guarda es B.

Ejemplo: Si la cabeza de una regla es el átomo abuelo(x,y) y su cola corresponde a la secuencia progenitor(x,z), progenitor(z,y) esta es una regla acerca de la relación "ser abuelo de" y se denota de la siguiente manera:

```
abuelo(x,y):- progenitor(x,z),progenitor(z,x).
```

Una <u>cláusula de Horn</u> es una regla o un hecho.

Una <u>regla recursiva</u> es una regla cuya cola incluye por lo menos un átomo con el mismo identificador que el de la cabeza de la regla, v.gr.

```
p(x) :- q(x,A), p(A), r(x1), p(x).
```

Una <u>regla recursiva terminal</u> es una regla recursiva para la cual únicamente el identificador de el último átomo de su cola coincide con el identificador de su cabeza, v.gr.

```
concat(cons(u,v),y,cons(u,z)):- concat(v,y,z).

p(x):- q(x,x1), p(x).
```

Una <u>lista de objetivos</u> es una sucesión finita de atomos

$<Q_1,\ldots,Q_s>$. Esta noción corresponde a una pregunta o consulta cuya respuesta debe deducirse mediante inferencias, de los hechos y reglas con que se cuenta. Cada uno de los átomos que conforma la consulta se llama un objetivo. La consulta correspondiente a la lista de objetivos $<Q_1,\ldots,Q_s>$ se denota ?- $Q_1,\ldots,Q_s$.

Un procedimiento lógico se denota mediante un símbolo de relación n-aria ($n\geq 0$) (su identificador) seguido de una sucesión de n símbolos de variable llamados parámetros digamos $p(x_1,\ldots,x_n)$, y está compuesto de una secuencia finita de cláusulas acerca de p, llamada el cuerpo de p. Un procedimiento lógico se interpreta como la conjunción de sus cláusulas. Procedimentalmente, el cuerpo de p corresponde a la declaración del procedimiento.

Un programa lógico es un conjunto de procedimientos lógicos. Se interpreta como la conjunción de las cláusulas pertenecientes a los procedimientos lógicos que las conforman.

2. Sustituciones, asociaciones y noción de estado.

Una sustitución S es un conjunto finito de la forma $\{v_1/t_1,\ldots,v_n/t_n\}$ donde cada v_i es una variable y cada t_i es un término diferente de v_i y además $v_i \neq v_j$ para $i\neq j$.

Una sustitución corresponde a una función del conjunto de todas las variables en el conjunto de todos los términos donde la imagen de una variable cualquiera v es t_i, si $v=v_i$ y $v_i/t_i \in S$, en otro caso es v misma.
La sustitución correspondiente a la función identidad se denota por {}.
Si S es una sustitución de la forma $\{v_1/t_1,\ldots,v_n/t_n\}$ y e una expresión, entonces Se es una expresión que se obtiene de e mediante el remplazo simultáneo de cada ocurrencia de la variable v_i ($1\leq i\leq n$) en e por el término t_i. Se dice entonces que Se es una instancia de e.

Ejemplo Si S= {x/A, y/f(B), z/C} y e = p(x,y,z) entonces

$$Se = p(A,f(B),C)$$

Si $e_1,e_2,\ldots,e_m$ son expresiones y S es una sustitución se dice que S es un unificador de $e_1,\ldots,e_m$ si $Se_1 = Se_2 = \ldots = Se_m$. Se dice entonces que $e_1,\ldots,e_n$ son unificables.

Por ejemplo e_1= p(A,y) y e_2= p(x,f(B)) son unificables mediante la sustitucion unificadora {x/A, y/f(B)}.

Un unificador U de las expresiones $e_1,\ldots,e_m$ es un unificador más general si para cualquier unificador S de $e_1,\ldots,e_m$ existe una sustitución T tal que $S = U\circ T$.
Por ejemplo si e_1= p(z,f(z),f(u)) y e_2= p(A,x,f(g(y))) la sustitución {z/A, x/f(A), u/g(y)} es un unificador más general para e_1 y e_2.

Una asociación es una pareja ordenada con una variable como primer elemento, y un término como segundo elemento.

Un ambiente o estado es un conjunto de asociaciones cuyos primeros elementos no se repiten.

Si (v,t) pertenece a un ambiente E se dice que v está ligada a t en E.

El valor de una expresión e en un ambiente A, denotado $val_A(e)$, es el resultado de remplazar cada variable en e que esté ligada en A a un término t por el valor de t ($val_A(t)$) en A. Observe que si una variable no está ligada a ningún término en un ambiente A, su valor en A es ella misma.

Ejemplo: el valor de la expresión p(x,f(y,g(z,x))) en el estado

{(y,f(v,w)), (z,g(x,u)), (u,h(x))} es

p(x,f(f(v,w),g(g(x,h(x)),x)))

3. Interpretación procedimental (secuencial no-determinística).

Una interpretación procedimental secuencial no-determinística de un programa lógico expresado mediante cláusulas de Horn procesa en orden sus objetivos, pero selecciona sus cláusulas de una manera no-determinística.

Un programa lógico se activa mediante la consulta de una lista de objetivos iniciales digamos Q1,...,Qn y continua usando sus procedimientos para derivar listas de objetivos nuevas de viejas. Tal derivación de listas de objetivos se realiza mediante un proceso llamado resolución (que se interpreta como una llamada de procedimiento):

Se obtiene no-determinísticamente del programa una cláusula de la forma Q:- P1,...,Pm ($m \geq 0$) tal que Q unifique con Q1 (en el sentido de que algun unificador mas general S hace a Q y Q1 idénticos) y se deriva una nueva lista de objetivos de la forma

<S(P1),S(P2),...,S(Pm),S(Q2),...,S(Qn)>.

Este proceso se continua aplicando recursivamente con este nuevo objetivo hasta obtener un objetivo vacío.
Si durante la ejecución del proceso que se acaba de describir fuese imposible encontrar una regla que unifique con el primer átomo de cualquiera de los objetivos generados se dice que la ejecución falla y el programa termina con una respuesta negativa.

Observe que en este planteamiento no se considera la posibilidad de reintentar (backtrack) otras posibilidades después que la ejecución haya fallado.

El proceso secuencial de resoluciones de los objetivos con selección no-determinística de cláusulas unificables anteriormente descrito, corresponde en la programación tradicional a la noción de computación propuesta por Dijkstra[D76] con su lenguaje de comandos guardados, donde las condiciones (guardas) de un comando alternativo son seleccionadas no-determinísticamente. La decisión de optar por este tipo de interpretación se toma por la sencillez que se logra para el análisis de corrección total de estos programas. Como en el caso de la programación imperativa, estos programas se implantan facilmente en términos de programas con selección secuencial de reglas.

La noción de computación asociada corresponde a la sucesión de listas de objetivos $L_1, L_2, \ldots, L_n$ obtenida a partir de un programa lógico P y una lista inicial de objetivos L_1, donde cada L_{i+1} se obtiene de L_i mediante la resolución del primer objetivo de L_1 usando una regla de P (seleccionada no-determinísticamente).

El resultado de la computación es el valor final de las variables en la lista de objetivos original L_1 las cuales resultan afectadas por las sustituciones unificadoras más generales asociadas a las resoluciones realizadas.

Una lista de objetivos se trata como una pila de llamadas de procedimientos. El átomo seleccionado es aquél en el tope de la pila. Las nuevas llamadas de procedimiento que por el proceso de resolución remplazan la llamada de procedimiento seleccionada, se insertan en el tope de la pila para así obtener una nueva lista de objetivos.

Más formalmente, si se define una <u>situación</u> s como un par (O,A), donde O es una lista de objetivos y A un ambiente, entonces

Dados: un programa lógico P,

una situación <u>inicial</u> $s = (O,A)$, con $O = \langle Q \rangle$ tal que Q es un objetivo con variables en $\bar{v}$

El par (P,s) se denomina una <u>consulta</u> (simple).

Una <u>computación</u> C[P,Q,A] de la consulta (P,s) es una sucesión posiblemente infinita de situaciones

$$C[P,Q,A] = \langle s_0, s_1, \ldots \rangle$$

definida así:

a) $s_0 = s$

b) Para $k \geq 0$, si $s_k = (O_k, A_k)$, con $O_k = \langle Q_1, \ldots, Q_r \rangle$

Si r=0 : s_{k+1} no se define, y se dice que C[P,s] <u>termina</u> (exitosamente).
En este caso, se define el <u>resultado</u> de la computación como

$$C[P,O,A](\bar{v}) = val_{Ak}(\bar{v})$$

Si $r>0$: es posible escoger no-determinísticamente en P una regla $O:- R_1,\ldots,R_m$ tal que O y O_1 unifiquen mediante un unificador más general σ, se define

$s_{k+1} = (O_{k+1}, A_{k+1})$ así:

$O_{k+1} = \langle \sigma R_1,\ldots,\sigma R_m,\sigma O_2,\ldots,\sigma O_r\rangle$

$A_{k+1} = \sigma \circ A_k$

En otro caso no se define s_{k+1} y se dice que C[P,s] falla.

Ejemplo 3.1 El siguiente es un programa lógico para concatenar dos listas. El término cons(x,y) se interpreta como una lista cuyo primer elemento es x y cuya cola (resto de lista) es y (abreviado [x|y]). La constante Nil denota la lista vacía (o también []).

```
concat(Nil,w,w).
concat(cons(x,y),z,cons(x,u)):- concat(y,z,u).
```

concat(x,y,z) denota la relación: "z es la lista resultante de concatenar la lista x con la lista y".

Más claramente, usando las abreviaciones se tiene:

```
(1) concat([],w,w).
(2) concat([x|y],z,[x|u]):- concat(y,z,u).
```

Para calcular el resultado de concatenar la lista [2] a la lista [1] el programa se activa mediante la situación inicial s = (O,A) con O = <concat([1],[2],v)>

A = {}

$\bar{v}$ = v

Una computación asociada C podría ser ($s_k = (O_k, A_k)$)

O_0 = <concat([1],[2],v)>

A_0 = {}

Ahora : concat([1],[2],v) unifica con la cabeza de (2) mediante el unificador más general
σ_1 = {x/1, y/[], z/[2], v/[1|u]}

Entonces:

O_1 = <concat([],[2],u)>

$A_1 = \sigma_1 \circ \{\}$

Después: concat([],[2],u) unifica con la cabeza de (1) mediante el unificador más general σ_2 = {w/[2], u/[2]}

Así:

$O_2 = \square$ (donde $\square$ denota una secuencia nula de objetivos)

$A_2 = \sigma_2 \circ \sigma_1 \circ \{\}$

Termina y s_3 no se define.

El resultado de la computación es:

$$C(v) = val_{A2}(v) = [1|[2]]$$

4. Noción de corrección de un procedimiento lógico.

La corrección de un procedimiento lógico se expresará en términos de predicados de algún metalenguaje usado como lenguaje de especificación de entradas (datos) y salidas (resultados) y en general del estado de las variables del procedimiento durante su ejecución.

Si α es un predicado del metalenguaje de especificación que incluye unicamente variables en $\bar{v} = v_1,\ldots,v_n$ (denotado $\alpha(\bar{v})$) entonces

$$(\alpha \mid v_1,\ldots,v_n : t_1,\ldots,t_n)$$

denota el predicado que se obtiene de la sustitución simultanea de todas las ocurrencias de v_i en α por el término t_i. Si $\bar{t}$ denota la secuencia de términos $t_1,\ldots,t_n$ entonces se puede abreviar a $(\alpha \mid \bar{v}:\bar{t})$

Se entenderá por aserción un predicado sobre (los valores de) las variables de un ambiente dado.

Si P es un programa lógico y Q es un objetivo con variables $\bar{v}$ se dice que la consulta ?-Q es totalmente correcta con respecto al programa P, a una aserción de entrada (o precondición) y a una aserción de salida (o postcondición) $\beta(\bar{v})$ notado

$$\{\alpha\}\ (?-\ Q)/P\ \{\beta(\bar{v})\}$$

si y solo si $(\beta \mid \bar{v}:\ C[P,Q,A](\bar{v}))$ vale en todo ambiente A para el cual vale α, es decir β satisface el resultado de la computación C[P,Q,A] para todo ambiente A en el cual vale α.

Si un programa lógico P contiene un procedimiento lógico con identificador r cuya lista de parámetros $\bar{v}$ se puede dividir en párametros de datos $\bar{x}$ y de resultados $\bar{y}$ se dice que r es totalmente correcto con respecto a P, a una precondición $\alpha(\bar{x})$ y una postcondición $\beta(\bar{x},\bar{y})$ denotado

$$\{\alpha\}\ r(\bar{x},\bar{y})/P\ \{\beta\}$$

si y solo si para toda instanciación explícita $\bar{X}$ de $\bar{x}$ y todo vector de variables $\bar{z}$ se cumple

$$\{(\alpha|\bar{x}:\bar{X})\}\ (?\text{-}r(\bar{X},\bar{z}))/P\ \{(\beta|\bar{x},\bar{y}:\bar{X},\bar{z})\}$$

es decir toda consulta de un objetivo $r(\bar{X},\bar{z})$ es correcta con respecto a $(\alpha|\bar{x}:\bar{X})$ y $(\beta|\bar{x},\bar{y}:\bar{X},\bar{z})$ para toda secuencia $\bar{X}$ de términos explícitos y toda secuencia $\bar{z}$ de variables que correspondan respectivamente a $\bar{x}$ y $\bar{y}$.

5. Semántica operacional en términos de transición de estados.

De ahora en adelante se estudiará una clase especial de procedimientos lógicos: aquellos cuyas reglas recursivas son todas terminales.
Un procedimiento lógico se llama iterativo si sus reglas recursivas son terminales.
Es más nos interesan procedimientos iterativos (digamos) $p(\bar{x},\bar{y})$ para los cuales los parámetros en $\bar{x}$ son usados para transmitir datos y los de $\bar{y}$ para transmitir resultados. Consecuentemente se supondrá que toda consulta al procedimiento p es de la forma

$$?- p(\bar{X},\bar{z})$$

donde $\bar{X}$ es un vector de términos explícitos y $\bar{z}$ vector de variables. No se pierde mucha generalidad si por simplicidad se supone además que el último átomo de toda regla recursiva acerca de p es de la forma $p(\bar{t},\bar{w})$ con $\bar{w}$ conformado unicamente por variables.
Los procedimientos iterativos anteriormente descritos serán denominados especiales.
Con el propósito de dar una interpretación imperativa a un procedimiento lógico especial, se presentará en seguida una semántica operacional que interpretará tanto la consulta inicial del procedimiento como las fórmulas atómicas que conforman cada una de las cláusulas de su cuerpo, en forma de comandos o instrucciones de tipo imperativo. Los efectos de estos comandos se reflejarán en transformaciones del ambiente de ejecución, mediante asociaciones de términos a las variables y parámetros de dicho procedimiento, realizadas por comandos de asignación definidos a continuación.

Primeramente se definirán comandos básicos para alterar un estado o ambiente de ejecución.
Si v es una variable y t un término en el cual v no ocurre, la acción de ejecutar el comando

$$v := t$$

(asignación del término t a la variable v) en el estado E es la de añadir la asociación (v,t) al conjunto de asociaciones que conforman el estado E previa eliminación de cualquier asociación en E cuyo primer elemento sea v.

Se define también un comando de asignación especial

$$v :=$$

cuyo efecto es el de dejar "libre" (su valor es ella misma) a la variable v, su acción sobre el ambiente en que se ejecuta, es la de eliminar de éste cualquier asociación cuyo primer elemento sea v.

Similarmente, si $\bar{v}$ y $\bar{t}$ son respectivamente vectores de variables diferentes $v_1,\ldots,v_n$ y términos $t_1,\ldots t_n$ para los cuales v_i no aparece en t_i entonces

$$\bar{v} := \bar{t}$$

se interpreta como un comando de asignación simultánea, cuyo efecto al ser ejecutado en un estado E es el de añadir las asociaciones (v_i, t_i) al conjunto de asociaciones de E previa eliminación de cualquier asociación en E cuyo primer elemento sea alguno de los v_i.

La ejecución de un procedimiento P con identificador $p(\bar{x},\bar{y})$ perteneciente a un programa lógico PL, se puede entonces visualizar como un proceso iterativo con variables globales $\bar{x}$, $\bar{y}$, $\bar{y}_1$ que corresponden respectivamente a los parámetros de datos, de resultados, y de resultados locales (obtenidos después de "procesar" una cláusula seleccionada para resolver un objetivo) y que se inicia con la invocación de un objetivo de la forma

$$?- p(\bar{X},\bar{z})$$

(donde $\bar{X}$ es un vector de términos explícitos y $\bar{z}$ vector de variables donde se obtendrán los resultados de ejecución) cuyo efecto inmediato es la realización de la asignación simultánea

$$\bar{x},\bar{y} := \bar{X},\bar{y}_1$$

A continuación se selecciona no-deterministicamente una cláusula del cuerpo de P cuya cabeza unifique con el atomo $p(\bar{X},\bar{y}_1)$. Dicha unificación ocasiona instanciaciones de las variables en la cláusula escogida, así como de los parámetros en $\bar{y}_1$, (resultados locales que se van acumulando en los parámetros $\bar{y}$, los cuales guardarán los resultados definitivos) que alteran el estado de ejecución y pueden explicarse en términos de comandos de asignación de variables.

El anterior proceso de resolución lleva a una sucesión (posiblemente nula) de llamados a ejecución de procedimientos de PL "externos" a P (las debidas a los átomos presentes en la parte derecha de la cláusula seleccionada) la cual puede culminar con una nueva invocación (recursiva) de P, $p(\bar{H},\bar{w})$ que conduce a que $\bar{x}$ tome el valor de $\bar{H}$ y $\bar{y}_1$ juegue el papel de $\bar{w}$, después de lo cual se selecciona de nuevo una cláusula de P para ser resuelta con $p(\bar{H},\bar{w})$ continuando el proceso conforme al ciclo ya descrito.

En caso de que la sucesión de llamadas a ejecución de procedimientos externos a P no fuera sucedida por una invocación recursiva de P, y de que dichas llamadas fueran realizadas exitosamente tendríamos una terminación normal del procedimiento P cuyos resultados podrían apreciarse después de la asignación $\bar{z} := val(\bar{y})$, en los valores de las variables $z_1,\ldots,z_n$ correspondientes al ambiente final de ejecución.

La ejecución de P falla si durante alguna iteración del ciclo que se acaba de describir, no es posible realizar la unificación correspondiente, o cuando la llamada de alguno de los procedimientos externos falla. Una terminación anormal se inter-

preta como una respuesta negativa a la pregunta acerca de la existencia de variables $\bar{z}$ que cumplan una cierta especificación preestablecida para el objetivo ?- $p(\bar{X},\bar{z})$.

Dentro de este contexto la ejecución de un procedimiento lógico $p(\bar{x},\bar{y})$ (invocado usando a $\bar{x}$ como parámetros de datos y a $\bar{y}$ como parámetros de resultados) se puede describir por medio de las siguientes 3 operaciones básicas:

-Invocación (inicial o recursiva) de un objetivo correspondiente a un átomo con identificador p.

-Resolución de un átomo correspondiente a $p(\bar{x},\bar{y})$ con una cláusula del cuerpo de $p(\bar{x},\bar{y})$.

-Llamada de procedimiento lógico "externo" (i.e. distinto de $p(\bar{x},\bar{y})$).

Estas operaciones interactuan en un ciclo de máquina como el siguiente. Supongamos que el objetivo a satisfacer es $p(\bar{X},\bar{z})$ donde como antes, $\bar{X}$ es una sucesión de términos explícitos que corresponden a los datos de la consulta, y $\bar{z}$ (sucesión de variables), donde se obtienen sus resultados.

Paso 0. (Invocación inicial) Efectuar las asignaciones

$\bar{x},\bar{y} := \bar{X},\bar{y}_1$ /* en $\bar{y}$ se calculan los resultados finales del procedimiento */

Paso 1. (Resolución)

Si existe una cláusula $C \bullet p(\bar{u},\bar{v}) :- Q_1,\ldots,Q_m$ donde $\bar{u}$ es una secuencia de términos con variables $r_1,\ldots,r_e$ que se resuelve con $p(val(\bar{x}),\bar{y}_1)$
entonces para reflejar el efecto de la unificación de $p(\bar{u},\bar{v})$ y $p(val(\bar{x}),\bar{y}_1)$, efectúe las asignaciones

$\bar{r},\bar{y}_1 := \bar{s},\bar{v}$;
/* donde $\bar{s} = s_1,\ldots,s_e$ son los subtérminos de $val(\bar{x})$ correspondientes a los de $\bar{r}$ de acuerdo a la unificación realizada. */
si no falle.
/*el procedimiento falla, dando una respuesta negativa*/

Paso 2. (Llamada de procedimientos externos)

Ejecute en orden (comenzando por Q_1) los procedimientos Q_i cuyos símbolos de relación sean diferentes a p (la igualdad solo se puede dar con Q_m, el último átomo de C). Estas llamadas pueden afectar las variables en $\bar{v}$ y otras locales pertenecientes a la cláusula C. Si alguna de estas ejecuciones falla (termina con una respuesta negativa) falle.

Si todas estas ejecuciones terminan normalmente entonces
Si C es una regla recursiva (digamos $Q_m \bullet p(\bar{t},\bar{w})$) con $\bar{t}$ secuencia de términos y $\bar{w}$ una secuencia de variables (que normalmente ocurren en $\bar{u}$ y $\bar{v}$ respectivamente) entonces vaya a Paso 3.
sino vaya a Paso 4.
sino falle

Paso 3. (Invocación Recursiva)

Efectue
$\bar{x} := val(\bar{t})$; /* reinicia a $\bar{x}$ */
$\bar{y} := val(\bar{y})$ /* independiza $\bar{y}$ de variables intermedias*/
$\bar{y}_1 := \_$; /* limpia $\bar{y}_1$ */
$\bar{w} := \bar{y}_1$; $\bar{y} := val(\bar{y})$; /* sustituye a $\bar{w}$ por $\bar{y}_1$ en $\bar{y}$. (útil si alguna variable de $\bar{w}$ ocurre en $\bar{v}$)*/

Elimine del ambiente todas las asociaciones cuyo primer elemento es una variable local a la regla $p(\bar{u},\bar{v}) :- Q_1,\ldots,Q_m$.
Vaya al Paso 1.

Paso 4. (Terminación) /* se recibe en $\bar{z}$ los resultados de la ejecución */
$\bar{z} := val(\bar{y})$;
Limpie el ambiente (excepto por $\bar{z}$);
Termine.

Ejemplo 5.1 Dado el procedimiento lógico concat, especificado como en el ejemplo 4.1:

{$x_1 = X_1$, $x_2 = X_2$ y X_1, X_2 son listas} concat(x_1, x_2, y) {$y = X_1 * X_2$}

cuyo cuerpo está conformado por las siguientes cláusulas

(a) concat([],u,u).
(b) concat([p|t],u,[p|w]):- concat(t,u,w).

y el siguiente objetivo:

?- concat([A],X_2,z).

donde X_2 representa una lista cuya descripción exacta no tiene relevancia en este contexto, se explicará la obtención en z de la lista resultante de la concatenación de [A] y X_2 por medio del ciclo de máquina descrito anteriormente.

Claramente $\bar{x}$ corresponde en este caso a las variables x_1 y x_2, así como $\bar{X}$ a los términos explícitos [A] y X_2, y $\bar{z}$ a la variable z entonces

en el paso 0
$\bar{x}, \bar{y} := \bar{X}, \bar{y}_1$ corresponde a la instrucción $x_1, x_2, y := [A], X_2, y_1$ llegandose al estado

(x_1 , [A]) (x_2 , X_2) (y , y_1);

en el paso 1 se resuelve el objetivo concat([A],X_2,y_1) con la cláusula (b) dando lugar a la unificación de $\bar{u}$ con val($\bar{x}$) y de val($\bar{y}$) con $\bar{v}$ es decir, de [p|t],u con [A],X_2 y de y_1 con [p|w]; ésto queda reflejado en la asignación

$p, t, u, y_1 := A, [], X_2, [p|w]$ que corresponde a $\bar{r}, \bar{y} := \bar{s}, \bar{v}$ en el ciclo de máquina; el estado ahora es

(x_1 , [A]) (x_2 , X_2) (p , A) (u , X_2) (t , [])
(y_1 , [p|w]) (y_2, y_1)

ahora $\bar{y} :=$ val($\bar{y}$) corresponde a y:= [A|w] el ambiente se actualiza conforme a este cambio:

(x_1 , [A]) (x_2 , X_2) (p , A) (u , X_2) (t , [])
(y_1 , [p|w]) (y , [A|w]) ;

el paso 2 corresponde aquí a una acción nula puesto que concat no invoca procedimientos externos, el ambiente por tanto permanece igual; las instrucciones ejecutadas en el paso 3 son

$y_1 :=$;
$x_1, x_2 :=$ [], X_2;
w:= $\bar{y}_1$;
y:= [A|y_1];

más la eliminación del ambiente de las asociaciones de variables diferentes de x_1, x_2, y. El estado resultante es entonces

$$(x_1 , []) \quad (x_2 , X_2) \quad (y , [A|y_1])$$

volviendo ahora al Paso 1 la cláusula (a) (concat([],u,u)) se resuelve con concat([],X_2,y_1) con las siguientes asignaciones:

$\bar{r},\bar{y}_1 := \bar{s},\bar{v}$	que corresponde a	$u,y_1 := X_2,u$
$\bar{y} := val(\bar{y})$	que corresponde a	$y := [A\|X_2]$

el estado ahora es

$$(x_1 , []) \quad (x_2 , X_2) \quad (u , X_2) \quad (y_1 , u)$$
$$(y , [A|X_2])$$

siendo la cláusula (a) un hecho, la ejecución termina en el Paso 4 que deja en z el valor $[A|X_2]$; el ambiente final es entonces,

$$(z, [A|X_2])$$

6. Cálculo inferencial de la corrección de un procedimiento.

El numeral anterior ilustra la posibilidad de dar una explicación en términos de transición de estados de la ejecución de un procedimiento de un programa lógico. Esto brinda la oportunidad no solo de definir la corrección de un procedimiento sino también de anotar su cuerpo mediante aserciones lógicas.

Por ejemplo una regla recursiva terminal de un procedimiento de un programa lógico podría especificarse de la siguiente manera:

$$\{\alpha\}\ p(\bar{u},\bar{v}) :- Q_1,\ldots,Q_{m-1},p(\bar{t},\bar{w})\ \{\beta\}$$

donde α y β son predicados lógicos de especificación, $\bar{u}$, $\bar{v}$, $\bar{t}$ son secuencias de términos y $\bar{w}$ es una secuencia de variables de la misma longitud que $\bar{v}$; $\bar{u}$ y $\bar{t}$ son de la misma longitud, y normalmente $\bar{t}$ ocurre en $\bar{u}$.

La interpretación procedimental de esta especificación sería: si el ambiente de ejecución del procedimiento lógico P cumple con el predicado α exactamente antes de resolver dicha regla con un objetivo cuyo primer átomo unifica con $p(\bar{u},\bar{v})$, entonces el procedimiento p ejecuta normalmente hasta el momento en que los objetivos creados por $Q_1,\ldots,Q_{m-1}$ ya han sido satisfechos y se debe satisfacer el objetivo correspondiente a la llamada recursiva $p(\bar{t},\bar{w})$, llegandose después de lo cual, a un estado que cumple con lo estipulado por el predicado β.

Más aún, es posible anotar cada una de las cláusulas de Horn de un procedimiento (digamos $q :- r_1,\ldots,r_s$) de la siguiente forma

$$\{\alpha\}\ q :- \{\gamma_1\}\ r_1\ \{\gamma_2\} \ldots \{\gamma_s\}\ r_s\ \{\beta\}$$

Se podría como se hizo arriba, dar definiciones procedimentales de estas anotaciones en términos de la semántica operacional de las operaciones básicas (invocación, resolución y llamada de procedimiento lógico externo) con que se explicó imperativamente la ejecución de un procedimiento.

Sinembargo, es preferible explicar la noción de corrección en términos de una semántica axiomática que se apoye en un conjuntc de axiomas (en realidad esquemas de axiomas) y reglas de inferencia en el estilo del cálculo desarrollado por Hoare [H69].

Las reglas de inferencia se describirán como

$$\frac{\alpha_1}{\beta} \qquad o \qquad \frac{\alpha_1 \ \wedge \ \alpha_2}{\beta}$$

donde α_1 y α_2 son los <u>antecedentes</u> (condiciones bajo las cuales la regla de inferencia es aplicable) y ß es el <u>consecuente</u> (expresión a ser deducida). Cada antecedente es o una especificación de un comando (previamente demostrada) o una expresión del lenguaje de especificación que se debe demostrar separadamente como un lema.

Se usarán como reglas de inferencia las mismas usadas por Hoare:

Reglas de Consecuencia

D1
$$\frac{\{\alpha\}\ C\ \{\beta\} \ \wedge \ (\beta \dashrightarrow \gamma)}{\{\alpha\}\ C\ \{\gamma\}}$$

D2
$$\frac{\{\gamma\}\ C\ \{\beta\} \ \wedge \ (\alpha \dashrightarrow \gamma)}{\{\alpha\}\ C\ \{\beta\}}$$

donde C es cualquiera de los tres comandos básicos ya mencionados.

La secuenciación de comandos es un método de composición básico en cualquier lenguaje de programación, y que en este artículo solo ha sido usado implicitamente, será denotada separando los comandos mediante un símbolo de "punto y coma" (;).

Regla de Composición

D3
$$\frac{\{\alpha\}\ C_1\ \{\alpha_1\} \ \wedge \ \{\alpha_1\}\ C_2\ \{\beta\}}{\{\alpha\}\ C_1;C_2\ \{\beta\}}$$

Los axiomas en que se basará cualquier deducción serán los siguientes:

A0 Axioma de invocación inicial

Si al invocar un predicado especial $p(\bar{x},\bar{y})$, mediante la consulta $?\text{-}p(\bar{X},\bar{z})$, vale $(\alpha|\bar{x}:\bar{X})$ entonces se llega a un estado que cumple $\alpha \wedge \bar{y}=\bar{y}_1 \wedge \bar{x}=\bar{X}$. O sea

$$\{(\alpha|\bar{x}:\bar{X})\}\ \text{"invocación inicial de } ?\text{-}p(\bar{X},\bar{z})\text{"}\ \{\alpha \wedge \bar{y}=\bar{y}_1 \wedge \bar{x}=\bar{X}\}$$

A1 Axioma de Resolución

Si durante la ejecución de un procedimiento lógico p, corresponde realizar la resolución de ún átomo con identificador p con una cláusula C de la forma

$$p(\bar{u},\bar{v}) :- Q_1,\ldots Q_m$$

(para la cual la secuencia $Q_1,\ldots Q_m$ puede ser vacía), entonces

si dicho estado cumple el predicado α y además existe una sustitución Θ y un predicado ß tal que

(1) $\alpha \dashrightarrow$ ("$\bar{u}$ y val($\bar{x}$) unifican via Θ como unificador más general")

y

(2) $(\alpha \wedge \bar{y}_1 = \text{val}(\Theta\bar{v})) \dashrightarrow$ ß

después de realizar la resolución se llega a un estado para el cual vale ß. Es decir

$$\{\alpha\}\ \text{"resolución de objetivo con cláusula C"}\ \{ß\}$$

A2 Axioma de Procedimiento Externo

Si durante la ejecución de un procedimiento p de un programa lógico PL, se está en un estado para el cual un átomo con identificador r distinto de p, que corresponde a un procedimiento de PL distinto de p, es el primero en la lista de objetivos, y cumple con la especificación

$$\{\alpha\}\ r(\bar{m},\bar{n})\ \{ß\}$$

entonces, si dicho estado cumple $\mathcal{J}$ y $(\alpha|\bar{m}:\bar{M})$, (donde $\mathcal{J}$ es un predicado en el cual no ocurren las variables en $\bar{z}$, excepto talvez en igualdades de variables libres) una vez concluida la llamada al procedimiento r, se llega a un estado para el cual vale $\mathcal{J}$ y $(ß|\bar{m},\bar{n}:\bar{M},\bar{z})$. Es decir

$$\{\mathcal{J} \wedge (\alpha|\bar{m}:\bar{M})\}$$
$$\text{"llamada de procedimiento externo } r(\bar{M},\bar{z})\text{"}$$
$$\{\mathcal{J} \wedge (ß|\bar{m},\bar{n}:\bar{M},\bar{z})\}$$

A3 Axioma de invocación recursiva

Si durante la ejecución de un procedimiento lógico p, un átomo $p(\bar{H},\bar{w})$ que no corresponde a la invocación inicial del procedimiento P es el primero en la lista de objetivos, entonces si α vale y $(\alpha \dashrightarrow (ß|\bar{x},\bar{y}_1:\bar{H},\bar{w}))$ es cierto para un cierto predicado ß, se tiene que después de la invocación se llega a un estado

que cumple ß. Es decir

$$\{\alpha\}\ \text{"invocación recursiva de la regla terminal RT"}\ \{ß\}$$

A4 Axioma de Terminación

Si durante la ejecución del procedimiento lógico $p(\bar{x},\bar{y})$ se llega a un estado para el cual, la lista de objetivos es vacía, y vale el predicado $(\alpha\,|\,\bar{z}:\bar{y})$ entonces, la ejecución de dicho procedimiento termina en un estado que cumple α :

$$\{(\alpha\,|\,\bar{z}:\bar{y})\}\ \text{"terminación"}\ \{\alpha\}$$

Con estos axiomas y reglas de inferencia es posible demostrar la corrección de procedimientos lógicos especiales con respecto a una especificación dada; sin embargo, el nivel de detalle de estas deducciones resultaría poco claro y tedioso. Con el propósito de agilizar y hacer más intuitivo este proceso deductivo se presentará un axioma adicional, deducible de los anteriores pero útil para calcular corrección de aserciones sobre reglas recursivas terminales que no invocan procedimientos externos, y un teorema cuya aplicación permitirá demostrar la corrección de un procedimiento lógico especial con respecto a una especificación.

A5 Axioma adicional

Si durante la ejecución de un procedimiento lógico p, se está en un estado inmediatamente anterior a la resolución del primer átomo del objetivo actual con una regla de la forma

$$p(\bar{u},\bar{v}) :- p(\bar{t},\bar{w})$$

es decir una regla recursiva terminal que no incluye átomos correspondientes a procedimientos diferentes a p, entonces

si dicho estado satisface el predicado α, θ es el unificador más general para $\bar{u}$ y val($\bar{x}$) y existe un predicado ß tal que

$$(\alpha \cdot \bar{y}_1 = \bar{v}) \dashrightarrow (ß\,|\,\bar{x},\bar{y}_1 : \theta\bar{t},\bar{w})$$

después de realizadas la resolución y la invocación recursiva se llega a un estado que cumple ß. Es decir

$$\{\alpha\}\ p(\bar{u},\bar{v}) :- p(\bar{t},\bar{w})\ \{ß\}$$

claramente este axioma combina los efectos estipulados por los axiomas A1 y A3.

<u>Teorema</u> 6.1 Suponga que en un programa lógico P, $p(\bar{x},\bar{y})$ es un procedimiento lógico especial con cláusulas no recursivas $CNP_1,\ldots,CNR_e$ y clausulas recursivas terminales $CR_1,\ldots,CR_f$ cuyo cuerpo responde a la siguientes especificaciones:

$\{\gamma_1 \wedge \mu\}\ CNR_1\ \{(\beta|\bar{x}:\bar{X})\}$

$\vdots$

$\{\gamma_e \wedge \mu\}\ CNR_e\ \{(\beta|\bar{x}:\bar{X})\}$

$\{\varepsilon_1 \wedge \mu\}\ CR_1\ \{\mu\}$

$\vdots$

$\{\varepsilon_f \wedge \mu\}\ CR_f\ \{\mu\}$

para ciertos predicados lógicos $\mu, \gamma_1, \ldots, \gamma_e, \varepsilon_1, \ldots, \varepsilon_f$ y si además

(1) $(\alpha \wedge \bar{x}=\bar{X} \wedge \bar{y}=\bar{y}_1) \Longrightarrow \mu$

(2) $\mu \Longrightarrow (\bigvee_{i=1}^{e} \gamma_i \circ \bigvee_{j=1}^{f} \varepsilon_j)$

(3) $\{\alpha\}\ p(\bar{x},\bar{y})\ \{T\}$ (T es el predicado universal)

entonces $\{(\alpha|\bar{x}:\bar{X})\}\ (?- p(\bar{X},\bar{z}))/P\ \{(\beta|\bar{x},\bar{y}:\bar{X},\bar{z})\}$

vale para toda sucesión de términos explícitos $\bar{X}$ y por tanto

$$\{\alpha\}\ p(\bar{x},\bar{y})/P\ \{\beta\}$$

μ se llama un <u>invariante</u> del procedimiento especial $p(\bar{x},\bar{y})$.

<u>Demostración</u>:

Del axioma A0 se obtiene

$$\{(\alpha|\bar{x}:\bar{X})\}\ \text{"invocación inicial"}\ \{\alpha \wedge \bar{y}=\bar{y}_1 \wedge \bar{x}=\bar{X}\}$$

entonces por (1) y la regla D1 se tiene

$$\{(\alpha|\bar{x}:\bar{X})\}\ \text{"invocación inicial de ?- } p(\bar{X},\bar{z})\text{"}\ \{\mu\}.$$

Ahora bien, del hecho anterior y del axioma (2) se deduce que exactamente después de la invocación inicial se presenta una de las siguientes dos situaciones:

(a) valen μ y algún ε_j, y el objetivo puede resolverse con una regla recursiva no terminal CR_j;

(b) valen μ y algún γ_i y el objetivo se resuelve con la regla no recursiva CNR_i.

En el primer caso (situación (a)) despues de ejecutarse las llamadas de procedimiento en CR_j se llega a un estado para el cual vale μ (de acuerdo a la especificación de CR_j); y entonces, debido a (2) uno de los predicados γ_i o ε_j vale, volviendo de nuevo a presentarse una de las situaciones (a) o (b). La hipótesis (3) garantiza que la ejecución del procedimiento termina normalmente, y por tanto eventualmente se llegará a un estado para el cual vale uno de los predicados $\gamma_1, \ldots, \gamma_e$ caso que corresponde a la situación (b).

En el segundo caso, después de ejecutarse las llamadas de procedimientos en en la cláusula CNR_i, se llega a un estado para el cual, la lista de objetivos es vacía y vale $(\beta|\bar{x}:\bar{X})$ por axioma A4 (puesto que $((\beta|\bar{x},\bar{y}:\bar{X},\bar{z})|\bar{z}:\bar{y}) = (\beta|\bar{x}:\bar{X})$); la ejecución del procedimiento termina entonces en un estado que cumple $(\beta|\bar{x},\bar{y}:\bar{X},\bar{z})$. Como $\bar{X}$ es una lista de términos explícitos arbitrarios

$$\{\alpha\}\ p(\bar{x},\bar{y})/P\ \{\beta\}$$

vale. ∎

7. Ejemplos ilustrativos y aplicaciones.

En esta sección se presentarán dos ejemplos que muestran como desarrollar el cálculo de corrección presentado en este artículo de una manera ágil y efectiva.

Primeramente se mostrará la corrección del procedimiento lógico concat estudiado ya en los ejemplos 3.1, 4.1 y 5.1.

Ejemplo 7.1

El cuerpo del procedimiento concat puede anotarse así:

C1 $\{\mu \wedge (x_1=[])\}$ concat([],u,u) $\{y=X_1 * X_2\}$

C2 $\{\mu \wedge (x_1 \neq [])\}$ concat([p|t],u,[p|w]):- concat(t,u,w) $\{\mu\}$

siendo

$\mu \equiv [x_1, x_2, X_1, X_2$ son listas $\wedge$ $x_2=X_2 \wedge$

$(\exists 1$: 1 es lista $\wedge$ $1 * x_1 = X_1 \wedge y = 1 * y_1)]$

Usando el cálculo desarrollado en la sección 6 y con ayuda del teorema allí demostrado se probará que

$\{\alpha$: x_1, x_2 son listas$\}$ concat(x_1, x_2, y) $\{\beta$: $y = x_1 * x_2\}$

Dentro de la terminología de dicho teorema se tienen las siguientes correspondencias:

e=1 y $\gamma_1 \equiv (x=[])$, f=1 y $\varepsilon_1 \equiv (x \neq [])$ también

(1) corresponde a

$(x_1, x_2$ son listas $\wedge$ $x_1=X_1 \wedge x_2=X_2 \wedge y=y_1) \Rightarrow \mu$
y su validez es clara.

(2) corresponde a

$[x_1, x_2, X_1, X_2$ son listas $\wedge$ $(\exists 1$: 1 lista $\wedge$ $1 * x_1 = X_1 \wedge y = 1 * y_1)] \wedge$

$\wedge$ $(x_2=X_2) \Rightarrow (x_1=[]$ o $x_1 \neq [])$ que también vale.

(3) La terminación del procedimiento es intuitivamente clara pues el número de inferencias realizadas corresponde a la longitud de la lista X_1.

Se define ahora $\alpha_1 \equiv \mu \wedge (x_1=[])$ y $\alpha_2 \equiv \mu \wedge (x_1 \neq [])$ para mostrar que el código está bien anotado conforme al teorema 1.

Prueba de C1: (Se usa axioma 1) Suponga que α_1 vale. Sea θ la sustitución $\{(u, X_2)\}$ entonces val(x_1), val(x_2) (es decir [], X_2) unifican con $\bar{u}$ (ó sea [],u), con θ como unificador más general además

además

$$\alpha_1 \cdot y_1 = \theta u \equiv \mu \cdot x_1 = [] \cdot y_1 = X_2$$

$$\equiv (\exists l{:}l \text{ es lista} \cdot y = l * y_1 \cdot l * x_1 = X_1) \cdot x_2 = X_2 \cdot x_1 = [] \cdot y_1 = X_2$$

$$\Rightarrow y = X_1 * X_2 \quad (\text{porque } l = X_1)$$

y la prueba sigue del axioma A1

Prueba de C2: Suponga que α_2 vale, como [p|t] y u unifican respectivamente con val(x_1) y val(x_2) mediante la sustitución

$$\theta \equiv \{(p, car(x_1)), (t, cdr(x_1)), (u, X_2)\}$$

entonces por el axioma A5 basta mostrar que

$\alpha_2 \cdot y_1 = \theta[p|w] \Rightarrow \mu | x_1, x_2, y_1 : \theta t, \theta u, w)$ es decir

$[x_1, x_2, X_1, X_2$ son listas $\cdot x_2 = X_2 \cdot y_1 = [car(x_1)|w] \cdot x_1 \neq [] \cdot$

$(\exists l{:}l$ es lista $\cdot l * x_1 = X_1 \cdot y = l * y_1)] \Rightarrow [cdr(x_1), X_1, X_2$ son

listas $\cdot X_2 = X_2 \cdot (\exists l_1{:}l_1$ es lista $\cdot l_1 * cdr(x_1) = X_1 \cdot y = l_1 * w)]$

que se verifica facilmente.

Sobre este ejemplo se puede anotar lo siguiente:

(1) Aunque la semántica provista por la lógica clausal constituye en este caso, una explicación clara y natural del procedimiento en términos recursivos, no deja de ser interesante la explicación puramente iterativa que se obtiene con la interpretación en términos de transición de estados aquí desarrollada.

(2) Del cálculo detallado de corrección que se acaba de mostrar, es posible extraer una explicación iterativa más intuitiva y manejable:

Sabiendo que las reglas del procedimiento "concat" cumplen las siguientes especificaciones

C1 $\{\mu \cdot (x_1 = [])\}$ concat([],u,u) $\{y = X_1 * X_2\}$

C2 $\{\mu \cdot (x_1 \neq [])\}$ concat([p|t],u,[p|w]) :- concat(t,u,w) $\cdot \{\mu\}$

Se puede concluir que concat cumple:

$\{\alpha : x_1, x_2$ son listas$\}$ concat(x_1, x_2, y) $\{\beta : y = x_1 * x_2\}$

puesto que el resultado del teorema 6.1 vale si se cumplen las siguientes condiciones:

i) De la precondición α de concat debe poderse deducir que el invariante μ vale. Lo cual es evidente.

ii) Si el invariante μ y la condición x_1=[] valen entonces el procedimiento termina su ejecución mediante la aplicación del hecho concat([],u,u) cumpliendo con la postcondición $y=X_1*X_2$. Esto se deduce de que el efecto de la unificación con dicho hecho presupone que x_1=[] y que y_1 tome el valor de x_2.

iii) Si en algún estado de la ejecución del procedimiento valen el invariante μ y la condición x_1 es una lista diferente de [], entonces la regla en C2 es aplicable lo que hace que tanto x_1 como y_1 se actualicen respectivamente con los valores obtenidos de la aplicación de la funciónes cdr y cons (con car(x_1)) sobre sus valores previos. Las alteraciones del ambiente que se acaban de describir, hacen que después de la aplicación de la regla en cuestión valga de nuevo el invariante μ, lo que es facilmente verificable.

iv) La terminación del procedimiento está garantizada por los efectos producidos por la regla en C2 sobre el parámetro x_1, que hacen que su valor sea eventualmente igual a la lista vacía ([]).

A continuación se presenta un procedimiento para el cual la explicación de su corrección en términos de la semántica de la lógica clausal no es evidente, debido a su naturaleza iterativa. En contraste la interpretación en términos de transición de estados que propone este artículo da una explicación clara de su corrección, si se usa el modo informal de presentación que se ilustró al final del ejemplo anterior.

Ejemplo 7.2 Se desea demostrar que el procedimiento reverse(x,y) perteneciente al siguiente programa lógico

```
L:  reverse(u,v):- rev(u,[],v).
    rev([],u,u).
    rev([u1,u2],u3,v):- rev(u2,[u1|u3],v).
```

cumple la siguiente especificación:

{α: x es lista} reverse(x,y)/L {ß: $y=x^R$ (versión invertida de x)}

Basta probar que para todo termino explicito X y toda variable z se cumple

(*) {x es lista . x=X} ?-reverse(X,z)/L {$z=X^R$}

suponga entonces, que la siguiente especificación del procedimiento rev vale

(=) {α1: x_1,x_2 listas} rev(x_1,x_2,y)/L {ß1: $y=x_1^R*x_2$}

como el cuerpo del procedimiento rev consta de una sola regla, por el axioma A0 la prueba de (*) se reduce entonces a mostrar

(**) {x es lista ∧ x=X ∧ y=y_1} reverse(u,v):- rev(u,[],v) {y=X_R}

Para ello considere la sustitución θ = {(u,X)} para la cual u y val(x) unifican con como unificador más general. El axioma A1 garantiza que el estado en que se invoca a rev como procedimiento externo cumple el predicado

u es lista ∧ (y=y_1 ∧ y_1=v ∧ u=X)

luego por el axioma A2 después de realizada la invocación rev(u,[],v) vale

v=u^R ∧ (y=y_1 ∧ y_1=v ∧ u=X) de donde se deduce que y = X^R con lo que (**) queda demostrado.

Para demostrar la suposición hecha en (¤) se anotarán las reglas del cuerpo de rev de la siguiente forma

B1 {θ : x_1=[] ∧ μ} rev([],u,u).{y=X_1^R*X_2}

B2 {ξ:x_1 es lista≠[] ∧ }rev([u1|u2],u3,v):- rev(u2,[u1|u3],v){μ}

donde μ ≡ [x_1, x_2 son listas ∧ (x_2^R*x_1 = X_2^R*X_1) ∧ y=y_1]

Ahora bien, por teorema 6.1 es suficiente verificar las siguientes condiciones:

(i) La precondición α1, y=y_1 y el hecho de que x_1=X_1 y x_2=X_2 son ciertas debe implicar μ, lo cual es evidente.

(ii) Si el invariante μ y la condición x_1=[] valen, el procedimiento termina su ejecución con la aplicación del hecho rev([],u,u) cumpliendo con la postcondición y=X_2^R*X_1. Esto se infiere de que la resolución con tal hecho presupone x_1=[] y que y_1 tome el valor de x_2 (que unifica con u) y por tanto se obtiene

x_2^R*[] = x_2^R*X_1 ∧ y_1=x_2 ∧ y=y_1 de donde se deduce que

y^R = X_2^R*X_1 y así y = X_1^R*X_2.

(iii) Si en algún estado de ejecución del procedimiento valen el invariante μ y la condición x_1 es lista diferente de [] entonces la regla en B2 es aplicable lo que ocasiona que tanto x_1 como x_2 se actualicen respectivamente con los valores $cdr(x_1)^2$ cons(car(x_1),x_2). Estas alteraciones preservan la validez del invariante μ después de la aplicación de la regla.

(iv) La terminación del procedimiento está garantizada por los efectos producidos por la regla B2 sobre el parámetro x_1 que hacen que su valor sea eventualmente igual a [].

Conclusiones.

La programación lógica fué concebida por Kowalski como un lenguaje de programación con una semántica natural y auto descriptiva. Sin embargo, PROLOG el lenguaje más popular que pretende implantar estas ideas, dista mucho de poseer las cualidades propuestas por Kowalski, en parte debido a los comandos y predicados adicionales que se le añadieron con el fin de hacerlo más eficiente y práctico [CM 84], pero también debido al estilo usado por los programadores profesionales, con el fin de escribir programas eficientes. Como ya se indicó en la introducción de este artículo la búsqueda de eficiencia se traduce en perdida de claridad para el programa, imponiendose de este modo, la necesidad del desarrollo de una noción precisa de corrección y de un cálculo para ésta.

En este trabajo se intentó dar un primer paso en esa dirección definiendo la noción de corrección de un programa lógico en términos de una interpretación de su ejecución mediante un esquema de transición de estados. La explicación de dicha interpretación no resultó del todo natural ni clara. Posteriormente se presentará una segunda versión de este artículo que supere dichas deficiencias.

Agradecimientos. El autor desea reconocer la influencia y consejo del profesor Rodrigo Cardoso en algunos aspectos del tratamiento de este artículo, sin embargo sobra anotar que los errores u omisiones que pudieran encontrarse son entera responsabilidad del primero.

Es importante también destacar la eficiente labor del Centro de Documentación del instituto CIFI de la facultad de Ingeniería de la universidad de Los Andes, en la persona de su director Jairo Mora sin cuya ayuda y soporte técnico en la consecución de la biliografía necesaria este artículo nunca se habría escrito.

BIBLIOGRAFIA

[C73] Colmerauer A. et al, Un system de Comunication Homme-machine en Francaise. Rapport Groupe d'Intelligence Artificielle, Universite d'Aix Marseille, Luminy, 1973.

[CM84] Cloksin W.F., Mellish C.S., Programming in Prolog, 2nd ed. Springer Verlag, New York, 1984.

[D76] Dijkstra E.W., A Discipline of Programming, Prentice-Hall Inc, Englewood Cliffs, New Jersey, 1976.

[H69] Hoare C.A.R., An Axiomatic Basis for Computer Programming. Comm. ACM 12 (10), pp 576-583, Octubre 1976.

[K74] Kowalski R.A., Predicate Logic as a Programming Language. Proc. IFIP 74, North Holland Publishing Co., Amsterdam, pp 569-574, 1974.

[K79] Kowalski R.A., Logic for Problem Solving, Elsevier North Holland, New york, 1979.

[R75] Roussel P., PROLOG: Manuel de Reference et d'Utilisation, Groupe d'Intelligence Artificielle, Universite d'Aix-Marseille, Luminy, Sept 1975.

[W77] Warren D.H.D., Pereira L.M., Pereira F., PROLOG The Language and its Implementation compared with LISP. Proc. Symp. on Artificial Intelligence and Programming Languages, SIGPLAN Notes, vol 12 No 8 and SIGART Newsletters No 64, pp 109-115, August 1977.