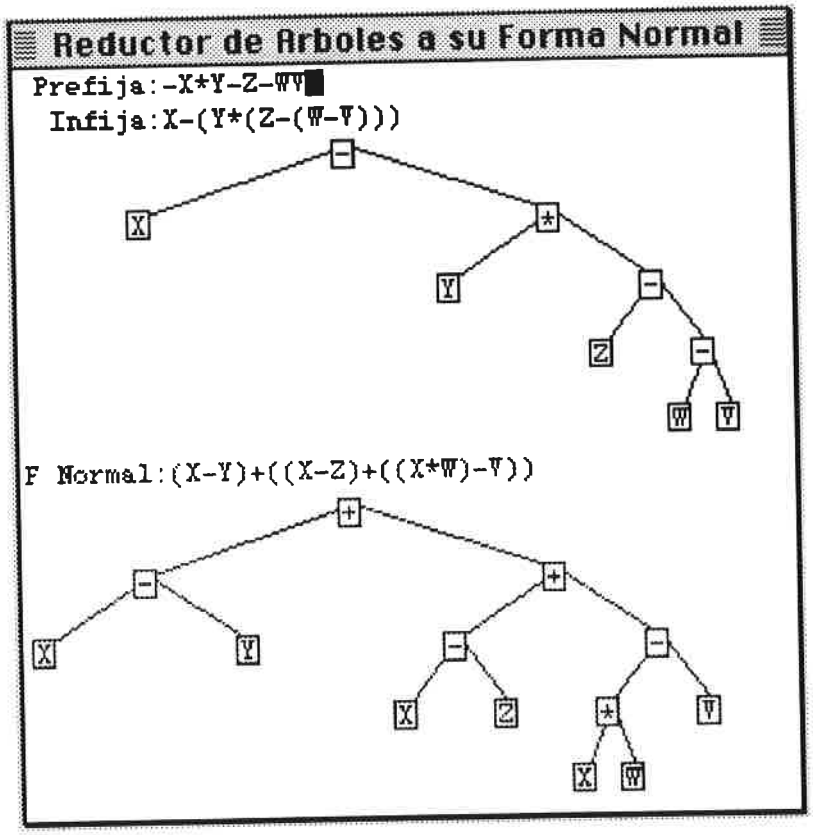
Conversión de un árbol CSG a su forma normal disyuntiva. Implementación de una versión gráfica e interactiva.

Antonio Aguilera

Report LSI–97–5–T

Conversión de un árbol CSG a su Forma Normal Disyuntiva.



Implementación de una versión gráfica e interactiva.

Antonio Aguilera.

Julio de 1996

1.- Introducción.

En este trabajo se describe un método para transformar cualquier árbol CSG a otro equivalente que es la unión de subárboles más simples [GHF86], [GMF89]. La transformación produce un árbol nuevo, una *Forma Normal* que tiene las siguientes propiedades:

a) El operando derecho de cualquier intersección o diferencia será siempre una primitiva.

b) El operando izquierdo de cualquier intersección o diferencia será siempre, o bien una primitiva, o bien, otra intersección o diferencia (nunca una unión).

En este trabajo, el algoritmo de conversión ha sido implementado en forma gráfica e interactiva.

2.- Arbol CSG y Forma Normal Disyuntiva.

CSG son la iniciales para la expresión inglesa de **Geometría Constructiva de Sólidos** (Constructive Solid Geometry). La cual es un modelo de definición de sólidos. El modelo CSG representa a los sólidos como el resultado de aplicar operaciones booleanas y movimientos rígidos a un conjunto de instanciaciones de sólidos más simples llamadas primitivas [Män88].

La forma más natural de poder representar un modelo CSG es precisamente mediante un *árbol CSG* que según la notación BNF se puede definir recursivamente como [Män88]:

<árbol CSG> ::= <primitiva> |
<árbol CSG><operación booleana><árbol CSG> |
<árbol CSG><movimiento rígido>

donde:

<primitiva> es una instaciación de un sólido primitivo representado mediante un identificador del tipo de primitiva y una secuencia de parámetros de dimensión.

<movimiento rígido> es una translación o una rotación y

<operación booleana> es una unión ($\cup$), una intersección ($\cap$) o una diferencia (-).

Así, las primitivas se representan en las hojas del árbol CSG, mientras que los nodos internos representan, o bien operaciones booleanas, o bien movimientos rígidos [Män88]. La evaluación de un árbol CSG da lugar a un sólido, que incluso puede llegar a ser muy complejo.

Por otra parte, un árbol CSG en Forma Normal Disyuntiva (FND) se puede expresar como:

$$A_1 \cup A_2 \cup \ldots \cup A_n$$

con $A_j = x_1 \otimes x_2 \otimes \ldots \otimes x_k$ para cada j en 1...n.

donde $\otimes$ representa una intersección ($\cap$) o una diferencia (-).

Un expresión en Forma Normal es pues, lo que se conoce como UoI (Union of Intersections). Todo árbol CSG se puede normalizar mediante un proceso que se basa en la aplicación de equivalencias lógicas conocidas hasta llevarlo a su forma normal. Este proceso se detalla en la siguiente sección.

3.- Formas englobantes.

Una forma de simplificar los cálculos en el modelo CSG es trabajar con formas aproximadoras [ref5]. Disponer de una aproximación del objeto permite agilizar todas las clasificaciones, y por tanto, todos los procesos sobre el modelo CSG.

Las formas englobantes usadas más ampliamente son los paralelepípedos (o cajas).

En [ref8] se demuestra que la caja englobante de un objeto, representado por un árbol CSG, depende de la expresión en la que el árbol esté dado, y que la caja mínima se consigue cuando el árbol está expresado en su forma normal disyuntiva.

4.- Algoritmo de normalización.

El Algoritmo de normalización utiliza el siguiente conjunto de equivalencias:

1. $X-(Y \cup Z) = (X-Y)-Z$

2. $X \cap (Y \cup Z) = (X \cap Y) \cup (X \cap Z)$

3. $X-(Y \cap Z) = (X-Y) \cup (X-Z)$

4. $X \cap (Y \cap Z) = (X \cap Y) \cap Z$

5. $X-(Y-Z) = (X-Y) \cup (X \cap Z)$

6. $X \cap (Y-Z) = (X \cap Y)-Z$

7. $(X \cup Y)-Z = (X-Z) \cup (Y-Z)$

8. $(X \cup Y) \cap Z = (X \cap Z) \cup (Y \cap Z)$

Estas equivalencias encapsulan las propiedades asociativas y distributivas, y fueron elegidas porque representan a todas las configuraciones no normalizadas posibles para un nodo. Las equivalencias 2, 3, 5, 7 y 8 descomponen una expresión en sumas, mientras que las equivalencias restantes (1, 4 y 6) reemplazan la asociación derecha con asociación izquierda. Las demostraciones de las equivalencias se proporcionan en el Apéndice A.

El algoritmo de normalización utilizado es [GMF89]:

```
Procedure Normalize(T:tree);
{reduce a CSG tree to sum-of products}
Begin
   If T = PRIMITIVE Then Return;
   Repeat
      While (T matches left side of any set equivalence)
         replace with right side using equivalences 1-6
         before using 7 or 8;
      Normalize (T.left);
   Until (T=∪) Or ((T.right=PRIMITIVE ) And (T.left≠∪));
   Normalize (T.right);
End;
```

Algoritmo 1. Conversión de un árbol CSG a Forma Normal.

Esta versión del algoritmo [GMF89] corrige un error en una presentación anterior del mismo en [GHF86]. Por otro lado, esta versión no es el único algoritmo de normalización posible ya que hay muchas maneras de convertir una expresión Booleana a Forma Normal Disyuntiva [GMF89].

Finalmente, el algoritmo 1 presentado, tiene las siguientes propiedades:

1. Dado cualquier árbol CSG de entrada, el algoritmo termina.

2. Al terminar, el algoritmo deja el árbol CSG en Forma Normal.

3. En cada paso de reestructuración, el algoritmo sólo requiere de información local (tipo del nodo y tipo de sus nodos hijos).

4. Si el árbol inicial no contiene subárboles redundantes ni primitivas repetidas, el algoritmo 1 no agregará términos redundantes ni primitivas repetidas dentro de un producto.

5.- Implementación

Para realizar la implementación del algoritmo se escribieron procedimientos que realizan las reestructuraciones del árbol CSG de acuerdo a las ocho equivalencias anteriores. Sin embargo sólo se escribieron tres procedimientos (y no ocho) basándose en la observación de que las equivalencias podían ser representadas por tres modelos de equivalencias distintos.

Se observó que las equivalencias 1, 4 y 6 son isomorfas entre sí en la estructura de sus subárboles, tanto en el miembro izquierdo como en el derecho. El modelo 1 representa a estas tres equivalencias (Fig 1). Similarmente el modelo 2 representa a las equivalencias 2, 3 y 5 (Fig 2); y el modelo 3 a las 7 y 8 (Fig 3).

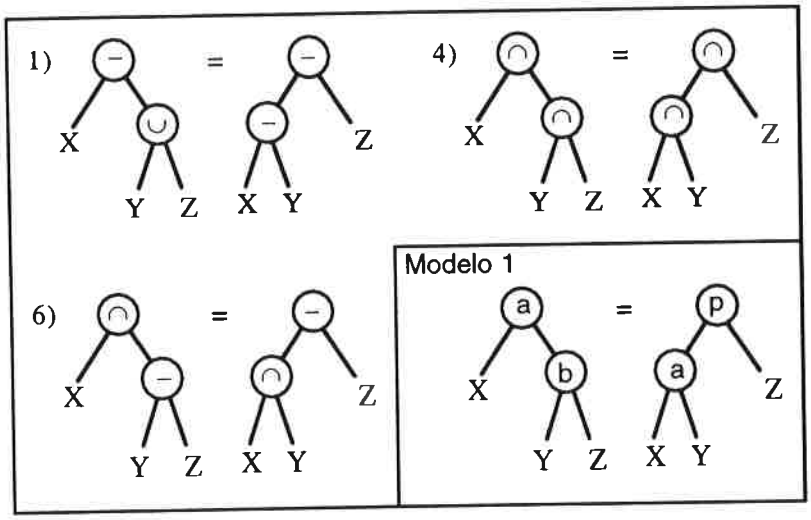


Figura 1. Equivalencias 1, 4 y 6, representadas por el Modelo 1.

Las diferencias se resuelven mediante las variables **a**, **b** y **p**. Así pues en el modelo 1 (figura 1) la variable **a** representa a la operación booleana en el nodo raíz del subárbol considerado en cada paso antes de la normalización, la cual quedará en el nodo del hijo izquierdo después de la normalización. Por su parte, la variable **b** representa a la operación del nodo en el hijo derecho del nodo raíz del subárbol considerado, operación que, sin importar cuál sea, será ignorada pues no aparece en la estructura resultante después de la normalización. Finalmente la operación resultante en el nodo raíz es de diferencia para las equivalencias 1 y 6; y de intersección para el caso de la equivalencia 4 (ver figura 1); de cualquier forma, esta operación resultante se especifica mediante el parámetro **p** que se le pasa a la rutina a la hora de la ejecución.

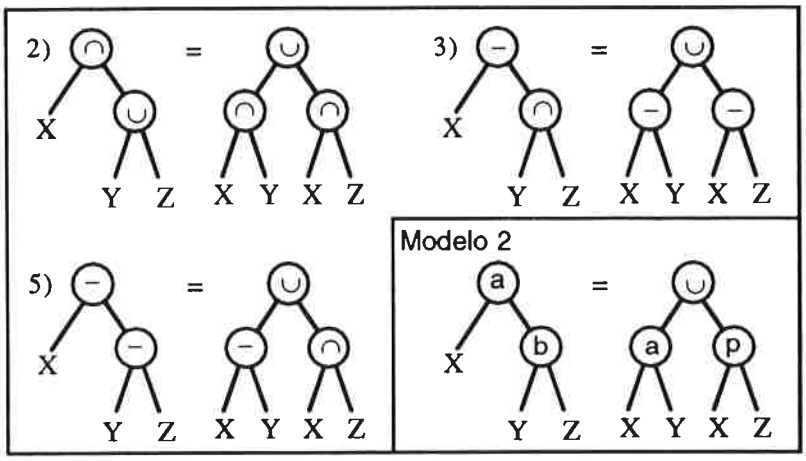


Figura 2. Equivalencias 2, 3 y 5, representadas por el Modelo 2.

Las variables **a**, **b** y **p** tienen un papel análogo en los modelos 2 y 3. Sin embargo cabe hacer notar que en estos dos modelos, la operación resultante en el nodo raíz es de unión, por lo que trata como constante.

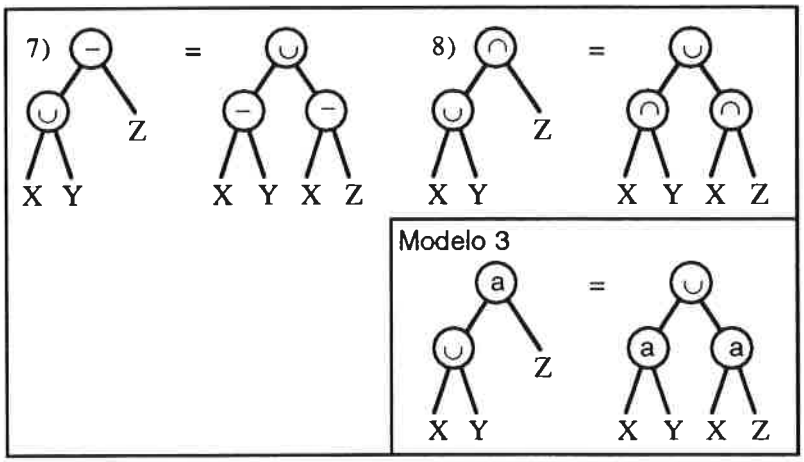


Figura 3. Equivalencias 7 y 8, representadas por el Modelo 3.

	Xa(YbZ) = (XaY)pZ	Modelo 1:		
1.	X-(Y∪Z) = (X-Y)-Z	a=(-)	b=(∪)	p=(-)
4.	X∩(Y∩Z) = (X∩Y)∩Z	a=(∩)	b=(∩)	p=(∩)
6.	X∩(Y-Z) = (X∩Y)-Z	a=(∩)	b=(-)	p=(-)
	Xa(YbZ) = (XaY)∪(XpZ)	Modelo 2:		
2.	X∩(Y∪Z) = (X∩Y)∪(X∩Z)	a=(∩)	b=(∪)	p=(∩)
3.	X-(Y∩Z) = (X-Y)∪(X-Z)	a=(-)	b=(∩)	p=(-)
5.	X-(Y-Z) = (X-Y)∪(X∩Z)	a=(-)	b=(-)	p=(∩)
	(X∪Y)aZ = (XaZ)∪(YaZ)	Modelo 3:		
7.	(X∪Y)-Z = (X-Z)∪(Y-Z)	a=(-)		
8.	(X∪Y)∩Z = (X∩Z)∪(Y∩Z)	a=(∩)		

Figura 4. Valores de **a**, **b** y **p** para cada equivalencia y por modelo.

La figura 4 sumariza en forma textual a los tres modelos y muestra los valores de **a**, **b** y **p** para cada equivalencia.

Esta implementación está diseñada para recibir una cadena de caracteres que representa el árbol CSG a normalizar escrito en notación prefija.

De la cadena recibida se genera, mediante una sencilla rutina recursiva (algoritmo 2), la estructura de un árbol binario estándar (es decir cada nodo tiene tres campos: Info, Liga izquierda<L> y Liga derecha<R>) donde se almacena el árbol CSG a normalizar. En el algoritmo 2, `PreStr` es la cadena en prefijo; `strPos` es es un apuntador (inicializado en cero) a la posición del caracter en proceso dentro de la cadena.

```
Procedure CreaArbol (Var treePos: Integer);
   Var ch: Char;
Begin
   strPos := Succ(strPos);
   If strPos <= length(PreStr)
      Then ch := PreStr[strPos]
      Else ch := '?';  {si faltan operandos, sustituirlos por "?"}
   NewNodo(treePos);
   If treePos > 0 Then
      With Nodo[treePos] Do
         Begin
            Info := ch;
            L := 0;
            R := 0;
            If Info In ['+', '-', '*'] Then
               Begin
                  CreaArbol(L);
                  CreaArbol(R);
               End;
         End;
End;
```

Algoritmo 2. Creación del árbol a partir de la notación prefija.

Posteriormente se genera la notación infija del árbol CSG a normalizar mediante un recorrido inorden del árbol, y también se genera una representación gráfica del mismo árbol CSG mediante un recorrido en preorden.

Inmediatamente después se procede a normalizar el árbol mediante el algoritmo 3, obtenido directamente del algoritmo 1, en donde el cuerpo de la estructura repetitiva `while` es ahora una llamada a la acción `Reestructura`, la cual se presenta en el algoritmo 4.

```
Procedure Normalize(T:tree);
{reduce a CSG tree to sum-of products}
Begin
   If T = PRIMITIVE Then Return;
   Repeat
      While (T matches left side of any set equivalence)
         Reestructura(T);  {véase algoritmo 4}
      Normalize (T.left);
   Until (T=∪) Or ((T.right=PRIMITIVE ) And (T.left≠∪));
   Normalize (T.right);
End;
```

Algoritmo 3. Conversión de un árbol CSG a Forma Normal.

El algoritmo 4 muestra el proceso de reestructuración de un nodo. La acción `Reestructura` analiza la información local del nodo y la de sus hijos, para identificar, en su caso, la necesidad de aplicar alguna de las ocho equivalencias; primero la equivalencia 1, despues la 2,... etc. y al último la equivalencia 8; que es el orden en que deben ser analizadas [GMF89].

Si existe la necesidad de aplicar alguna equivalencia, ésta se aplicará mediante la invocación de alguno de los tres modelos de reestructuración, aquel que le corresponda, recordando que el modelo 1 maneja a las equivalencias 1, 4 y 6; el modelo 2 maneja a las equivalencias 2, 3 y 5; y el modelo 3 a las 7 y 8 (Fig 4).

El algoritmo 4 muestra también los procedimientos `Modelo1`, `Modelo2` y `Modelo3`, que son los que aplican las equivalencias según el modelo respectivo. El segundo argumento de los procedimientos `Modelo1` y `Modelo2` es el mismo parámetro **p** que se muestra en las figuras 1, 2 y 4, y que no es obtenible del modelo, por lo que se le debe suministrar a la rutina en cuestión. Por su parte, los modelos 2 y 3 duplican al subárbol X (ver figuras 2 y 3), proceso que se lleva a cabo por la acción recursiva copia, mostrada también en el algoritmo 4.

Finalmente se genera la notación infija del árbol CSG ya normalizado mediante un recorrido inorden del árbol resultante, y también se genera una representación gráfica del mismo árbol CSG mediante un recorrido en preorden, con el fin de compararlos con la versión sin normalizar (véase la figura de la portada).

```
Function Copia (t: Integer): Integer;
  Var d: Integer;
Begin
   If t = 0 Then
      Copia := 0
   Else
      Begin
         NewNodo(d);
         With Nodo[d] Do
            Begin
               Info := Nodo[t].Info;
               L := Copia(Nodo[t].L);
               R := Copia(Nodo[t].R);
            End;
         Copia := d;
      End;
End;

Procedure Modelo1(T:Integer; p:Char);
Var x: Integer; {subárbol X}
Begin
   With Nodo[T] Do
      Begin
         x := L;
         L := R;
         R := Nodo[L].R;
         Nodo[L].R := Nodo[L].L;
         Nodo[L].L := x;
         Nodo[L].Info := Info;
         Info := p;
         Cambio := True;
      End;
End;

Procedure Reestructura (T: Integer);
    Begin
       With Nodo[T] Do

          If (Info = '-') & (Nodo[R].Info = '+') Then
             Modelo1(T, '-')                { 1: X-(Y+Z) -> (X-Y)-Z      }

          Else If (Info = '*') & (Nodo[R].Info = '+') Then
             Modelo2(T, '*')                { 2: X*(Y+Z) -> (X*Y)+(X*Z)  }

          Else If (Info = '-') & (Nodo[R].Info = '*') Then
             Modelo2(T, '-')                { 3: X-(Y*Z) -> (X-Y)+(X-Z)  }

          Else If (Info = '*') & (Nodo[R].Info = '*') Then
             Modelo1(T, '*')                { 4: X*(Y*Z) -> (X*Y)*Z      }

          Else If (Info = '-') & (Nodo[R].Info = '-') Then
             Modelo2(T, '*')                { 5: X-(Y-Z) -> (X-Y)+(X*Z)  }

          Else If (Info = '*') & (Nodo[R].Info = '-') Then
             Modelo1(T, '-')                { 6: X*(Y-Z) -> (X*Y)-Z      }

          Else If (Info = '-') & (Nodo[L].Info = '+') Then
             Modelo3(T)                     { 7: (X+Y)-Z -> (X-Z)+(Y-Z)  }

          Else If (Info = '*') & (Nodo[L].Info = '+') Then
             Modelo3(T)                     { 8: (X+Y)*Z -> (X*Z)+(Y*Z)  }
    End;

Procedure Modelo2(T:Integer; p:Char);
Var x: Integer; {subárbol X}
Begin
   With Nodo[T] Do
      Begin
         x := L;
         NewNodo(L);
         Nodo[L].Info := Info;
         Nodo[L].L := x;
         Nodo[L].R := Nodo[R].L;
         Nodo[R].L := Copia(x);
         Nodo[R].Info := p;
         Info := '+';
         Cambio := True;
      End;
End;

Procedure Modelo3 (T: Integer);
Var z: Integer; {subárbol Z}
Begin
   With Nodo[T] Do
      Begin
         z := R;
         NewNodo(R);
         Nodo[R].Info := Info;
         Nodo[R].L := Nodo[L].R;
         Nodo[R].R := z;
         Nodo[L].Info := Info;
         Nodo[L].R := Copia(z);
         Info := '+';
         Cambio := True;
      End;
End;
```

Algoritmo 4. Procedimiento de reestructura y sus rutinas auxiliares: Modelo1, Modelo2, Modelo 3 y Copia.

En esta implementación todos los procesos anteriores se realizan por cada caracter que se introduzca en la cadena, de modo que el programa es totalmente interactivo y permite observar el resultado de la normalización en cada instante.

Lo anterior, junto con el hecho de que una expresión escrita en notación prefija no contiene paréntesis, y por lo tanto no hacía falta escibir un parser elaborado, han sido las razones por las cuales se tomó la decisión de que la cadena debía ingresarse usando dicha notación.

6.- Un ejemplo interactivo.

La figura 5 muestra tres estados de una sesión interactiva en la que se ingresa la expresión: $(A \cup B) \cap (C \cup D)$.

Notas: 1.- En esta implementación, se utilizan los signos "+" y "*" para representar a las operaciones unión e intersección respectivamente.
2.- El ingreso de la expresión se hace en notación prefija.

De acuerdo a las notas anteriores la expresión: $(A \cup B) \cap (C \cup D)$ se ingresa en este programa como: *+AB+CD.

La figura 5.a se muestra el estado cuando se han ingresado los primeros cuatro caracteres "*+AB" de la expresión prefija (en el primer renglón), los cuales no forman una expresión completa (válida), pues falta el segundo operando de la intersección(*). Esto se refleja con la presencia del cursor (rectángulo sólido negro) a la derecha de la "B" (el último caracter ingresado), también se refleja con la presencia de un cursor gráfico en el nodo del hijo derecho de la raíz del árbol CSG.

Sin embargo ya se ha realizado la conversión, el segundo operando faltante se simboliza con un signo de interrogación(?), generado por el algoritmo 2 y que aparece gráficamente en cuatro posiciones:

1. en la expresión en notación infija (segundo renglón);
2. en el nodo del árbol al que le corresponde el operando faltante;
3. en la expresión resultante del proceso de conversión a la Forma Normal. Cabe señalar que en esta expresión hay dos signos de interrogación ya que en el proceso de conversión se aplicó la distribución de la intersección sobre la unión;
4. en el árbol normalizado, en el nodo al que le corresponde el operando faltante (que también en este caso son dos).

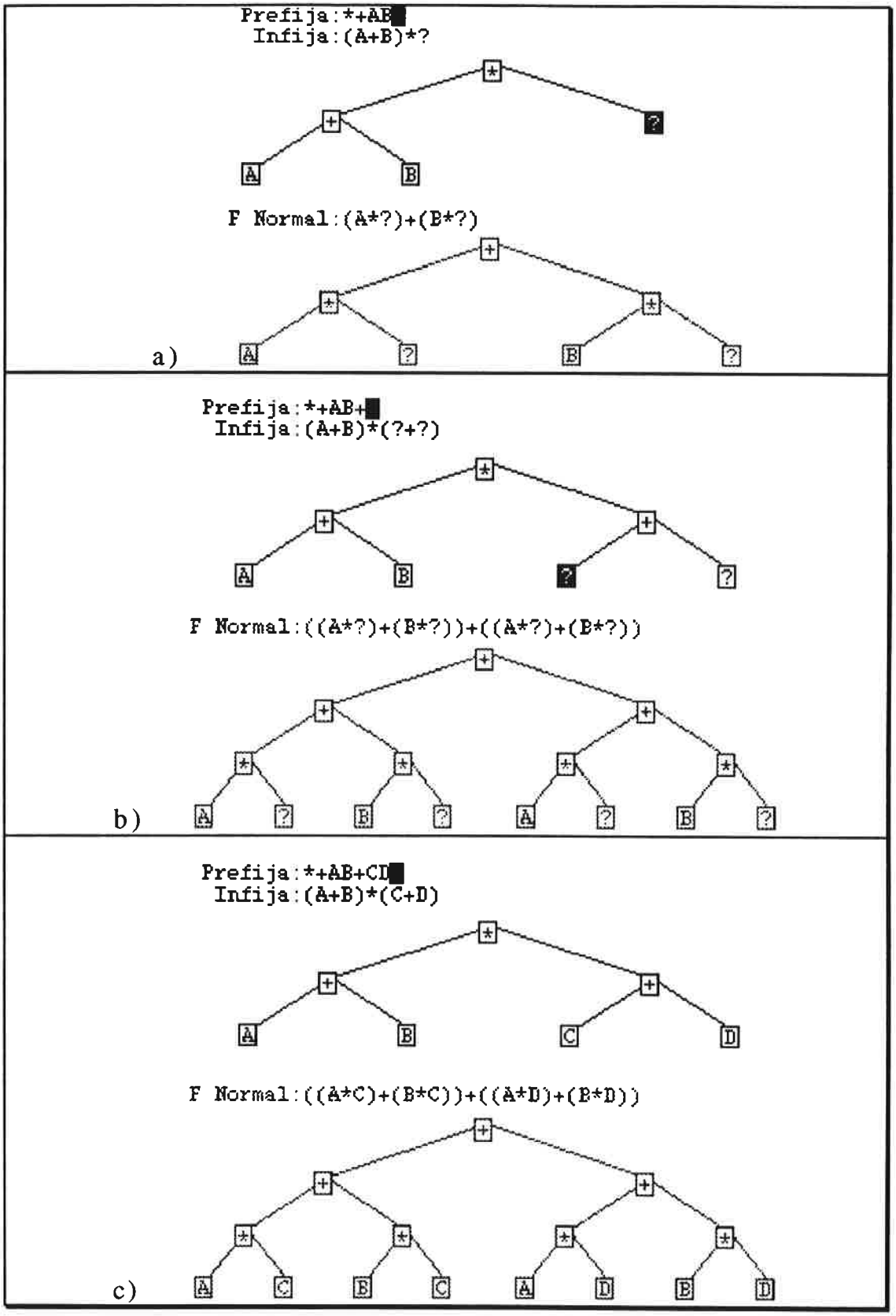


Figura 5. Proceso interactivo de la expresión prefija *+AB+CD.
a) se ha ingresado parcialmente la cadena hasta "*+AB".
b) se ingresa el "+".
c) se termina de ingresar la cadena.

La figura 5b muestra el estado cuando se ingresa el "+" (el siguiente caracter de la expresión) y se puede apreciar que la estructura del árbol resultante cambia para reflejar la nueva (y definitiva) normalización.

Aquí cabe hacer notar tres hechos: que en el nodo del árbol donde antes estaba el cursor gráfico se coloca el "+" recién ingresado; que se despliegan dos nodos nuevos para simbolizar los operandos de esta unión; y que cada uno lleva un "?" para simbolizar los dos operandos faltantes.

Aunque ahora faltan dos operandos, ya se conoce la nueva estructura de la expresión y se reconoce que para normalizarla se deben aplicar dos veces la ley distributiva, razón por la que ahora aparecen cuatro "?"´s, dos para cada operando faltante, y aparecen tanto en la expresión en infija como en el árbol que la representa.

Finalmente, en la figura 5c se muestra el estado de la sesión al momento de finalizar el ingreso de la cadena a procesar. Se han ingresado los caracteres "C" y "D", los cuales simplemente vienen a sustituir a los correspondientes signos de interrogación que los representaban.

7.- Conclusiones.

El algoritmo demostró cumplir su objetivo de convertir un árbol CSG en Forma Normal Disyuntiva en forma eficiente.

Por otro lado, este algoritmo, vestido de una implementación gráfica e interactiva, tiene atractivos potenciales didácticos.

8.- Bibliografía

[GHF86] Goldfeather, J.; Hultquist, J. and Fuchs, H. *Fast Constructive Solid Geometry in the Pixel-Powers Graphics System.* ACM Computer Graphics Proc. SIGGRAPH, (20):4. 1986.

[GMF89] Goldfeather, J.; Molnar, S. and Fuchs, H. *Near Real-Time CSG Rendering using Tree Normalization and Geometric Pruning.* IEEE Computer Graphics & Applications. 1989.

[Hof89] Hoffmann, C. *Geometric and Solid Modeling.* Morgan Kauffmann Publishers, Inc., 1989.

[MaC94] Mazzetti, M. and Ciminiera, L. *Computing CSG-tree boundaris as algebraic expressions.* CAD, 26(6):417-425, 1994.

[Män88] Mäntylä, Martti. *An Introduction to Solid Modeling.* Computer Science Press. 1988.

Apéndice A

Demostración de las equivalencias.

1. $X-(Y\cup Z) = (X-Y)-Z$

$$\begin{aligned} X-(Y\cup Z) &= X\cap(Y\cup Z)' \\ &= X\cap(Y'\cap Z') \\ &= (X\cap Y')\cap Z' \\ &= (X-Y)-Z \end{aligned}$$

2. $X\cap(Y\cup Z) = (X\cap Y)\cup(X\cap Z)$

Ley distributiva.

3. $X-(Y\cap Z) = (X-Y)\cup(X-Z)$

$$\begin{aligned} X-(Y\cap Z) &= X\cap(Y\cap Z)' \\ &= X\cap(Y'\cup Z') \\ &= (X\cap Y')\cup(X\cap Z') \\ &= (X-Y)\cup(X-Z) \end{aligned}$$

4. $X\cap(Y\cap Z) = (X\cap Y)\cap Z$

Ley asociativa.

5. $X-(Y-Z) = (X-Y)\cup(X\cap Z)$

$$\begin{aligned} X-(Y-Z) &= X-(Y\cap Z') \\ &= X\cap(Y\cap Z')' \\ &= X\cap(Y'\cup Z) \\ &= (X\cap Y')\cup(X\cap Z) \\ &= (X-Y)\cup(X\cap Z) \end{aligned}$$

6. $X\cap(Y-Z) = (X\cap Y)-Z$

$$\begin{aligned} X\cap(Y-Z) &= X\cap(Y\cap Z') \\ &= (X\cap Y)\cap Z' \\ &= (X\cap Y)-Z \end{aligned}$$

7. $(X\cup Y)-Z = (X-Z)\cup(Y-Z)$

$$\begin{aligned} (X\cup Y)-Z &= (X\cup Y)\cap Z' \\ &= (X\cap Z')\cup(Y\cap Z') \\ &= (X-Z)\cup(Y-Z) \end{aligned}$$

8. $(X\cup Y)\cap Z = (X\cap Z)\cup(Y\cap Z)$

Ley distributiva.