Anales PANEL'81/12 JAIIO
Sociedad Argentina de informática
e Investigación Operativa. Buenos Aires, 1981

UN SISTEMA DE PROGRAMACION AUTOMATICA

Leopoldo H. Carranza

RESUMEN

Un sistema de programación automática capaz de interpretar la descripción de un problema y sintetizar automáticamente un programa que lo resuelve es descripto brevemente.

Un lenguaje de especificaciones basado en el cálculo de predicados de la lógica matemática propuesto como interface con el usuario es detallado y su aplicación al manejo de base de datos es comentado.

Algunos aspectos teóricos sobre el poder expresivo del lenguaje y problemas de computabilidad son analizados informalmente.

UN SISTEMA DE PROGRAMACION AUTOMATICA

INTRODUCCION

Un sistema de programación automática capaz de producir automáticamente programas para resolver una clase restringida de problemas, es descripto brevemente.

A diferencia del enfoque habitual este sistema no requiere de los programas para probar teoremas, en cambio es una extensión del sistema de manejo de / base de datos.

La programación automática es la aplicación del computador a la producción de sus propios programas, esto es: usar el computador para mecanizar las tareas de programación. En cierto sentido es software para producir software.

El objetivo final es implementar un sistema capaz de entender los requerimientos del usuario, analizar el problema y usando su conocimiento sobre el tema, obtener el método de solución, codificar y optimizar el programa necesario.

Este objetivo está aún lejos de ser alcanzado y quedan aún muchas dificultades que vencer.

Una dificultad es que en general el problema puede ser recursivamente insoluble, esto es: no computable, o sea que no exista un programa que lo resuelva.

Sin embargo es posible aproximarse a este objetivo final.
La idea es restringir el universo del discurso, reduciendo la clase de problemas a aquellos que se puede asegurar que existe un programa que los resuelve y se conoce un método para obtener ese programa.

El sistema que aquí se describe se basa en limitar la clase de problemas a tratar restringiendo el lenguaje al universo de una base de datos.

En un trabajo anterior del autor se explicaban bajo otro punto de vista algunas de estas ideas.

El sistema propuesto está básicamente constituído por un compilador (de hecho un compilador de compilador) capaz de traducir el lenguaje de especificación al FORTRAN y se diferencia de otros tales como el de Z.Manna y R.Waldinger en que sus objetivos son más modestos como para no requerir que los programas para probar teoremas y es en cambio similar al Model II pero difiere en su concepción y en particular es más expresivo que los lenguajes de alto nivel de manejo de base de datos como el INGRES o el SQL (que no permiten obtener la clausura de una relación) y a diferencia de otros lenguajes como el SETL no requiere como primitivas las funciones de agregado.

Este sistema está parcialmente implementado y solo algunos componentes fueron probados. El proyecto fue encarado con fines de investigación y como parte de las tareas académicas del autor.

En lo que sigue se explican algunos conceptos básicos y se dan algunas definiciones preliminares y fundamentos teóricos, se describe el lenguaje de especificaciones y se comentan brevemente algunas de sus características más importantes.

Se omite toda referencia a los problemas de implementación, eficiencia, optimización, etc., así como toda referencia a la bibliografía clásica.

Se advierte que la exposición no es rigurosa ni formal.

CONCEPTOS BASICOS

Un sistema de programación automática es un sistema capaz de interpretar la descripción de un problema y sintetizar automáticamente un programa adecuado para resolver el problema planteado.

La descripción del problema puede realizarse a dos niveles: en términos de requerimientos o de especificaciones.

Los requerimientos son las condiciones que deben cumplir los resultados en función de los datos de entrada pero sin especificar el método de solución. No es posible diseñar un sistema para resolver cualquier problema dados los requerimientos ya que puede no existir un algoritmo de solución.

Las especificaciones en cambio son las condiciones que deben cumplir los resultados en función de los datos de entrada incluyendo la indicación del método de solución adoptado.

El objetivo es obtener un sistema que a partir de las especificaciones genere un programa cuyo output satisfaga las especificaciones.

La idea es expresar las especificaciones en el lenguaje del cálculo de predicados de la lógica matemática. Las condiciones que debe cumplir el output en función del imput se pueden expresar con una fórmula lógica: La solución es buscar los valores que hacen verdadera fórmula, esto es: satisfacer las especificaciones.

Este lenguaje de especificaciones puede considerarse como un lenguaje de programación de muy alto nivel, en el que solo se indican las características generales que debe cumplir la solución y omitiendo detalles sobre el proceso del cálculo permitiendo con pocas líneas de código resolver problemas complejos.

En camio, los lenguajes de programación tradicionales obligan al programador a codificar una serie de detalles del proceso de cálculo lo que constituye una tarea tediosa y sujeta al error.

Se estima que para una amplia gama de aplicaciones el uso de un lenguaje de especificaciones de este tipo puede incrementar la productividad de los programadores en una magnitud apreciable.

Otra idea interesante es considerar las bases de datos como única estructura de datos.

El programa generado por el sistema toma como imput una base de datos y produce como output la misma base de datos pero modificada. El usuario ingresa sus datos a la base y obtiene sus resultados consultando la base modificada, de esta forma el usuario solo debe aprender el lenguaje de interfase con la base de datos.

El programador solo necesita conocer el lenguaje de especificaciones (y un poco la lógica) para describir las condiciones que debe cumplir la base de datos modificada en función de la inicial.

En este sentido el lenguaje de especificaciones es también un lenguaje de manejo de base de datos.

El modelo de base de datos usado es el modelo relacional que asimila el conjunto de valores almacenados en la base a una relación en el sentido matemático.

El esquema de la base de datos provee los predicados básicos con los que se construyen las especificaciones.

Un predicado básico es un atributo ó nombre de relación del esquema de la base de datos y para un valor dado será verdadero si y solo si dicho valor aparece en la base, de lo contrario es falso. Las especificaciones se construyen como una fórmula bien formada a partir de los predicados básicos usando conectores y cuantificadores.

El programa producido por el sistema se limita a recorrer la base de datos (tantas veces como sea necesario) hasta encontrar el conjunto de valores que cumplen las especificaciones. Estos valores son, a su vez, almacenados en la base de datos.

Pero las fórmulas que expresan las especificaciones no pueden ser arbitrarias, es necesario introducir algunas restricciones. Las variables deben estar restringidas a un universo finito: solo pueden tomar valores entre los contenidos en la base o estar expresadas en función de estos valores.

Estas restricciones son necesarias para asegurar que el problema sea computable y reducirlo fundamentalmente a una simple búsqueda en la base de datos.

Estas ideas y otras han sido expuestas por el autor en otro trabajo, ya mencionado, en el que se hace énfasis en los aspectos teóricos de las bases de datos.

FUNDAMENTOS

Sea el lenguaje del cálculo de predicados L= (Lsim, Var, Oper, Pred) donde Lsimb. Var, Oper y Pred son conjuntos de símbolos lógicos (conectores y cuantificadores) variables, operadores y predicados respectivamente.

Con estos elementos definimos la sintáxis y semántica del lenguaje en la forma habitual.

SINTAXIS

Un término es una variable o expresión de la forma F (t1,t2,...,tn) donde F es un operador n-ádico y t1, t2,...,tn son términos.
Una constante es un operador cero-ádico.
Un predicado es una expresión de la forma P (t1,t2,...,tn) donde P es un símbolo de predicado n-ario y t1,t2,...,tn son términos o es de la forma (AvB), (A$\wedge$B),($\neg$A), (A=>B), (A$\Leftrightarrow$B), ($\forall$v,A), ($\exists$v,A) donde AyB predicados y v una variable.

SEMANTICA

Sea U= (Univ,Fun,Rel) una estructura donde Univ es un conjunto de elementos (universo del discurso), Fun un conjunto de funciones y Rel un conjunto de relaciones definidas entre los elementos de Univ.

Una interpretación o modelo del lenguaje L en la estructura U es una correspondencia que asigna a cada variable como rango el universo y a cada operador una función y a cada símbolo de predicado una relación.-

Si v es una variable la interpretación le asigna algún vi en Univ.-

Si F es un operador n-ádico la interpretación le asigna una función fi; $(Univ)^n \rightarrow Univ$.

Si P es un símbolo de predicado n-ario, la interpretación le asigna una relación $Ri \in (Univ)^n$.-

Si F (t1,t2,...,tn) es un término su denotación se define por:

$$i\,(f(t1,t2,\ldots,tn)) = fi(i(t1),\ i\,(t2),\ldots,i\,(tn))$$

(si f es 0-ádico, fi es una constante)

Si P (t1,t2,...,tn) es un predicado su denotación es:

$$i(\ P(t1,t2,\ldots,tn) = (i(t1),\ i(t2),\ldots,i(tn)) \in Ri$$

(Si P es 0-ario, su interpretación es un valor de verdad)

Si A y B son predicados y v una variable entonces:

i (AvB) = i(A) ó i (B)
i ($A_{\wedge}B$) = i(A) y i (B)
i ($\neg A$) = no i (A)
i ($A \Rightarrow B$) = si i (A) entonces i (B)
i ($A \Leftrightarrow B$)= i (A) si y solo si i (B)
i ($\forall v,A$) = para todo vi, i (A)
i ($\exists v,A$)= para algún vi, i(A)

Con este procedimiento se asigna un significado a cada expresión (término o predicado) del lenguaje.

Un cálculo es un sistema formal C=(Lexpr,Axm,Inf) donde Lexpr es el conjunto de expresiones en un lenguaje L, Axm es un subconjunto de Lexpr elegido como axiomas e Inf es un conjunto de reglas de interferencia que a partir de ciertas secuencias de expresiones permite encontrar otras. -

Una expresión e es deducible (es un teorema) en el sistema bajo el conjunto de hipótesis T; $T \vdash e$, si hay una prueba de e, esto es si hay una secuencia finita de expresiones que a partir de los axiomas y las hipótesis (Axm$\cup$T) permite llegar a e por sucesivas aplicaciones de las reglas de inferencia.

Un conjunto T es consistente si para ninguna expresión e vale: $T \vdash e$ y $T \vdash (\neg e)$.

Una expresión e tiene un modelo en la estructura U bajo las hipótesis T; se escribe $T \models e$, si hay una interpretación que satisfaga las hipótesis y hacen verdadera la expresión e .-

Una expresión e es válida si es verdadera en toda posible interpretación sobre U. Se escribe $U \models e$.-

Una teoría T de una estructura U en un lenguaje L es el conjunto de expresiones de L que son válidas en U. -

Una teoría T es decidible si T es un conjunto recursivo. -

Con estas definiciones preliminares, se pueden tratar algunos aspectos teóricos importantes. -

Previamente suponemos que se haya hecho el trabajo de formalizar más rigurosamente lo anterior: elegir el conjunto de símbolos del lenguaje de modo que sea recursivo, con un conjunto numerable de variables, etc. y elegir los axiomas del cálculo de predicados de primer orden en la forma standard y fijar reglas de inferencia adecuadas, (por ej: modus pones) .-

Los siguientes teoremas pueden encontrarse en cualquier buen texto de lógica matemática (por ejemplo: los de J.D. Monk ó Z. Manna) .-

1 .- Los conjuntos de símbolos, expresiones, términos, predicados, axiomas lógicos y las pruebas lógicas a partir de un conjunto recursivo de hipótesis, son recursivos. -

2 .- El conjunto de teoremas lógicos es recursivamente enumerable, pero no recursivo. -

3 .- Teorema de completitud : T⊢e si y solo si T⊨e (en cálculo de primer orden) .-

Como resumen vale la pena notar estos teoremas famosos :

4 .- Hay sistemas formales (de deducción) completos para el cálculo de predicados de primer orden (pero no para el segundo orden) .-

5 .- El problema de validez de cálculo de predicados de primer orden es insoluble, pero parcialmente resoluble. -

6 .- La mayoría de las teorías interesantes (por ejemplo: la teoría de conjuntos, la aritmética de Peano) son indecidibles .-
Estos resultados implican la imposibilidad de obtener programas para resolver cualquier problema por lo que es necesario restringir la interpretación del lenguaje. -

BASE DE DATOS

Una base de datos relacional es BD = (Nrel,Natr,Esq,Int) donde Nrel es una colección de nombre de relaciones; Natr es un conjunto de nombres de atributos, Esq es un conjunto finito de expresiones de la forma Ri (Aj1, ..., Ajn) e Int es un conjunto de condiciones de integridad (expresiones en el lenguaje L) .-

Un estado S es una correspondencia que a cada nombre de relación le asigna una relación finita; esto es: un conjunto finito de n-uplas de valores, cada uno de sus componentes en el dominio del atributo correspondiente. -

Un estado es consistente si cumple con las condiciones de integridad.

Una computación es una secuencia de estados consistentes .

El dominio actual es el conjunto de valores realmente contenidos en la base en un estado. El dominio de definición es el conjunto de posibles valores ('a priori') . -

La idea es restringir nuestro lenguaje L= (Lsim,Oper,Pred) interpretando sus símbolos sobre una base de datos, o mejor: sobre una estructura U' =(Univ', Fun', Rel') donde Univ' se toma como la unión de los dominios de definición de los atributos, Fun' es una colección de funciones recursivas entre los elementos de ese dominio y Rel' el conjunto de relaciones contenidas en la base .

Se toma como símbolo de predicado los nombres de relación y su interpretación es la relación contenida en la base.

Si se limita el universo a la unión de dominios actuales (Valores realmente almacenados en la base en el estado actual), el universo resulta finito y la validez de cada expresión es decidible (basta recorrer la base tantas veces como necesaria, verificando para cada valor si satisface o nó la expresión dada). Pero una restricción de este tipo es demasiada limitativa para nuestros propósitos. Como se pretende actualizar la base, el universo debe incluír todos los valores posibles de ser almacenados en la vida útil de la base.

La idea es dejar el universo irrestricto pero entonces restringir la forma de las expresiones permitidas en el lenguaje, limitando las expresiones a las llamadas fórmulas ajustadas (tight) .-

Una fórmula está en forma normal disyuntiva si tiene la forma:

$$\bigvee_{i=1}^{n} \bigwedge_{j=1}^{m} e_{ij}$$

Esto es: si está expresada como una disyunción de conjunciones.

Una variable está ajustada en una de las conjunciones si:

1) Aparece por lo menos una vez en un predicado básico, no negado.
2) O aparece en una fórmula de la forma X= f(z1,z2,...zn) y z1,...zn están ajustadas.

Una fórmula está ajustada (o apretada?) si en la forma normal disyuntiva todas sus variables libres o ligadas están ajustadas.

Para decidir si una fórmula ajustada es válida, basta verificar la validez de la disyunción, esto es: investigar si se satisfacen alguna conjunción y como cada variable aparece por lo menos una vez en cada conjunción con un predicado básico no negado, o está expresada en función de variable ajustada, entonces basta recorrer los valores de la relación en la base de datos que la interpretación elegida asigna a los símbolos de predicado. Como las relaciones en la base son finitas este procedimiento siempre termina, por lo que se puede concluír que :

" La validez de una fórmula ajustable es decidible "

Con este resultado en mente, vemos de conectar todo lo anterior con el tema de la programación automática . -

LA PROGRAMACION AUTOMATICA

Estamos interesados en programas para resolver problemas. Se comienza formalizando un poco el concepto de problema. -

Sea T una teoría sobre una estructura U expresada en lenguaje L :

$$T=\{e:\ e \text{ es expresión en L y } U \models e\}$$

Un problema en la teoría T es un par P =(e,r) donde e es el enunciado: una expresión en el lenguaje L de la teoría y r es la respuesta pretendida .-

Un problema 'si-no' es un problema donde $r \in \{'si' , 'no'\}$

y se cumple (1) r= 'si' si y solo si $U \models e$
(2) r= 'no' si y solo si $U \not\models e$

Esto es: La respuesta será 'si' en caso que la expresión e sea una fórmula válida en la teoría y será 'no' en caso contrario.
Como las mayorías de las teorías interesantes son indecidibles (Cf.6) Se sigue que en general el problema 'si'-'no' es no computable.-

Como el problema de validez del cálculo de predicados de primer orden es insoluble, pero parcialmente resoluble, se sigue que en un problema del cálculo de predicado se puede hacer un programa que dada una expresión válida, obtenga en tiempo finito la respuesta 'si', en cambio si no es válida puede ciclar indefinidamente. -

Otra clase de problemas son los de forma P= (e(I,0),0) donde la entrada I y la salida 0, son variables con rango en el universo subadyacente de la teoría. El enunciado es: para los datos de entrada I encontrar los resultados 0 que satisfacen la especificación e (I,0) . Esto es encontrar los valores de 0 que verifican

$$U \models \forall I, \exists 0, e(I,0)$$

y por el teorema de completitud esto equivale a :

$$T \vdash \forall I, \exists 0, e(I,0)$$

lo que reduce el problema de la validez al de deducibilidad. -

Una prueba constructiva de este teorema da un método para encontar algún 0, para todo valor de I. La construcción automática de un programa que resuelva el problema se reduce a simular este método. Esta es la base de los sintetizadores de programas basados en probadores de teoremas (se aplica el pricipio de resolución y muchos esfuerzos se concentran en el perfeccionamiento del algoritmo de unificación). Pero los resultados sobre la indecibilidad de la mayoría de las teorías interesantes previene contra la adhesión a este enfoque tan general. Se ha seguido este trabajo bajo un criterio diferente, restringuiendo la forma e interpretación de las especificaciones y por lo tanto, la clase de problemas a tratar.

Lo que se busca es una función recursiva f que aplicada a los datos de entrada I, dé los valores de los resultados 0. Esto es: 0=f(I) de modo que e(I,0) (suponiendo una única respuesta). -
Un programa F es una descripción finita de f en algún lenguaje de programación .

Un sintetizador de programa es un programa Z que a partir de las especificaciones e produce como resultado el programa F que describe a f. Esto es: F =Z(e). Este programa Z, sintetizador de programas, puede verse como un compilador, que traduce el lenguaje de especificaciones L a un lenguaje de programación F (por ejemplo el FORTRAN), de modo que a cada posible ex

presión e en L le hace corresponder un programa F en F .-

Por supuesto, la condición es que cuando el programa F se corra con los datos I, se obtenga como resultado 0, los valores que hacen válida la expresión e (I,0). -
Con las restricciones anotadas, interpretando los símbolos de predicado como relaciones en una base de datos y limitándonos a usar fórmulas ajustadas, el problema es decidible, esto es: recursivamente resoluble.

EL SISTEMA

El sistema propuesto se reduce a un compilador (de hecho un compilador de compilador). Sus componentes principales son :

1) Un analizador lexicográfico: que se limita a detectar unos pocos símbolos ya que la mayor parte del trabajo de análisis se transfiere al parser.
2) Un analizador sintáctico: que aplica el método de Earley para hacer el parsing de gramáticas de contexto libre .
3) Un traductor propiamente dicho: que se basa en un esquema de traducción dirigido por la sintáxis.
4) Un traductor del código intermedio al Fortran.
5) Rutinas que simulan un sistema de manejo de base de datos.

El sistema toma como imput la gramática de la sintáxis y la gramática de la traducción y lee las especificaciones y produce un programa que a su vez lee los datos de una base de datos y almacena los resultados actualizando la misma base de datos. El programa obtenido se limita a recorrer los valores almacenados en la base hasta encontrar los que satisfacen las especificaciones.
Este procedimiento es simple, aunque poco eficiente. (se advirtió que no se han tenido en cuenta los problemas de optimización) . Se omiten más detalles sobre el sistema ya que la mayor parte de su diseño es standard y bien descripto en los textos de diseño de compiladores .-

El funcionamiento del sistema puede resumirse así:

1.- Lee las especificaciones y la lleva a la forma normal disyuntiva de prefijo (cambiando variables por variantes alfabéticas si es necesario).
2.- Para cada variable identifica los predicados básicos en que debe figurar en cada conjunción. Si una variable X no figura en algún predicado lógico en alguna conjunción debe localizar la expresión de la fórma X= f(Z1,...,ZN) en que figura. -
3.- Separa las variable en libre y ligadas. -
4.- Si X1,X2,...XN son las variables libres de la fórmula A genera el código intermedio:

```
for each X1 in U1,do
        :
        :
    for each Xn in Un,do
        if A then insert <X1,X2,...,Xn> in R
    od...od
```

que luego será traducido como un anidamiento de ciclos donde las variables X recorren las relaciones U asociadas contenidas en la base y cada N-upla de valores de las X que satisfacen la fórmula son insertadas en la base como una nueva relación R. -

5.-Si A es de la forma $\forall y,A(y)$ se las sustituye por el código:

```
for each in Uy,A(y)
```

si es de la forma $\exists y,A(y)$ se la cambia por

```
for any Y in Uy,A(y)
```

6.-Estas porciones de código se traducen luego generando ciclos que recorren los valores contenidos en los dominios asociados a cada variable para calcular el valor de verdad de la fórmula.

7. Es posible agregar pasos intermedios (para una mayor eficiencia) por ejemplo: modificar las variables que toman valor en el dominio de definición de los atributos, pueden cambiarse por otras cuyos rangos de valores son las N-upla de las ralaciones como en la práctica habitual en los lenguajes de consulta de base de datos. También es posible optimizar el código, pero estos pasos intermedios no han sido implementados. -

UNA EXTENSION

Tal como se ha descripto el lenguaje de especificaciones sufre de una serie de desventajas comunes a los lenguajes de consulta de base de datos: no es posible obtener la clausura transitiva de una relación.

Un remedio es extender el lenguaje así:

1.- En la sintáxis: Una ecuación es una expresión de la forma

A(X1,...,Xn) == E

donde A es una variable de predicado, X1,...,Xn son variables y E es un predicado, o mejor: una fórmula ajustada cuyas variables libres son: X1,...,Xn .-

2.- En la semántica:Se interpreta A(X1,...,Xn) como una relación R tal que <X1,...,Xn>$\in$R. sii X1,...,Xn hacen verdadera la fórmula E .-
Si la ecuación es recursiva, esto es: A==E(A) donde la variable de predicado figura en ambos miembros de la ecuación entonces se interpreta A como el punto fijo menos definido del funcional descripto por E .-

Otra extensión útil es permitir en la categoría de términos construcciones de la forma :

$$\{x:A(x)\}$$

donde A(x) debe ser una fórmula ajustada.

Con estas extensiones se incrementa el poder expresivo del lenguaje, que supera así al de los lenguajes de consulta de base de datos tradicionales. Se puede obtener la clausura transitiva de una relación y también las funciones de agregado tales como la sumatoria, el máximo y el mínimo de un conjunto sin necesidad de incluír estos operadores como primitivos del lenguaje.
Por último se debería agregar: una especificación es una secuencia de ecuaciones. Esto lleva al problema de estudiar una solución de un sistema de ecuaciones.

Por supuesto, siguiendo las técnicas de la semántica denotacional sería necesario exigir las condiciones de monotonía y continuidad a los funcionales para asegurar la unicidad del punto fijo menos definido, de modo que las ecuaciones recursivas tengan una interpretación aceptable.- Esta discusión está fuera de los límites de este trabajo. -

CONCLUSIONES Y OBSERVACIONES

Se ha descripto un sistema de programación automática como una mera extensión de los lenguajes de consulta de base de datos y se ha expuesto informalmente la teoría subadyacente.
El objetivo principal ha sido poner de manifiesto las dificultades más evidentes.
La restricción introducida de limitar las fórmulas aceptadas a las llamadas ajustadas asegura la existencia de un método de solución. -

Pero aún queda la pregunta: ¿ cuál es la clase más amplia de problemas para la que es posible encontrar siempre un método de solución ?

Se estima que las investigaciones en este sentido permitirán obtener sucesivas ampliaciones del poder expresivo del lenguaje de especificaciones.

Quedan en pié varias cuestiones y algunas ideas que se ha omitido a propósito, que vale la pena mencionar rápidamente :

1.- Como un usuario de un sistema de este tipo está interesado solo en respuestas de longitud finita, las relaciones en la base de datos pueden verse como fórmulas del tipo:

$$P(X) == x=a1 \vee x=a2 \vee \ldots \vee x=an$$

esto es como una disyunción de ecuaciones en la forma x=ai donde x es una variable y ai una constante.
En general se puede tomar todo predicado básico como una abreviatura de una disyunción de conjunciones de ecuaciones, así :

$$P(x1,\ldots,xn) == \bigvee_{j=1}^{m} \bigwedge_{i=1}^{n} xi = ai$$

2.- Aceptado lo anterior, una fórmula aceptable E es una función que a partir de expresiones de longitud finita como las indicadas, permite obtener otras del mismo tipo.

3.- Una especificación dada por la ecuación A==E es encontrar la fórmula de longitud finita expresada como disyunción de conjunciones de ecuaciones que satisface la ecuación dada.

4.- Con lo mencionado es posible una descripción más sintáctica del problema .-
Un sistema de programación automática como el propuesto puede entonces entenderse como un sistema que a partir de expresiones de longitud finita obtiene otras expresiones similares, en un tiempo finito .-

Se estima que esta línea de investigación, puede ayudar a mejorar nuestro entendimiento de los problemas de la programación automática, mientras se esperan los frutos de los estudios en Inteligencia Artificial. Las ideas expuestas son clásicas en las teorías de lenguajes formales, semántica denotacional, base de datos y teoría de la computación.-
La intención ha sido llamar la atención sobre este tema, mostrando un panorama que se estima de interés para todos, quienes estén involucrados en la tarea de desarrollo de software .-

* * * *

BIBLIOGRAFIA

1) J.Backus - Can Programming Be Liberated fron the Von Newman Style. A. Funtional Style Its Algebra of Programs - ACM - Comm.Vol 21,Num. 8, Agost 1978 .-

2) R.B. Banerji - Artificial Intelligence - North Holland - N.Y.1980.

3) M.A.Casanova y P.A. Bernstein- A Fourmal System for Reasoning about Programs Accesing a Relational Database- ACM,TOPLAS -Vol 2,Num.3, Jul 1980.

4) M.Davis- Computability and Unsolvalibity - Mc Graw Hill - 1958.

5) R.B.M. Dewar, A. Grand, SSU-Cheng Liu, y S.T. Schwarts - Programming by Refinement as Exemplified by the SETL Representation Sublanguage - ACM - TOPLAS, Vol 1 N 1, Julio 1979.

6) J.R. Hindley, B.Lerchery, S.P. Seldin -Introduction to Combinatory Logic - Cambridge Press 1972 .

7) D.W.Loveland - Automated Therem Proving: A Logical Basis - North Holland 1978 .

8) Z.Manna - Mathematical Theory of Computation Mc Graw Hill -1974.

9) Z.Manna - R.Waldinger - Studies on Automatic Programming Logic - North Holland - New York 1977 .

10) Z.Manna y R.Waldinger - A Deductive Approach to Programming Synthesis - ACM, TOPLAS, Vol.2 Num. 1, Enero 1980 .-

11) J.D.Monk - Mathematical Logic - Spring Verlag - 1976.

12) N.S.Paywes, A.Pnueli y S. Shastry -Use of a Non procedural Specification Language and Associated Program Generator in Software Development - ACM,TOPLAS . V.1 N 2,Oct. 1979.

13) R.Rusting et al.-Formal Semantic of Programming Languages -Prentice Hall 1972 .

14) J.E.Stoy -Denotational Semantics: The Scott - Strachey Approach to Programming Language Theory - MIT Press - 1979 -

15) D.A.Turner- A New Implementation Technique for Applicative Language- Software - Practice and Experience, Vol 9-31-49-1979.

16) D.H.D.Warren -Logic Programming and Compiler Wrinting - Software - Practice and Experience - Vol. 10, 97-125-1980.-

17) P.H. Winston - Artificial Intelligence - Addison Wesley - 1979 -

18) L.H.Carranza -Teoría de Base de Datos- 11 JAIIO - Oct.1980 - (Contiene otra referencias desde el punto de vista de base de datos.) .-