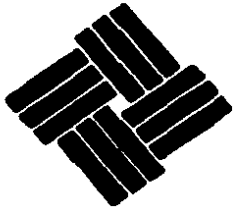
UNIVERSIDAD ANAHUAC

ESCUELA DE ACTUARIA
INCORPORADA A LA U.N.A.M.



"S. E. A.: UN SINTETIZADOR AUTOMATICO DE PROGRAMAS EN APL"

T E S I S

QUE PARA OBTENER EL TITULO DE:

A C T U A R I O

P R E S E N T A :

JESUS OROZCO TOPETE

México, D. F. 1988

SEA: Un Sintetizador Automático de Programas en APL

Un enfoque de Inteligencia Artificial.

I n d i c e

Introduccion

La lectura de esta tesis puede ser una introduccion al tema de Inteligencia Artificial y, en especial, a la Programacion Automatica para los lectores no familiarizados con el tema.

Para los lectores que han profundizado en las areas descritas, la lectura de esta tesis pueda ser la apertura de un panorama distinto y accesible al problema de Programacion Automática y al de resolucion de problemas.

Esta tesis esta dirigida a los investigadores en las areas de programacion automatica y resolucion de problemas. Espero, por tanto, contribuir con mi trabajo al avance de la ciencia en estas dos areas.

El tema de Programacion Automatica (de aqui en adelante denominada P.A.) es, al igual que muchas de las ramas de la Inteligencia Artificial, un tema controvertido en cuanto a su definicion y a su diferenciacion de otros campos fuera de la Inteligencia Artificial.

Antes de dar la definicion para el termino Programacion Automática (también llamada Síntesis Automatica de Programas) que se adoptara en esta tesis, mencionaremos las ideas de Manna y Siklossy, al respecto.

El comentario de Manna resalta lo que es la sintesis de un programa:

"La síntesis de un programa es la construccion de un programa en computadora a partir de especificaciones dadas. Un sistema de síntesis automatica de programas debe combinar la habilidad de razonamiento y la de programacion con una amplia cantidad de conocimiento del tema del programa. Esta habilidad y conocimiento deben ser manifestados tanto en procedimientos (por medio de programas) como en estructura (por medio de la eleccion de representacion). ...". Manna (71).

La idea de Siklossy es en torno a lo que un sintetizador de programas hace:

"... Un sintetizador de programas transforma la descripcion implicita o explicita de un algoritmo en un codigo ejecutable. Aunque un compilador de un lenguaje de alto nivel pueda ser considerado un sintetizador, ya que transforma un algoritmo

escrito en el lenguaje, en codigo de maquina ejecutable. generalmente las discusiones de sintetizadores de programas estan restringidas a los sistemas que hacen transformaciones en descripciones de código que están lejos de ser ejecutables. El concepto de 'lejos' es relativo al estado del conocimiento. ...". Siklossy (42).

De acuerdo a la idea de Siklossy, es necesario enfatizar la diferencia que hay entre un sintetizador de programas y un compilador. A nuestro juicio, el primero es capaz de lograr aprendizaje, cambiar su conducta conforme se van sintetizando programas, resolver problemas no previstos y hacer inferencias sobre algun tema en particular. En el caso de un compilador no existe ninguna de estas cuatro características.

Entendemos por el termino "aprendizaje", en un sintetizador de programas, el incrementar o disminuir el numero de programas que se encuentran en la memoria del sintetizador. Lo anterior repercutira en un cambio de conducta durante el proceso de síntesis.

Entendemos por "cambio de conducta", en un sintetizador, el producir un codigo distinto ante las mismas especificaciones del programa a sintetizar. Esto es debido al incremento o decremento de alternativas, causado por el aprendizaje.

Por otra parte, es importante resaltar lo que Manna señala respecto al conocimiento que debe existir acerca del tema del programa. Este hecho es una consecuencia directa de la falta de capacidad de un sintetizador de inventar o descubrir conocimiento. Por ello no se puede pedir a un sintetizador que trate de sintetizar un programa del que no tiene conocimiento alguno. Este hecho es igualmente aplicable a un programador humano.

En nuestra definicion de P.A. enfatizaremos el hecho externo de lo que un sintetizador automatico de programas significa para un usuario, más que el proceso interno que esta detras de la síntesis de un programa.

En esta tesis, entenderemos por P. A. la sustitucion del ser humano, en lo referente a la programación de una computadora, por una computadora programada que sea capaz de obtener el codigo de un programa que realice alguna función que ha sido especificada por un usuario. La especificacion de tal función tendra las características siguientes:

- Estara hecha en un lenguaje de comunicacion que permita la interrelacion entre un usuario y la computadora que sintetizara el programa.

- Dentro de la especificacion, no necesariamente esta implicito el codigo que se espere sea el resultado.

- Deberá ser completa a nivel de lenguaje, con el fin de que el equivalente en codigo de esa especificacion corresponda a la intencion del usuario respecto al codigo resultante. Es decir, que lo que el usuario quiere, sea "entendido" con toda precisión por la computadora.

Esta ultima caracteristica de una especificacion de programa ayudara mas tarde a que definamos el significado de una descripcion completa de un programa.

Ahora que ya hemos dicho lo que entendemos por P.A., pasemos a destacar la importancia que esta tiene hoy en dia.

La investigación en P.A. resulta importante, desde el punto de vista teorico de Inteligencia Artificial, por su vinculacion en las areas de aprendizaje, resolucion de problemas, prueba de teoremas y otras derivadas de estas.

Con lo anterior queremos decir que el enfoque dado en esta tesis al problema de P.A., ayudara a entender un teorema como un problema y a la vez como un programa a ser sintetizado y que la capacidad de síntesis, es decir de resolucion de problemas o de prueba de teoremas, sera incrementada a medida que se dé aprendizaje.

El analisis del proceso de aprendizaje en la síntesis de programas puede resultar util en otras areas de la Inteligencia Artificial. Esto se debe a que el enfoque particular de P.A. en lo referente al aprendizaje muestra una vision propia del cambio de conducta en un "programa inteligente", una vez que nuevo conocimiento ha sido adquirido.

Dentro de la resolucion de problemas, la P.A. puede ser de gran ayuda cuando un problema logre ser expresado como un programa a ser sintetizado. Lo anterior es particularmente cierto cuando el enfoque usado en la P.A. es del tipo de resolucion de problemas.

Finalmente, hemos dicho que el área de prueba de teoremas tambien esta vinculada con la P.A., esto se debe a que un teorema puede ser representado por un programa en computadora. Una manera de lograr esto es considerando a los programas existentes, en la memoria del sintetizador, como premisas que han sido aceptadas como verdaderas, y el programa a sintetizar, como el teorema por demostrar.

En el campo de aplicacion propio de P.A., es evidente la gran aplicacion que tendra, en un futuro, en la generación de software de una manera rapida y sencilla.

Mediante SEA se pretende dar un enfoque novedoso al tratamiento del problema de P.A. en lo referente a la representacion de conocimiento y a la forma de resolver el problema de sintesis.

En este enfoque se ha puesto énfasis en la concepcion del "conocimiento" y del "entendimiento", logrando esquemas sencillos de conocimiento, lo cual ha dado lugar a una mecanica simple para la resolución de problemas.

El fruto de esta investigacion es el sintetizador SEA (Sintetizador en AFL). SEA es un conjunto de funciones, escritas en lenguaje AFL, que permiten la sintesis de programas, a partir de especificaciones del programa que se desea sintetizar.

Las especificaciones de un programa son dadas a SEA mediante el uso de una tabla, con la cual se transcribe informacion especifica en cada una de las casillas de la tabla. El programa ya sintetizado es desplegado en la pantalla por SEA.

Cabe notar que el proceso de sintesis de SEA esta ligado al hecho de que el codigo resultante sea en lenguaje AFL. Es dejado para futuras investigaciones el ampliar la capacidad de SEA para que pueda lograr la sintesis de programas en otros lenguajes de programacion.

La razon para usar el lenguaje de programacion AFL, en la construccion de SEA, fue el demostrar que el AFL es una herramienta apropiada en la elaboracion de programas inteligentes. Con lo anterior no queremos decir que no existan lenguajes mas adecuados, ni tampoco hemos pretendido dar una opinion comparativa al respecto.

Dentro del contexto de P.A., SEA puede jugar un papel muy importante debido a la flexibilidad que posee. Esto se debe a que SEA puede estar involucrado en cualquier campo del conocimiento humano que encuentre alternativas de tratamiento por medio de una computadora. Lo anterior es una diferencia respecto a otros sintetizadores que están "encadenados" a un tipo exclusivo de problemas.

Sin embargo, donde SEA encuentra flexibilidad tambien encuentra su gran limitante. Al ser cierto que SEA puede intervenir en campos variados, tambien es cierto que requiere de un gran volumen de información en cada una de las areas donde opera. Por lo tanto, antes de esperar que SEA sintetice una amplia variedad de programas, es necesario alimentar la información suficiente para que SEA "conozca" muchos de los detalles del mundo que no le han sido proporcionados.

SEA tiene tres elementos que lo distinguen de otros sintetizadores y que le dan la capacidad requerida de solucion de problemas para la síntesis de un programa:

- Una representacion de conocimiento propia.
- Estrategias propias empleadas en la síntesis de programas.
- Una base de conocimientos existentes en SEA, previos a la síntesis de un programa.

En este trabajo hemos incluido dos estrategias de síntesis, sin embargo, no dudamos que puedan existir otras que llevarían a SEA a conseguir la síntesis de programas que actualmente no es factible realizar sin una alimentacion "amplia" de conocimientos.

Cabe ahora mencionar que hemos dado especial enfasis en el tratamiento de los tres problemas mencionados antes, resolviendo de una manera parcial e ineficiente el resto de los problemas inherentes a la P.A.. Tal es el caso del proceso de especificacion de un programa, que requiere familiaridad, por parte del usuario, de la representacion de conocimiento de SEA para dar una especificacion adecuada de un programa.

Por otro lado, el enfoque de otros autores les ha llevado a dar mas importancia a otros problemas de la P.A. y tratar de una manera un tanto ligera lo que para nosotros ha sido importante. Así por ejemplo, tenemos el caso de Jouannaud (24), Silliossy (42) y Shaw (41), que se concentraron en el problema de la especificacion de los programas a ser sintetizados mediante ejemplos.

Otros autores han puesto su atencion en el problema de la síntesis, mas que en el de la especificacion del programa a ser sintetizado, al igual que hemos hecho nosotros. Tal es el caso de Clark (8) y Green (18), en cuyos trabajos hicieron uso de herramientas como son Prueba de Teoremas y Cálculo de Predicados.

Este trabajo tuvo su inspiracion en trabajos como el de Guida (22), Manna (31), Balzer (2), Caio (5), Katz (25) y Manna (33), donde trataron el problema de P.A. como un caso de resolucion de problemas en el que la solucion puede ser logrado caminando de atras hacia adelante, es decir, desde el objetivo, que es la especificacion del problema, hasta los argumentos del programa y los conocimientos previos en la memoria del sintetizador.

Un elemento metodologico que puede resultar interesante, es que para confrontar el aspecto tecnico central de este trabajo, que es el de la solucion de problemas, se observo el

comportamiento en niños ante situaciones de aprendizaje y resolucion de problemas, al igual que el de programadores experimentados al realizar un programa.

Las observaciones realizadas del comportamiento humano llevaron a las preguntas siguientes:

- ¿ Que representacion de conocimiento debe usarse, de tal forma que facilite el proceso de sintesis y lo haga eficiente ?

- ¿ Cual debe ser el proceso, en la solucion de un problema, para que este sea lo suficientemente general y pueda resolver una gama amplia de problemas a partir de la representacion de conocimientos usada ?

- ¿ Que conocimientos deben existir en la memoria de SEA, previamente a la resolucion de un problema ?

Al tratar de responder las ultimas dos preguntas, fue necesario volver a resolver la primera, cada vez que se detectaba falta de generalidad o poca eficiencia en la representacion de conocimientos usada.

Lo anterior llevo a perfeccionar la representacion de conocimiento, al igual que las estrategias de sintesis usadas, de tal forma que se fueron logrando versiones cada vez mejores de SEA.

Actualmente, SEA cuenta con 2 estrategias de solucion de problemas. Ambas estrategias requieren de la intervención del usuario, por carecer del modulo de Procesamiento de Terminos Algebraicos, cuya adaptacion a SEA es dejado para futuras investigaciones.

El procesador de terminos algebraicos tiene la funcion de determinar si un sistema de ecuaciones, en notacion AFL, tiene solucion o no. En caso afirmativo debe proporcionar la solución. Es en estas dos funciones donde el usuario debe intervenir durante el proceso de sintesis.

La tesis se encuentra dividida en capitulos que se describen a continuación.

El capitulo I trata el problema de la representacion del conocimiento y describe la tabla usada para ello; ademas, justifica el uso de dicha tabla.

El capitulo II muestra un ejemplo de cada tipo de estrategia usada por SEA para el proceso de sintesis, indicando el estado de

la memoria de SEA, las especificaciones del programa deseado y el resultado obtenido.

El capitulo III refiere los supuestos basicos indispensables para que SEA opere adecuadamente, describe la mecanica del funcionamiento de SEA y hace una breve reseña de las dificultades encontradas en la investigacion y la forma en como fueron tratadas.

El capitulo IV muestra las conclusiones obtenidas y menciona el trabajo que se deja para futuras investigaciones.

I. Representacion del conocimiento.

Entenderemos por conocimiento en SEA la especificacion de un programa a ser sintetizado, los programas que existen en la memoria del sintetizador y las estrategias usadas para realizar la sintesis.

En la realizacion de, SEA fue importante el definir con detalle la representacion de conocimiento a usar. Tal representacion deberia resolver satisfactoriamente los problemas de comunicacion con el usuario en el momento de la especificacion del programa a ser sintetizado y, por otra parte, deberia ser una representacion que garantizara ser util durante el proceso de sintesis.

El problema de comunicacion con el usuario lo denominaremos problema de ambigüedad. Es decir, se pretende evitar un mal entendido por parte de la computadora respecto a los requerimientos del programa a ser sintetizado.

Para llegar a una solucion del problema de ambigüedad, se establecio una hipotesis en la especificacion de programas, de tal forma que si el usuario la respeta, el problema habra desaparecido.

Esta hipotesis consiste en que el usuario debe especificar los programas a ser sintetizados dando una "Descripcion Completa" de los mismos.

A continuacion estableceremos lo que se entiende por una "descripcion completa de conocimiento", pero antes de definir este concepto haremos uso de un ejemplo.

Supongamos que tenemos un lapiz y establecemos los siguientes postulados para DESCRIBIRLO:

- Cuerpo solido.
- Material: Madera.
- Peso: 27 gramos.
- Color: amarillo.
- Forma: cilindrica.
- Longitud: 7 cm..
- Radio: 0.25 cm..

Esta descripcion del lapiz pareceria ser suficiente para distinguirlo de cualquier otro objeto presente a su alrededor. Sin embargo, si colocamos otro lapiz "igual" junto al primero y despues los revolvemos, no habra un hecho determinante que nos indique cual de los dos es el primero, a partir de la descripcion dada antes.

La descripcion de arriba no se refiere exclusivamente a un objeto en el universo, sino que se refiere a una clase de objetos: todos los objetos del universo que cumplan con la descripcion dada. (En este caso, el conjunto de los lapices amarillos, de madera. ...)

Es cierto que podremos dar descripciones de programas y de clases de programas. Un sintetizador de programas no debera detenerse en distinguir entre un programa o una clase de estos, ya que le bastara el sintetizar un programa que represente la descripcion dada.

Regresando al problema de ambigüedad, observamos que para indicar la clase de programas a la que pertenece una descripcion de programa, el usuario debera dar la descripcion completa de esa clase. De la misma forma que si pedimos un lapiz que pinte de color gris, debemos entonces pedir un lapiz gris y no solamente un lapiz. Si pidieramos exclusivamente un lapiz, estamos dando la libertad de que se fijen las demas variables de manera no controlada y que si nos dan un lapiz que pinte rojo nos habran cumplido el requerimiento de darnos un lapiz.

Es factible que en el ser humano se realicen de manera inconsciente estadisticas de los valores que tomen ciertas variables de ciertos objetos. Tal es el caso de un lapiz que, por ejemplo, el 90 % de los casos pinta gris cuando se habla de él. Sin embargo, en el presente trabajo, tal tratamiento no ha sido considerado.

En base a lo anterior, si el usuario respeta la hipotesis mencionada, habremos superado el problema de ambigüedad en la especificacion de un programa a ser sintetizado. (Para referencias sobre este tratamiento consulte Ernst (27) y Carbonell (28)).

Como un resumen de lo anterior observamos que, para evitar el peligro de ambigüedad en los conocimientos, a cada descripcion completa de conocimiento o clase de conocimientos del mundo externo le debe corresponder una y solo una equivalencia en codigo del lenguaje de programacion, y de igual manera, a cada codigo (programa) del lenguaje de programacion le debe corresponder una y solo una descripcion completa de conocimiento. Lo anterior exige al usuario ser consistente en sus equivalencias entre el mundo externo a SEA y la representacion de ese conocimiento como un programa dentro de SEA.

Por tanto, SEA considerara a todas las representaciones de conocimiento como descripciones completas de conocimiento, con lo que se evite el problema de ambigüedad si se respetan las condiciones del parrafo anterior.

Hemos analizado el problema de la representacion del conocimiento en funcion de exigir al usuario el ser consistente con esas representaciones. Más adelante mencionaremos las herramientas con las que cuenta el usuario para generar representaciones de conocimiento.

El siguiente elemento a considerar en la representacion de conocimiento es la síntesis de un programa. El proceso de síntesis hace uso de los tres tipos de conocimiento que existen en SEA:

1. Los programas que se encuentran en la memoria de SEA, previos a la síntesis de un programa.

2. La subrutina de síntesis, la cual se encarga de obtener el código esperado.

3. La especificacion del programa a ser sintetizado.

En cuanto al primer tipo de informacion que manejaremos, los programas, consideramos, para efectos de este trabajo, que estan constituidos por argumentos, procedimientos y resultados.

Una buena representacion de un programa es aquella en la que podemos tener una descripcion completa de los tres elementos mencionados.

Así, para definir un programa en forma completa, sus tres elementos deben estar expresados, a su vez, en forma completa. Para ello haremos uso de los atributos, que son "calificativos" del ente de conocimiento al que califican: argumento o resultado. Notese que solamente existen atributos para los argumentos y para el resultado de un programa.

Para un procedimiento no existen atributos, ya que un atributo de procedimiento puede ser expresado como un atributo de resultado. Es decir, cuando es importante en la ejecucion de un programa el procedimiento usado, parte del resultado o efecto deseado sera causado por el procedimiento usado. Por lo tanto, para distinguir dos programas que difieren en el procedimiento, sera necesario usar atributos de resultado para diferenciarlos.

Tal es el caso de efectos laterales deseables en los programas. Por ejemplo, dos programas que abren un archivo y obtienen la suma de los datos contenidos en el; un programa deja el archivo abierto y el otro cerrado. Aun cuando los dos programas pertenecen a la clase de programas que abren ese archivo y suman sus datos, se les puede distinguir agregando un atributo de resultado indicando el estado final del archivo usado.

Cabe distinguir dos hechos importantes. Cuando se esta alimentando la base de conocimientos (programas) de SEA, o bien cuando SEA termina de sintetizar un programa, la informacion que se graba es: argumentos, atributos de argumentos, resultado,

atributos de resultado, las restricciones que deben guardar los argumentos y las variables de ambiente, y el codigo del programa (que representa al procedimiento). Ahora bien, en el caso de pedir a SEA que sintetice un programa, la información proporcionada es: Argumentos, atributos de argumentos, resultado y atributos de resultado. La diferencia esta en que en los primeros se considera las restricciones y el código, y en el segundo caso no.

Lo anterior está en concordancia con los criterios de esta tesis acerca de los elementos que debe tener un programa. Un programa en computadora consta de argumentos, procedimiento y resultado. Cada argumento y el resultado constan de una descripción corta y de atributos. El procedimiento esta definido por la descripcion corta y atributos de los argumentos, la descripción corta y atributos del resultado y el conjunto de restricciones que deben ser satisfechas.

Ante la situación de que no conocemos a priori toda la gama de conocimientos que se manejaran y el tamaño de las descripciones completas, debemos hacer que ese tamaño sea flexible y poder así facilitar la existencia de descripciones completas para cada uno de los componentes de un programa.

Hablemos en primer termino de los argumentos. Ellos son en todo lenguaje de programación antes de conocimiento presentes, o por lo menos como un supuesto están presentes por lo que no nos debe preocupar su obtención; están compuestos de una especificación corta y del conjunto de sus atributos.

En lo referente al resultado, diremos que este es el punto resultante de todo programa y esta representado por una especificacion corta y por el conjunto de sus atributos.

Respecto a los procedimientos realizados no definimos una descripcion particular, sino que para el caso de SEA: el nombre y atributos de los argumentos, el nombre y atributos del resultado y el conjunto de las restricciones que deben ser respetadas en el programa, definen de manera unica el procedimiento de un programa. Como se vera mas adelante, para cada argumento, atributo de argumento y atributo de resultado, se le podrá asignar una relación, en lenguaje natural, con el procedimiento usado.

De lo anterior se concluye que SEA no descubre codigo ejecutable sino que sabe combinar sus conocimientos para lograr la solución de un problema. Es decir, SEA no descubre procedimientos de programación sino que usa los ya existentes de manera individual o bien mediante composición de los mismos para la síntesis de un programa. Lo anterior lo logra cambiando, despejando de ecuaciones, los valores de los argumentos de los programas ya existentes.

Es decir, cuando se especifica un programa a ser sintetizado, con argumentos, atributos de argumentos, resultado y atributos de resultado, SEA busca en su base de conocimientos (programas) alguno que genere el mismo resultado y tenga por lo menos los mismos atributos de resultado. Podría tener más atributos de resultado que los pedidos.

En caso de no encontrar ningun candidato, el proceso de síntesis fracasa.

En caso contrario, existiendo algun candidato, o varios, SEA intentará transformar, mediante otro programa existente en su base de conocimientos, los argumentos del programa a ser sintetizado en los argumentos del programa candidato.

Como se puede observar, el funcionamiento de SEA se basa en la manipulacion de argumentos de programas y no en la invencion de codigo.

Tabla de información

Una vez que hemos mencionado los tipo de conocimiento que tendremos en la memoria de SEA, es decir, los programas previos a la síntesis de un programa, al igual que los elementos de estos, argumentos, atributos de argumento, resultado, y atributos de resultado, resta el ver la forma en que SEA los representa internamente.

Cada programa en el banco de información de SEA, se representa mediante una tabla. Vea la tabla 1.

La tabla de información es una herramienta para representar los elementos de un programa que esté, ya sea en la memoria de SEA, o bien, que vaya a ser sintetizado.

Las primeras nueve casillas se utilizan en la representación del codigo y las restricciones de los argumentos.

A partir de la casilla numero 10, se agrupan de seis en seis las casillas con el fin de representar atributos de resultado, argumentos y atributos de argumento.

Cabe ahora mencionar que la estructura de la tabla de información está intimamente ligada con la síntesis de programas en el lenguaje RPL. Es decir, se ha reservado algunas casillas para contener información específica del ambiente RPL. Tal es el caso de las casillas que indican la estructura de algun dato (vector, matriz, etc), la dimensión de dicha estructura, el origen de indices y las restricciones de algunas variables de ambiente que fuera necesario considerar en algun caso particular.

OBJE	DIMENSION DEL RESULTADO	A F L	CONDICION DE VALIDEZ
FORMA	1	4	--------
CONDVALIDEZ	--------	5	3
A F L	2	6	--------
CUALIDAD	3	7	9

OBJE			
TIPO DE ATRIBUTO	10	16	...
NOMBRE ATRIBUTO	11	17	...
FORMA	12	18	...
CONEXION-FUNCION	13	19	...
A F L	14	20	...
CUALIDAD	15	21	...

Tabla 1

A continuacion se describe cada una de las casillas. En los casos en que la casilla explicada indique información en lenguaje natural, se quiere decir que se puede proporcionar una frase no restringida en lenguaje natural. Es decir, su dominio es el conjunto de las frases en lenguaje natural.

En la casilla superior izquierda de los dos cuadros de la tabla, aparece el nombre que tiene el programa, dentro de la memoria de SEA. Cuando este nombre es "OBJE", quiere decir que se trata del programa objetivo, es decir, el programa a ser sintetizado.

Las casillas 1, 2 y 3, dan una descripcion corta del resultado.

1. Indica la forma o configuracion del resultado. Los valores validos para esta casilla son: vacio, escalar, matriz, o arreglo.

2. Indica la forma o configuracion (shape) del resultado con una formula en APL en terminos de los argumentos del programa, o bien como una constante.

3. Da la cualidad del resultado en lenguaje natural. Por ejemplo si el resultado fuera un vector de numeros primos, en esta casilla diria PRIMO.

Un ejemplo de estas tres primeras casillas es el de un programa cuyo resultado es un VECTOR (casilla 1) de 8 componentes (casilla 2) y consta de NUMEROS PRIMOS (casilla 3).

4. Esta casilla esta libre para una futura utilizacion en la descripcion de un programa. Por ahora no se usa.

5. El codigo en lenguaje natural y primitivos del APL del programa representado.

6. El codigo en lenguaje APL, que es el programa representado por toda la tabla.

7. El origen de indices (parte del medio ambiente propio de APL) para el que esta pensado que funcione el programa.

8. Las condiciones que deben prevalecer (de los argumentos, variables globales y del sistema) para que el programa funcione adecuadamente.

9. Esta casilla esta libre para una futura utilizacion en la descripcion de un programa. Por ahora no se usa.

Las casillas siguientes, estan ordenadas de seis en seis. De tal forma que un juego de seis casillas representa alguno de los siguientes elementos del programa:

+ Atributo de resultado.
+ Argumento, o
+ Atributo de argumento.

Las casillas 10,18,... se denominan Tipo de Atributo y pueden tomar los valores R, indicando que el juego de casillas encabezado por esta se refiere al resultado, o bien A##, que indica que el juego de casillas encabezados por esta se refiere a:

1.- El argumento A##, o

2.- Un atributo del argumento A## si seis casillas antes aparecía el mismo A##.

o bien un par del tipo A##A## para el caso de todos los argumentos, ya que estos, siempre se considerarán como atributos del resultado.

Las casillas 11,17,... se denominan Nombre del Atributo y pueden tomar el valor de una palabra o frase en lenguaje natural. Estas son las unicas casillas que están relacionadas con el procedimiento usado por el codigo.

Las casillas 12,16,... se denominan Valor Forma, o configuracion, del atributo y pueden tomar alguno de los valores: escalar, vector, matriz, arreglo.

Las casillas 13,19,... se denominan Configuracion y toma el valor numerico de la configuracion descrita en las casillas 12,16,... respectivamente. Es decir que si la configuracion de este atributo es Vector, entonces en esta casilla aparecera reflejado el numero de componentes que tiene, aun cuando sea mediante alguna formula en APL o bien se omite por no ser relevante el dato.

Las casillas 14,20,... se denominan APL y contienen el codigo en lenguaje APL del atributo de resultado, argumento o atributo de argumento, representado por el juego de las seis casillas a que pertenece esta casilla.

Las casillas 15,21,... se denominan Cualidad y se una(s) cualidad(es) en lenguaje natural del argumento, atributo de argumento o atributo de resultado, representado por el juego de las seis casillas a que pertenece esta casilla. Esta casilla ayudará en la realizacion de la descripcion completa.

El usuario deberá tener cuidado de escribir las frases en lenguaje natural, que tengan que ver con el procedimiento, en las casillas 11,17,... y de escribir en las casillas 15,21,..., las cualidades que consideren en forma aislada al objeto a que califican.

Por ejemplo. Para expresar el argumento "Multiplo de 3", que en una programa aparece como "Dividendo', se debe colocar la palabra "Dividendo" en la casilla 17 y la palabra "Multiplo de 3" en la casilla 21.

La tabla de conocimiento descrita tiene la ventaja de poder ampliar "ilimitadamente" el numero de atributos, ya sea de argumento o de resultado, lo cual permite la existencia de descripciones completas.

Un ejemplo de esto es la especificacion de un programa, que de por resultado un vector de numeros naturales, mayores a 1000, menores a 2000, y multiplos de 5. Esto quedaria representado por la tabla 2.

Esta representacion de conocimientos dentro de SEA permite localizar con precision cada elemento de un programa.

Por ejemplo en el caso concreto de la estrategia de solucion de SEA, cuando se este buscando un programa que de por resultado un Escalar (forma del resultado) Entero (calidad del resultado), se seleccionaran aquellos cuyas casillas 1 y 3 tengan los valores Escalar y Entero respectivamente. Y si ademas nos interesa que dicho escalar entero sea tambien par, SEA buscara un atributo que se refiera al resultado de entre aquellos atributos cuya casilla 10 + (k*6) contenga el caracter R, por referirse al resultado (para algun k entero mayor o igual a cero) y que en la casilla 15+(k*6), para la misma k de antes, valga PAR .

Para especificar un programa, se deberan guardar los siguientes supuestos de especificacion:

- Todos los argumentos deben aparecer en la casilla 8, de lo contrario no serian argumentos.
- Cada atributo, ya sea de resultado o de argumento, está representado por un solo juego de seis casillas.
- En un mismo programa no puede haber dos argumentos distintos que sean iguales en el tipo de argumento ni en su nombre.
- Debe haber alguna informacion en las casillas 1 y 3 de todo programa, ya sea que se vaya a sintetizar o este ya sintetizado. En el caso de un programa ya sintetizado, la casilla 5 debe contener el codigo del programa. El resto de las casillas pueden permanecer vacias si asi lo requiere la descripcion completa.
- Cuando se esté especificando un programa para que sea sintetizado, se dejaran vacias las casillas 4 a la 9.

OBJE	DIMENSION DEL RESULTADO	A P L	CONDICION DE CALIDAD
FORMA	VECTOR		--------
CONJ CUALIDAD	--------		
A P L		1000+5x^1+1200	--------
CUALIDAD	NUMEROS NATURALES	1	

OBJE	R	R	R
TIPO DE ATRIBUTO			
NOMBRE ATRIBUTO	MAYOR QUE	MENOR QUE	MULTIPLO DE
FORMA	ESCALAR	ESCALAR	ESCALAR
CONDICION			
A P L	1000	2000	5
CUALIDAD	ENTERO POSITIVO	ENTERO POSITIVO	ENTERO POSITIVO

Tabla 2

- En las casillas que se deba especificar un valor predeterminado, ESCALAR, VECTOR, etc., solo puede haber alguno de esos valores. Esto sucede en las casillas 1, 12, 18, 24,....

- Es responsabilidad del usuario el no poner atributos de un mismo argumento, o de resultado, que sean contradictorios.

- Cuando deba especificarse una expresion en APL, como en las casillas 6, 14, 20, 26, ..., se debera respetar la sintaxis del lenguaje APL. Ademas, solo podra haber argumentos de la forma A##.

- Las casillas que deban ser llenadas con frases en lenguaje natural, no tienen restriccion alguna en cuanto a su contenido.

- Los identificadores de argumento deben ser de la forma A##, donde cada # es un digito del conjunto (0,1,2,3,4,5,6,7,8,9).

- En el caso de los argumentos, deben tener un tipo de atributo (casilla 10,16,...) como el siguiente: R;A##. La "R" aparece debido a que todo argumento es un atributo del resultado. El ";" representa, por ahora, el caracter numero 220 del vector atómico del APL, tiene la funcion de separador. La "A" seguida de dos digitos representa al identificador del argumento.

- No puede haber dos argumentos distintos con el mismo identificador "A##".

- Los atributos de argumento deben estar en las casillas inmediatas siguientes al argumento que pertenecen y su tipo de atributo debe ser el mismo "A##" que se uso como identificador del argumento. Es decir, el siguiente conjunto de seis casillas se referira al atributo de argumento y llevara el mismo tipo de atributo, es decir identificador de argumento, (casilla 10 = 9+k), que el argumento, A##.

Es evidente que un sobobjetivo solo encontrara atributos de argumento. Es decir, no encontrara atributo de atributo de argumento, etc., por lo que el proceso de sintaxis se detiene en el nivel 2 del arbol de busqueda.

- Un atributo, no puede ser al mismo tiempo atributo de dos o mas argumentos o atributo de resultado y de argumento.

- La casilla APL de todo argumento (14, 20, ...) debe denominarse con una variable con el mismo identificador del argumento, es decir, con la forma

A##.

- Cuando un atributo de argumento tiene un valor desconocido para la casilla APL (14, 20, ...), se denominara, en esta misma casilla con alguna de las variables V1, V2, ..., o V9. Cuando un mismo argumento tenga mas de dos atributos con estas caracteristicas, se usara otra variable distinta de este mismo conjunto mencionado. Esta tipo de variables no esta restringido, en cuanto a que sea constante o variable.

- El origen de indices puede tomar el valor de 1, o 0 o ambos, en la casilla 7. En el caso que toma ambos valores, significa que el programa funciona bien para ambos casos. Cuando los dos valores aparecen, estaran separados por el caracter 220 del vector atomico.

- Al especificar un programa, las restricciones deberan estar separadas, en la casilla 6, por el caracter 220 del vector atomico del APL.

Resumiendo. Un programa en computadora consta de argumentos, procedimiento y resultado. Cada argumento y el resultado constan de una descripcion corta y de atributos. El procedimiento esta definido por la descripcion corta y atributos de los argumentos, la descripcion corta y atributos del resultado y el conjunto de restricciones que deben ser satisfechas. En el momento de la definicion de un programa, se permite definir, en lenguaje natural, la relacion que existe entre el procedimiento y los argumentos, atributos de argumentos y atributos de resultado, mediante la casilla NOMBRE ATRIBUTO (11, 17, ...).

El conocimiento que SEA tiene, acerca de las estrategias de sintesis de un programa, esta implicito en el codigo de SEA. En el capitulo siguiente, se describen las dos estrategias de sintesis de SEA.

Justificacion de la representacion

La tabla de representacion de conocimientos en nuestro trabajo muestra algunas caracteristicas interesantes:

- El nombre y cualidad de un argumento o resultado, no estan relacionados con el lenguaje de programacion, por lo que aqui se puede emplear el lenguaje natural necesario para hacer una descripcion completa.

- Si el nombre y cualidad de un argumento o resultado no son suficientes para una descripcion completa, entonces se puede recurrir al uso de atributos (tantos como sean necesarios).

Cabe notar que un atributo (de argumento o de resultado) utiliza el mismo tipo de casillas que un argumento.

El logro fundamental de esta representación de conocimiento es el haber sistematizado el conjunto de conocimientos contenidos en un programa, facilitando el proceso de síntesis.

II. Dos ejemplos del funcionamiento de SEA

Por su construccion, SEA esta preparado para resolver dos tipos de problemas, que van intimamente ligados con los dos tipos de estrategias que se usan para la resolucion del problema de P. A. en esta tesis.

Al referirnos a cada uno de los tipos de problemas, que SEA puede resolver, nos estamos refiriendo a las estrategias de solucion, que dan lugar a la resolucion de determinados problemas.

La estrategia I ataca el problema de síntesis, mediante la búsqueda de algun programa, en su base de conocimientos, que realice la funcion pedida a partir del mismo tipo de argumentos que hay en el programa objetivo.

Se dice que un programa, en la base de conocimientos, realiza la misma funcion pedida por el programa objetivo, cuando la descripcion corta y atributos del resultado del programa objetivo son "iguales" a los del programa de la base de conocimientos.

En los casos de los argumentos, atributos de argumentos y atributos de resultados, el sentido de igualdad no es estricto, sino que se evalua de acuerdo a la tabla 3.

		PROGRAMA EN LA BASE DE CONOCIMIENTOS	
		CONSTANTE	VARIABLE
PROGRAMA OBJETIVO	CONSTANTE	Se acepta solo si son iguales	Se acepta siempre
	VARIABLE	Se rechaza siempre	Se acepta siempre y se sustituye

ACEPTACION O RECHAZO DE UNA IGUALDAD

Tabla 3

Esto quiere decir, por ejemplo, que si en el programa objetivo, un atributo de resultado dado es una variable (casilla [4 + 5*j, para algun j >=0] y uno de los programas candidatos, que se encuentra en el banco de programas de SEA, que tiene las casillas 1 y 3 idénticas a las del programa a ser sintetizado (es decir, sus descripciones cortas del resultado son idénticas) y

tiene un atributo del resultado del mismo tipo que el del programa objetivo, siendo una constante (casilla 14+6*k, para algun k >=0), entonces el programa candidato es rechazado. Esto debido a que el programa existente funciona para un caso particular (constante) y se espera que funcione para un caso general (variable).

Dos atributos son del mismo tipo cuando las casillas 11 + 6*j, 12 + 6*j y 13 + 6*j, de uno son identicas a las casillas 11 + 6*k, 12 + 6*k y 13 + 6*k, de otro, para algun j y un k >=0. Es decir, que tienen el mismo nombre, la misma forma y la misma cualidad. Por tanto, las unicas casillas que se someten a la prueba constante variable de la tabla 3, son la 14 + 6*j y la 14 + 6*k, de cada programa respectivamente.

Recordando que estamos hablando en terminos del lenguaje APL, no bastara que la tabla "constante variable" sea aceptada sino que ademas la configuracion de los terminos sea igual (escalar, vector, etc.). Esto se afirmo antes con la identidad de las casillas 12 + 6*j y 12 + 6*k.

El hecho analogo sucede para los otros tres casos manifestados en la tabla 3.

Lo anterior nos permite concluir que existe una "analogia", cuando se acepte la igualdad y no son literalmente iguales los terminos.

La estrategia 2 basa su funcionamiento en los programas, de la base de conocimiento, que realizan la funcion pedida pero usan otro tipo de argumentos, o bien, argumentos del mismo tipo pero que han sido rechazados por la evaluacion de la tabla 3. La tarea, en esta estrategia, consiste en generar los argumentos de alguno de estos programas, a partir de los argumentos del programa objetivo.

En el caso de la estrategia 2, SEA genera el sub-objetivo de encontrar un programa, que transforme los argumentos originales en los argumentos para otro programa. Observese como la generacion de sub-objetivos, lleva al planteamiento de nuevos programas a ser sintetizados, por lo que a su vez puede generar nuevos sub-objetivos, que ahora llamaremos sub-sub-objetivos, a partir de la estrategia 2.

En el caso de SEA, esta generacion de sub-objetivos tiene un limite.

El proceso de generacion de sub-objetivos genera algo que le llamaremos arbol de busqueda. Al problema original, que es la raiz del problema, le llamaremos nivel 0. El buscar un programa candidato para la sintesis del problema original, sera nivel 1. La tarea de encontrar un programa que genere los argumentos para un programa en nivel 1, a partir de los argumentos del problema

original, nivel 0, la llamaremos nivel 2. Podríamos seguir así indefinidamente.

SEA detiene su proceso de búsqueda a nivel 2, en el árbol de búsqueda descrito en el párrafo anterior.

Estrategia 1 . (Problemas tipo 1).

Este es el caso de pedir a la computadora que sintetice un programa que es "análogo" a otro que se encuentra en el banco de programas de SEA.

Decimos que el programa objetivo (a sintetizar) es análogo a otro que se encuentra en el banco de programas, si:

- El resultado y atributos del resultado del programa a sintetizar, son iguales al resultado (casillas 1 y 3, es decir, descripción corta del resultado) y atributos del resultado del programa en el banco de programas.

- Los argumentos del programa en el banco de programas de SEA (que contenga las casillas 1 y 3 iguales a las del programa a sintetizar y que además tenga todos los atributos del resultado que tiene el programa a sintetizar) puedan ser despejados, de las ecuaciones formuladas al igualar las casillas APL de los atributos de resultado correspondientes (14 + 5*j ; 14 + 6*k) del programa objetivo y del programa en el banco, en función de los argumentos del programa a sintetizar.

El problema tipo 1, al parecer, es el caso trivial de la P.A., sin embargo, encierra en sí mismo una riqueza muy grande en términos de capacidad de resolución de problemas.

Un ejemplo sencillo que muestra este primer tipo de problema es el siguiente:

Suponga que en la memoria de SEA existe un programa como el siguiente:

La suma de los escalares "que" y "a" da como resultado un escalar.

Vea en la tabla 4 la representación del programa anterior, mediante el uso de la tabla de información y como SEA lo reconoce por el nombre: SUMA.

Nótese como el caracter 220 del vector atómico del APL, aparece como una pequeña mancha en la tabla. Su función es de separador.

SUMA	DIRECCION DEL RESULTADO	A P L	CONDICION DE VALIDEZ
FORMA	ESCALAR		--------
COND. VALIDEZ	--------	A01+A02	
A P L	1	A01+A02	--------
CUALIDAD		C■1	

SUMA			
TIPO DE ATRIBUTO	F■A01	F■A02	
NOMBRE ATRIBUTO	QUE	A	
FORMA	ESCALAR	ESCALAR	
CONEXION-ANCLOR			
A P L	A01	A02	
CUALIDAD			

Tabla 4

Supongamos ahora que se desea obtener el codigo del programa cuya descripcion completa es la siguiente:

La suma de los escalares "que" y "a" que da como resultado un escalar, segun se describe en la tabla 5.

OBJE	DIMENSION DEL RESULTADO	A P L	CONDICION DE CALIDAD
FORMA	ESCALAR		--------
CONDICALIDAD	--------		
A P L			--------
CUALIDAD			

OBJE			
TIPO DE ATRIBUTO	F■A03	F■A04	
NOMBRE ATRIBUTO	QUE	A	
FORMA	ESCALAR	ESCALAR	
CONEXION			
A P L	A03	A04	
CUALIDAD			

TABLA 5

Cuando SEA trata de sintetizar ese programa, se genera el sistema de ecuaciones mostrado en la siguiente pagina, para la solucion del problema.

En los sistemas de ecuaciones que SEA genera, se utilizan las siguientes modificaciones a la sintaxis matematica tradicional:

- El signo de igual en notacion matematica se sustituye por el signo: ←, que corresponde a la notacion APL.

- Cuando una expresion aparezca de la siguiente forma: S##(expresion), indica que las variables contenidas en la expresion pertenecen a la familia de variables ##. Por ejemplo, S01(A01) y S02(A01)(A02) equivale a las variables A01 de la familia 01 y las variables A01 y A02 de la familia 02. Es decir S01(A01) es diferente a S02(A01). Esto se debe a que cuando se formulan ecuaciones entre los atributos del resultado de los programas, A01 en un programa puede tener un significado distinto de A01 en otro programa. Cuando una expresion aparece sin el formato S##(expresion), por ejemplo A01, indica que la expresion se refiere a variables del programa objetivo y que pertenecen a la familia de variables 00. Las variables del programa objetivo son sus argumentos.

Una familia de variables, corresponde a un nivel en el arbol de busqueda. Por ejemplo, la familia de variables 01, representada por S01, corresponde al nivel 1 en el arbol de búsqueda.

Como hemos dicho anteriormente, las variables de las familias 01 y 02 deben ser despejadas en funcion de las variables del programa objetivo, o familia 00.

Una vez que se han despejado las variables pedidas y se le ha indicado a la computadora el valor de cada una, aparecera en la pantalla el codigo del programa que ejecuta la funcion deseada.

El programa ya sintetizado tiene su propia tabla de representacion, Tabla 6.

```
A04 ← → SOL(A02)
ESCALAR
A

A03 ← → SOL(A01)
ESCALAR
QUE


RESTRICCIONES
SOL()

VARIABLES A DESPEJAR:

SOL(A01)■SOL(A02)■


TIENE SOLUCION (S/N)?
S
TECLEE UNA A UNA LAS VARIABLES PEDIDAS
SOL(A01)
A03
SOL(A02)
A04
(A03,A04)
```

OBJE	[illegible]	A F L	[illegible]
FORMA	ESCALAR		--------
CONDICIONALIDAD	--------		■
A F L		(A03)*(A04)	--------
CUALIDAD		C■1	

OBJE	F■403	F■404	
TIPO DE ATRIBUTO			
NOMBRE ATRIBUTO	QUE	A	
FORMA	ESCALAR	ESCALAR	
CONDICIONALIDAD			
A F L	A03	A04	
CUALIDAD			

Tabla 6

Ahora bien, la estrategia tipo 1 dice que se debe buscar en la memoria de SEA todos aquellos programas cuyas casillas 1 y 3 sean iguales a las casillas 1 y 3 del programa objetivo. Es decir, se está buscando a todos aquellos programas cuya descripcion corta del resultado coincida con la descripcion corta del resultado del programa objetivo. A todos los programas en la memoria de SEA que cumplan con esta caracteristica se les denominará CONJUNTO A.

Si el conjunto A es vacio, habrán fallado las estrategias 1 y por lo tanto la estrategia 2. Esto se debe a que la estrategia 2 usa un sub-conjunto del CONJUNTO A de programas para tratar de sintetizar el programa objetivo.

Si el conjunto A no es vacio, se procede a encontrar el conjunto B a partir de los programas del conjunto A, en los que todos los atributos del resultado del programa objetivo se encuentran tambien en ellos. Con el hecho de que un atributo del resultado del programa objetivo se encuentre tambien en un programa del conjunto A, queremos decir que ambos atributos tienen el nombre identico, la forma identica (escalar, vector, etc.), que se puede establecer una analogia entre ellos (en las casillas 14 + k*6) de acuerdo a la tabla constante variable descrita antes (tabla 3) y que tiene la cualidad idéntica.

Si el conjunto B es vacio, habrán fallado las estrategias 1 y 2, debido a que la estrategia 2 se basa en la existencia del conjunto B.

Si el conjunto B no es vacio, se procede a establecer un conjunto de ecuaciones entre los atributos del resultado del programa a sintetizar con sus correspondientes atributos del resultado del programa candidato, con lo que se pretende despejar los argumentos del programa candidato en terminos de los argumentos del programa a sintetizar. Esto se debe a que los argumentos del programa a sintetizar son conocidos, o bien se contara con ellos durante la ejecucion del programa.

La solucion de este sistema de ecuaciones esta sujeto a las restricciones mencionadas en la casilla 8 del programa candidato, al igual que a los nombres, formas y cualidades de cada uno de los argumentos de ambos programas.

En caso de que el sistema de ecuaciones tenga solucion, SEA sustituira el valor de cada uno de los argumentos, del programa candidato, en el codigo (casilla 6) y en las restricciones (casilla 8), ademas de copiar el origen de indices (casilla 7) para el cual el programa candidato funciona.

Si se ha llegado hasta este punto y el sistema de ecuaciones no tiene solucion, se puede recurrir al uso de la estrategia tipo 2.

Problemas tipo 2. (Estrategia tipo 2).

Supongamos que SEA entiende que tipo de problema debe resolver, esto es que existe en el banco de SEA un conjunto no vacío (conjunto B de la estrategia 1) de programas cuyos resultados y atributos del resultado son los mismos que el programa objetivo. Sin embargo, este conjunto ya fue excluido del proceso de síntesis mediante la estrategia tipo 1.

La estrategia que caracteriza a los problemas tipo 2, consiste en Generar el sub-objetivo de encontrar uno de los argumentos (con sus atributos) del programa candidato del banco.

Como cada sub-objetivo es en sí un objetivo, limitaremos el "nivel de inteligencia" de SEA permitiendo exclusivamente un nivel 2 de profundidad en el árbol de búsqueda de solución (considerando que el nivel inicial es nivel 0), como ya fue explicado antes.

En la generación del sub-objetivo, se forma una tabla de información para el nuevo objetivo. Por ejemplo, si el argumento que se pretende encontrar, se encuentra en las casillas de la 10 a la 15, en la generación del objetivo la casilla 12 se convierte en la casilla 1, la casilla 15 en la casilla 3 y la casilla 14 se reserva para ser igualada con el código resultante y añadirse, esta ecuación, al conjunto de ecuaciones de solución del problema. Si el argumento, que es el sub-objetivo, tiene atributos, estos se convierten en atributos de resultado.

El desarrollo del sub-objetivo consiste en encontrar los conjuntos A y B descritos para el caso de la estrategia 1. En caso de que el conjunto b sea vacío, se tratara con otro argumento del programa candidato en nivel 1. En caso de que el conjunto B sea no vacío, se intentara resolver el conjunto de ecuaciones de la estrategia 1 junto con las ecuaciones que resulten de igualar los atributos del resultado del programa en el nivel 2 con los atributos del argumento del programa en el nivel 1.

Este último sistema de ecuaciones estara sujeto a las restricciones del programa del nivel 1 y a las restricciones del programa del nivel 2.

En caso de tener éxito, el resultado consiste en sustituir los valores de los argumentos del programa en nivel 1 en la casilla 6 del programa en nivel 1; sustituir los mismos datos en la casilla 8 del programa en nivel 1; y agregar a la casilla de restricciones (8) del nivel 1 la casilla de restricciones (8) del nivel 2 cuando se le han sustituido los valores de los argumentos del programa en nivel 2.

En caso de no lograrlo se considerara fracaso del de la estrategia 2 y por lo tanto del sintetizador.

Una restriccion adicional, en esta estrategia, es que SEA encuentra la interseccion de los conjuntos: Casilla 7 del programa en nivel 1, con la casilla 7 del programa en nivel 2. Es decir, los origenes de indices para los cuales ambos programas funcionan adecuadamente. Si esta interseccion es vacia, el proceso de sintesis falla. Si la interseccion no es vacia, esta sera el valor que tenga la casilla 7 del programa objetivo ya sintetizado.

Un ejemplo sencillo que muestra este segundo tipo de problema es el siguiente:

Suponga que en la memoria de SEA existen los siguientes dos programas:

1. A partir de un escalar entero, A01, genera un vector de numeros enteros sucesivos desde 1 hasta A01. Este programa, SEA lo reconoce por el nombre GENERAR. Tabla 7.

2. A partir de la suma de un vector de numeros enteros sucesivos mas un escalar entero, se obtiene un vector de numeros enteros sucesivos. SEA lo denomina SUMAR. Tabla 8.

Supongamos ahora que se desea obtener el codigo del programa cuya descripcion completa es la que aparece en la tabla 9 en donde se busca sintetizar un programa que a partir de un escalar entero, DESDE (A01), otro escalar entero, HASTA (A02), se genere un vector de numeros enteros sucesivos desde A01 hasta A02.

GENERAR	DIMENSION DEL RESULTADO	A P L	CONDICION DE VALIDEZ
FORMA	VECTOR		--------
CONDICION	--------	\A01	A01 /0
A P L	A01	\A01	--------
CUALIDAD	ENTERO SUCESIVO	1	

GENERAR			
TIPO DE ATRIBUTO	F A01	E	F
NOMBRE ATRIBUTO	CANTIDAD	DESDE	HASTA
FORMA	ESCALAR	ESCALAR	ESCALAR
CONEXION			
A P L	A01	1	A01
CUALIDAD	ENTERO	ENTERO	ENTERO

Tabla V

SUMAF	DIMENSION DEL RESULTADO	A F L	CONDICION DE CALIDEZ
FORMA	VECTOR		--------
CONDVALIDEZ	--------	A01*A02	
A F L	A#02	A01*A02	--------
CUALIDAD	ENTERO SUCESIVO	1■0	

SUMAR			
TIPO DE ATRIBUTO	A02	A02	A02
NOMBRE ATRIBUTO	A	DESDE	HASTA
FORMA	VECTOR	ESCALAR	ESCALAR
CONFIGU-RACION			
A F L	A02	V1	V2
CUALIDAD	ENTERO SUCESIVO	ENTERO	ENTERO

Tabla 3

SUMAF			
TIPO DE ATRIBUTO	A01	F	F
NOMBRE ATRIBUTO	QUE	DESDE	HASTA
FORMA	ESCALAR	ESCALAR	ESCALAR
CONFIGURACION			
A P L	A01	A01+U1	A01+U2
CUALIDAD	ENTERO	ENTERO	ENTERO

Tabla 8

OBJE	DIMENSION DEL RESULTADO	A P L	CONDICION DE VALIDEZ
FORMA	VECTOR		--------
COND VALIDEZ	--------		
A P L			--------
CUALIDAD	ENTERO SUCESIVO		

OBJE			
TIPO DE ATRIBUTO	F■A01	F■A02	
NOMBRE ATRIBUTO	DESDE	HASTA	
FORMA	ESCALAR	ESCALAR	
CONFIGURACION			
A P L	A01	A02	
CUALIDAD	ENTERO	ENTERO	

Tabla 9

En nuestro ejemplo, se generara un primer conjunto de ecuaciones que no tendra solucion. Este primer conjunto de ecuaciones manifiesta el intento de SEA por sintetizar el programa mediante la estrategia 1. En este intento, se tiene por programa candidato, al programa SUMAR, debido a que la descripcion corta y atributos del resultado son iguales a los del programa objetivo.

El programa GENERAR no ha sido considerado como candidato, debido a que, el atributo de resultado DESDE, aunque es ESCALAR y es ENTERO, es constante, por lo que al ser sometido a la prueba de la tabla 3, se rechaza la igualdad de atributos.

En el momento que se falla en la solucion de ecuaciones, SEA recurre a la estrategia 2 por lo que, en este ejemplo, se presenta un segundo conjunto de ecuaciones que si tienen solucion.

El segundo conjunto de ecuaciones se genero a partir de considerar el programa SUMAR como el programa seleccionado en nivel 1, y el programa GENERAR como el programa del nivel 2. Este ultimo se selecciono al tratar de generar uno de los argumentos del programa SUMAR. Es decir, el argumento Vector Entero Sucesivo.

A continuacion se muestran los conjuntos de ecuaciones que aparecen y las respuestas que el usuario dio a SEA ante la solicitud para resolverlos. La tabla 10 indica el resultado final de la sintesis del programa. Obsérvese en especial las casillas 6, 7 y 8, que contienen el fruto del proceso de sintesis.

```
A02 ← → SO1(A01+V2)
ESCALAR
HASTA
ENTERO
A01 ← → SO1(A01+V1)
ESCALAR
DESDE
ENTERO

RESTRICCIONES
SO1( )

VARIABLES A DESPEJAR:

SO1(A02)■SO1(A01)■


TIENE SOLUCION (S/N)?
N
```

```
A02 ← → S01(A01+V2)
ESCALAR
HASTA
ENTERO
A01 ← → S01(A01+V1)
ESCALAR
DESDE
ENTERO

S01(A02) ← → S02(⍳A01)
ENTERO SUCESIVO
VECTOR
S01(V2) ← → S02(A01)
ESCALAR
HASTA
ENTERO
S01(V1) ← → S02(1)
ESCALAR
DESDE
ENTERO

VARIABLES A DESPEJAR:

S01(A02)■S01(A01)■
S02(A01)■

RESTRICCIONES
S01(⍳)
S02(A01)>0)


TIENE SOLUCION S/N?
S
TECLEE UNA A UNA LAS VARIABLES PEDIDAS
S01(A02)
⍳A02+1-A01
S01(A01)
A01-1
TECLEE UNA A UNA LAS VARIABLES PEDIDAS
S02(A01)
A02+1-A01
(⍳A01-1)+(⍳A02+1-A01)
```

OBJE	DIMENSION DEL RESULTADO	A P L	CONDICION DE VALIDEZ
FORMA	VECTOR		--------
COND VALIDEZ	--------		(A02+1-A01)>0
A P L		(A01-1)+⍳(A02+1-A01)	--------
CUALIDAD	ENTERO SUCESIVO	⍳1	

OBJE			
TIPO DE ATRIBUTO	F⍳A01	F⍳A02	
NOMBRE ATRIBUTO	DESDE	HASTA	
FORMA	ESCALAR	ESCALAR	
CONEXION-FUNCION			
A P L	A01	A02	
CUALIDAD	ENTERO	ENTERO	

Tabla 10

Un hecho de importancia es que tanto en los problemas tipo 1 como en los problemas tipo 2 notamos que sería de gran utilidad el contar con la capacidad de procesamiento en paralelo. Es decir que cada uno de los elementos del conjunto B, mencionado antes, debe ser probado como candidato para la solución del problema. De tal forma que el ahorro en tiempo es considerable , SE= podría aspirar a seleccionar una solución optima cuando hubiera mas de una solución factible.

Esto último requeriría de un criterio para seleccionar un programa en lugar de otro.

III. Funcionamiento Interno de SEA.

SEA es un sintetizador que se encuentra actualmente funcionando y que resuelve los problemas tipo 1 y problemas tipo 2 descritos en el capitulo anterior. Abajo mencionamos los supuestos de SEA, la mecanica de su funcionamiento y algunas de las dificultades encontradas durante su creacion.

Supuestos

A continuacion se mencionan los supuestos que rigen el comportamiento de SEA y las convenciones de especificacion de un programa.

Dentro de este conjunto de supuestos, estan contenidas las convenciones de especificacion de un programa, mencionadas en el capitulo 1. Se les menciona nuevamente aqui, porque forman parte de los supuestos de funcionamiento de SEA.

Supuestos generales.

- Toda idea en el mundo real debe tener solo una representacion en el mundo interno de SEA, y toda representacion en el mundo interno de SEA, corresponde, solo, a una idea en el mundo real.

- Siempre habra descripciones completas de programas. Es decir, todo programa existente en el banco de informacion de SEA es independiente de todas las demas en el sentido de que no se tiene que efectuar la resolucion de ningun problema para completar su definicion, y ademas, toda la informacion que lo caracteriza se encuentra especificada en su tabla de informacion.

- Si se presenta el problema de sintetizar un programa que debe ejecutar uno de entre varios procedimientos, dependiendo del resultado de una o mas restricciones, estaremos hablando de una especificacion de programa del tipo de procedimiento, por lo que se debe alimentar a SEA con un programa que funcione con las condiciones pedidas y la invocacion de los procedimientos correspondientes (siempre y cuando no existe ya en la memoria) y al tener el objetivo de resolver ese problema, lo que ocurrira que la maquina se dedique a la resolucion de los procedimientos a manera de sub-objetivos. Al tratar de resolver este tipo de problemas se debe recordar la limitante actual de SEA, que solo le permite resolver un sub-objetivo de manera global.

- Cuando SEA este verificando la igualdad entre dos formas de atributos (o busque un programa con algunas caracteristicas) no solo verificara que la estructura (escalar, vector, etc..) sea igual sino que tambien el valor de la casilla Forma sea igual. Cuando ambas Formas sean iguales y sean variables, se agregara esta igualdad al conjunto de ecuaciones a ser resueltas en el proceso de busqueda y resolucion descrito en el capitulo anterior. Este supuesto esta pendiente de ser implementado en SEA.

- Es responsabilidad del usuario, el alimentar a SEA con programas que sean "verdaderos" y no contradictorios con otros programas. Con esto se garantizara, que los resultados de SEA seran tambien verdaderos y no contradictorios.

Supuestos de especificacion de programas.

- Todos los argumentos deben aparecer en la casilla 6, de lo contrario no serian argumentos.

- Cada atributo, ya sea de resultado o de argumento, está representado por un solo juego de seis casillas.

- En un mismo programa no puede haber dos argumentos distintos que sean iguales en el tipo de argumento ni en su nombre.

- Debe haber alguna información en las casillas 1 y 3 de todo programa, ya sea que se vaya a sintetizar o este ya sintetizado. En el caso de un programa ya sintetizado, la casilla 3 debe contener el codigo del programa. El resto de las casillas pueden permanecer vacias si asi lo requiere la descripcion completa.

- Cuando se este especificando un programa para que sea sintetizado, se dejaran vacias las casillas 4 a la 9.

- En las casillas que se deba especificar un valor predeterminado. ESCALAR, VECTOR, etc., solo puede haber alguno de esos valores. Esto sucede en las casillas 1,12,18,....

- Es responsabilidad del usuario el no poner atributos de un mismo argumento, o de resultado, que sean contradictorios.

- Cuando deba especificarse una expresión en AFL, como en las casillas 5, 10, 20, 26, ..., se debera respetar la sintaxis del lenguaje AFL. Dentro de estas expresiones, solo podra haber argumentos de la forma ARN.

- Las casillas que deban ser llenadas con frases en lenguaje natural, no tienen restriccion alguna en cuanto a su contenido.

- Los identificadores de argumento deben ser de la forma A##, donde # es un digito del conjunto (0,1,2,3,4,5,6,7,8,9).

- En el caso de los argumentos, deben tener un tipo de atributo (casilla 10,16,...) como el siguiente: R;A##. La "R" aparece debido a que todo argumento es un atributo del resultado. El ";" representa, por ahora, el caracter numero 220 del vector atomico del APL, tiene la funcion de separador. La "A" seguida de dos digitos representa al identificador del argumento.

- No puede haber dos argumentos distintos con el mismo identificador "A##".

- Los atributos de argumento deben estar en las casillas inmediatas siguientes al argumento que pertenecen y su tipo de atributo debe ser el mismo "A##" que se uso como identificador del argumento. Es decir, el siguiente conjunto de seis casillas se referira al atributo de argumento y llevara el mismo tipo de atributo, es decir identificador de argumento, (casilla 10 + o*k), que el argumento, A##.

- Un atributo, no puede ser al mismo tiempo atributo de dos argumentos o atributo de resultado y de argumento.

- La casilla APL de todo argumento (14, 20, ...) debe denominarse con una variable con el mismo identificador del argumento, es decir, con la forma A##.

- Cuando un atributo de argumento tiene un valor desconocido para la casilla APL (14, 20, ...), se denominara, en esta misma casilla con alguna de las variables V1, V2, ..., o V9. Cuando un mismo argumento tenga mas de dos atributos con estas caracteristicas, se usara otra variable distinta de este mismo conjunto mencionado. Este tipo de variables no esta restringido, en cuanto a que sea constante o variable.

- El origen de indices puede tomar el valor de 1, o 0 o ambos, en la casilla 7. En el caso que tome ambos valores, significa que el programa funciona bien para ambos casos. Cuando los dos valores aparecen, estaran separados por el caracter 220 del vector atomico.

- Al especificar un programa, las restricciones deberán estar separadas, en la casilla 8, por el caracter 220 del vector atomico del AFL.

Mecánica del funcionamiento de SEA

La forma en como SEA trata de sintetizar un programa, tiene la siguiente secuencia:

- Permite que el usuario de entrada a las especificaciones del programa que quiere que SEA sintetice.

- Verifica que las especificaciones descritas cumplan las reglas de sintaxis.

- Comienza el proceso de resolucion mediante la busqueda en el banco de SEA de todos aquellos programas cuyo resultado y atributos de este sean iguales a los pedidos en las especificaciones del programa.

- Si el conjunto de programas encontrado es no vacio, procede a verificar si el problema que se tiene es del tipo 1, es decir si se puede aplicar la estrategia 1 para su solucion.

- Si la estrategia 1 fallo, se procede a aplicar la estrategia 2.

- Si no se tiene exito se habrá facasado en sintetizar el programa, de lo contrario aparecera el codigo del programa deseado en la pantalla.

Dificultades encontradas

La dificultad mas grande fue el definir un procedimiento que de alguna manera llevara en forma segura a un resultado, cuando habia la informacion necesaria para conseguirlo.

Durante la elaboracion de ese procedimiento, fue importante detectar los elementos que son cruciales en la sintesis de un programa, mediante el enfoque de esta tesis.

Esos elementos son:

- Contar con una representacion de programas que fuera lo suficientemente completa, sistematica y flexible, para permitir que el proceso de sintesis se lograra y fuera lo menos restrictiva posible. La tabla de informacion mostrada antes, es una solucion a este problema. No obstante que cumple los requisitos pedidos, es complejo el proceso de alimentacion de datos.

- Definir con precision, el estado interno de SEA, para el cual se considera que el problema ha sido "entendido". La solucion a este problema se da cuando SEA encuentre un programa, en su banco de programas, que tiene la misma descripcion corta y atributos del resultado que el programa objetivo.

- Conocer claramente, en que momento SEA habia alcanzado la sintesis de un programa. Para tal efecto, se decidio que SEA planteara ecuaciones entre atributos de resultado de programas semejantes, a partir de concordancias de expresiones en lenguaje natural. El resolver con exito, estas ecuaciones, garantizaria, bajo ciertos supuestos, el haber alcanzado la solucion.

- Implementar en SEA, un procesador de terminos algebraicos, en notacion APL, que permitiera resolver las ecuaciones generadas durante el proceso de sintesis. Este problema no fue resuelto de acuerdo a su planteamiento inicial, sino que se opto, por pedir al usuario, que interviniera en el proceso de sintesis, sustituyendo al procesador de terminos algebraicos. Es decir, actualmente SEA formula las ecuaciones correspondientes y el usuario decide si tienen o no solucion. En caso de existir solucion, el usuario debe proporcionar los valores de las variables pedidas.

- Dotar a SEA, de cierta capacidad de sintesis, ante problemas no triviales. La respuesta a esta dificultad fue el programar la habilidad de generar sub-objetivos, ampliando un poco mas el alcance de solucion de problemas.

IV. Conclusiones

En terminos generales, se espera que un sintetizador automático de programas, sea capaz de resolver 'cualquier tarea' que un programador humano sea capaz de resolver.

Para lograr lo anterior, un sintetizador deberia poseer un amplio acervo de terminos en su vocabulario de lenguaje natural, un diccionario de sinonimos relacionado al tema de los programas que se pretenda sintetizar, una gama amplia de estrategias de solucion de problemas de programacion, una amplia variedad de formatos para expresar los resultados y quiza, algun conocimiento especializado, en alguna area en particular, en la que se le tenga por experto.

Lo anterior se refiere al conocimiento interno. En cuanto a su capacidad de interactuar con el usuario, deberia tener un interprete de lenguaje natural para poder traducirlo en terminos entendibles para la computadora, que fuera capaz de resolver dudas o ambigüedades, que en un lenguaje comun entre dos personas pueden surgir.

SEA no pretende dar una solucion a cada uno de los problemas de P.A. manifestados arriba, sino que, al igual que muchos autores, nos hemos abocado a hacer una aportacion en tres areas en particular:

- La representacion de conocimiento.
- Las estrategias de solución aqui consideradas.
- La base de conocimientos existente, previa a la sintesis de un programa.

Por tanto hemos tenido que dar soluciones parciales e ineficientes al resto de los problemas, que se salen del alcance de esta tesis.

Hemos considerado solo dos estrategias en la solucion del problema de P.A.; sin embargo, no descartamos que pueda haber muchas otras.

Esta tesis mostro la importancia de tener volumenes amplios de información, ya que esto aumenta la gama de problemas que un sintetizador puede resolver.

Observamos que nuestro trabajo, al igual que el de muchos autores, ha presentado novedades, en el tratamiento de algunos de los problemas, que se presentan en el contexto general de la Programacion Automatica.

Sin embargo, es notoria la baja productividad de SEA, al quedar parcialmente resueltos el resto de los problemas.

No nos queda duda, que el problema de Programación Automatica no es un problema Aislado del resto de los tópicos de la Inteligencia Artificial.

Se han realizado ya, muchos trabajos que enfoquen su esfuerzo en áreas particulares de la programacion automatica. En el futuro, seguramente se integraran los esfuerzos de los investigadores en las distintas áreas de la Inteligencia Artificial para producir resultados de mayor alcance.

Algunos temas han salido del ámbito de nuestra especialidad y los hemos querido dejar para futuras investigaciones. Ellos son:

- Ampliar el campo de interpretacion de una descripcion completa para hacerla relativa al contexto en que se encuentre. Actualmente los supuestos definen una sola interpretación para cada descripcion completa de conocimiento.

- Generacion de sub-objetivos multiples, simultaneamente. Actualmente se maneja un solo subobjetivo a la vez.

- Profundizar en el árbol de busqueda a un nivel superior a 2.

- La creacion de un procesador de términos algebraicos.

Esta tesis es una muestra de que la P.A. es una realidad y que las futuras generaciones humanas no podran prescindir de ella, ya que la imaginacion del ser humano no deberia detenerse en problemas tan molestos, como el programar una computadora, y dedicarse así al estudio de temas de orden superior.

Apendice A.

```
      )FNS
BORRA   CARATULA          CLS     DEPURA  ESTA    FORMA   IGUAL   IMPRIME LEE
LISTA   LISTAZ  ORIGEN  POF     PON     POSICION          PROGRAMAS
REEMPLAZA       REGLA1  REGLA11 REGLA12 REGLA121          SEA     SOLUCION
SUEVAR  SUJETOA TABLACV VARIABLE

      ∇SEA[⎕]∇
[0]  Z←SEA;Z
[1]   →2+3×'S'=⍞INKEY ⍝ ⎕←'YA TECLEO SU OBJETIVO (S/N)?' ⍝ CARATULA
[2]   'OBJE' LEE ⎕⍝ ⎕←'CUANTAS CASILLAS TIENE SU OBJETIVO'
[3]   →4+1×∧/'SI'=⍞⍝ ⎕←'ESTAN TODAS CORRECTAS (SI O NO)?'
[4]   →3 ⍝⍎'OBJE',(⍕⎕⍝ ⎕←'TECLEE EL NUMERO DE CASILLA'),'←',⍞⍝ ⎕←'TECLEE EL DATO CORRECTO
[5]   PON ◊ ⎕←Z←REGLA1 'OBJE' ◊ POF

      ∇BORRA[⎕]∇
[0]  BORRA A;A;B;C;D;E;F;G;H;I
[1]   →2+1≠'OBJE' IGUAL A
[2]   →6 ⍝ NUMOBJE←0 ⍝ C←NUMOBJE ⍝ B←A
[3]   →4+∧/('A'=1↑A),(4=⍴A),(B←¯3↑A)∊'0123456789O'
[4]   →30 ⍝ ⎕←'LA FUNCION DEBE SER DE LA FORMA Annn O BIEN LLAMARSE OBJE'
[5]   D←⍎A ⍝ C←NUM[⍎B]
[6]   →E×0=C←C-1 ⍝ ⎕EX D,⍕C ⍝ D←B,⍕C
[7]   →8+20×'OBJE' IGUAL A
[8]   F←G←0,2×⍴NOM,E←''
[9]   →9×0=F←F-1 ⍝ E←E,(16×G-F)+1+5×¯1+⍳4
[10]  I←F←(+/(G,4)⍴(,((G,5)⍴NOM)[;⍳4]∘.=A)[E])⍳4
[11]  →12×F=G
[12]  →11 ⍝ A←H ⍝ F←F+1 ⍝⍎A,'←',H←NOM[(5×F)+⍳4]
[13]  ⎕EX NOM[(5×G-1)+⍳4]
[14]  NUM←NUM[⍳I-1],NUM[I+⍳G-I]⍝ NOM←¯5↓NOM

      ∇CARATULA[⎕]∇
[0]  Z←CARATULA
[1]   0 0 20 80 ⎕WPUT FACE ⍝ ⎕CURSOR← 24 0 ⍝ ⎕←⍝l
[2]   ⎕SOUND((50 1 ⍴(50⍴ ¯1 1)×10×⍳50), 50 2 ⍴0)← 50 3 ⍴ 10 51 2000
[3]   Z←'' ⍝ ⎕←⍝l

      ∇CLS[⎕]∇
[0]  Z←CLS;Z
[1]   ⍝l
[2]   Z←''

      ∇DEPURA[⎕]∇
[0]  Z←DEPURA L;J
[1]   Z←''
[2]  ⍝ QUITA LOS ELEMENTOS PARES DE L
[3]  LOOP:→(0=⍴L)/FIN
[4]   →LOOP ⍝ L←J↓L ⍝ J←L⍳⎕AV[220]⍝ L←J↓L ⍝ Z←Z,L[⍳¯1+J←L⍳⎕AV[220]],⎕AV[220]
[5]  FIN:

      ∇ESTA[⎕]∇
[0]  Z←A ESTA B;Z;A;B;C
[1]   →2+∨/0=(+/B=A),+/B=B
[2]   →4 ⍝ Z←∨/C=⍴⍴(¯1+⍳C←+/B=A)⌽A∘.=B,' '
[3]   Z←A IGUAL B

      ⎕TCFF
```

```
      ∇FORMA[⎕]∇
[0]  Z←A FORMA B;A;B;Z
[1]   →2↑20/Z←'ARREGLO' IGUAL B
[2]   Z←A IGUAL B

      ∇IGUAL[⎕]∇
[0]  Z←A IGUAL B;Z;A;B;C
[1]   →2↑(C←+/A=A)≠+/B=B ⋄ Z←0
[2]   Z←C=+/A=B

      ∇IMPRIME[⎕]∇
[0]  C IMPRIME A;B;A;C;D
[1]   B←A
[2]   →2↑0=A←A-1 ⋄ ⎕←(⍕D),' ',⍕C,B[1+B-A]

      ∇LEE[⎕]∇
[0]  A LEE B;A;B;C;D
[1]   →6-4×((C-9)↑6)=⍳(¯9↑C←B)↑6
[2]   →2↑0=B←B-1 ⋄⍎A,D,'←',''''',⎕,'''' ⋄ ⎕←'TECLEE EL ELEMENTO ',D←B1↑C-B
[3]   →4↑A IGUAL 'OBJE'
[4]   →6 ⋄ NUM←,NUM,C ⋄⍎D,'←A' ⋄ NOM←NOM,(D←'A',((1↑10)⍕D)↑'OO'),D ⋄ D←B1↑0,2×⍴NOM),':'
[5]   NUMOBJE←C

      ∇LISTA[⎕]∇
[0]  LISTA;Z;CASILLAS;MAT;C;TEMP;CLAVE;IMP;D;ESP
[1]   Z←⍞ ⎕←'TECLEE EL NOMBRE DE LA FUNCION (OBJE, Annn)' ⋄ ⎕←⍳1
[2]   →3↑3×Z IGUAL 'OBJE' ⋄ ESP←' '
[3]   →1↑3×∧/('A' IGUAL 1↑Z),(¯3↑Z)∊'1234567890'
[4]   Z←⍎Z ⋄ CASILLAS←NUM[⍎¯3↑Z]
[5]   →7
[6]   CASILLAS←⍎'NUM',Z
[7]   MAT[80+⍳6]←8↑Z ⋄ MAT←,MATRIZ1 ⋄ CLAVE← 327 1117 1512 350 745 1140 1535 766 1552
[8]   ⍎(3×¯1+2×'S'=IMP)↑'POR    ' ⋄ C←10 ⋄ IMP←DINRET ⋄ ⎕← LO QUIERE POR IMPRESORA (S/N)?'
[9]   →10↑2×0=C←C-1
[10]  TEMP←68↑((22×67)⍴' ',⍕Z,⍕C),↑' '),⍕Z,⍕C
[11]  →11-2×0=⍴TEMP←22↓TEMP ⋄ MAT[(CLAVE[C]+79×4-(⍴TEMP)÷22)+¯1+⍳22]←22↑TEMP
[12]  ⍎((15×¯1+2×'S'=IMP)↑'                D←0 0 24 79 OUTPUT'),' 24 79 ⍴MAT' ⋄ OCURSOR← 24 0 ⋄ ⎕←⍳1
[13]  CLAVE← 50 485 880 1275 1670 2065 113 508 303 1298 1693 2088 136 531 926 1321 1716 2111
[14]  C←0
[15]  ⍎(7×¯1+2×'S'=IMP)↑'D←D←ESP DINRET'
[16]  →17↑5×CASILLAS[C+9
[17]  MAT[80+⍳8]←8↑Z ⋄ MAT←,MATRIZ2 ⋄ D←1+CASILLAS[C+27
[18]  →19↑2×(C+9)=D←D-1
[19]  TEMP←86↑((22×67)⍴' ',⍕Z,⍕D),↑' '),⍕Z,⍕D
[20]  →20-2×0=⍴TEMP←22↓TEMP ⋄ MAT[(CLAVE[D-9+C]+75×4-(⍴TEMP)÷22)+¯1+⍳22]←22↑TEMP
[21]  →16 ⋄ C←C+18 ⋄⍎(7×¯1+2×∧/('S'=IMP),(C+36)=⍳C+36)↑'   D←⍳1G←D←ESP' ⋄ ⎕← 31 79 ⍴MAT
[22]  ⍎(3×¯1+2×'S'=IMP)↑'POR    '

      ⎕TOFF
```

```
      ∇LISTA2[⎕]∇
[0]   LISTA2;Z;CASILLAS;MAT;C;TEMP;CLAVE;IMP;D;ESP
[1]    Z←⍞ ⋄ ⍞←'TECLEE EL NOMBRE DE LA FUNCION (OBJE, Annn)' ⋄ ⍞←⍕1
[2]    →3+3×Z IGUAL 'OBJE' ⋄ ESP←' '
[3]    →1+3×∧/('A' IGUAL 1↑Z),(¯3↑Z)∊'1234567890'
[4]    Z←⍎Z ⋄ CASILLAS←NUM[⍎¯3↑Z]
[5]    →7
[6]    CASILLAS←⍎'NUM',Z
[7]    MAT[80+⍳8]←8↑Z ⋄ MAT←,MATRIZ1 ⋄ CLAVE← 327 1117 1512 350 745 1140 1535 768 1556
[8]    ⍎(3×¯1+2×'S'=IMP)↑'PON    ' ⋄ C←10 ⋄ IMP←DINKEY ⋄ ⍞←'LO QUIERE POR IMPRESORA (S/N)?'
[9]    →10+2×0=C←C-1
[10]   TEMP←38↑((22×67)⍴' ',⍕Z,⍕C)↑' '),⍕Z,⍕C
[11]   →11-2×0=⍴TEMP←22↓TEMP ⋄ MAT[(CLAVE[C]+79×4-(⍴TEMP)÷22)+¯1+⍳22]←22↑TEMP
[12]   ⎕←(24 15 ⍴' '), 24 79 ⍴MAT ⋄ ⎕CURSOR← 24 0 ⋄ ⍞←⍕1
[13]   CLAVE← 90 485 880 1275 1670 2065 113 508 903 1298 1693 2088 136 531 926 1321 1716 2111
[14]   C←0
[15]   ⍎(7×¯1+2×'S'=IMP)↑'⎕←⎕←ESP DINKEY'
[16]   →17+5×CASILLAS[C+7]
[17]   MAT[80+⍳6]←8↑Z ⋄ MAT←,MATRIZ2 ⋄ D←1+CASILLAS[C+27]
[18]   →19+2×(C+9)=D←D-1
[19]   TEMP←38↑((22×67)⍴' ',⍕Z,⍕D)↑' '),⍕Z,⍕D
[20]   →20-2×0=⍴TEMP←22↓TEMP ⋄ MAT[(CLAVE[D-9+C]+79×4-(⍴TEMP)÷22)+¯1+⍳22]←22↑TEMP
[21]   →16 ⋄ C←C+18 ⋄⍎(7×¯1+2×∧/('S'=IMP),(C+36)=(C+36))↑'   ⎕←⍕1⎕←⎕←ESP' ⋄ ⎕←(31 15 ⍴' '), 31 79 ⍴MAT
[22]   ⍎(3×¯1+2×'S'=IMP)↑'POF    '

      ∇ORIGEN[⎕]∇
[0]   Z←A ORIGEN B
[1]    Z←'' ⋄ A←⍕(⍕A),⍕7 ⋄ B←⍕(⍕B),⍕7
[2]    Z←Z,(2×∧/('0' ESTA A),'0' ESTA B)↑'0',⎕AV[220]
[3]    Z←Z,(2×∧/('1' ESTA A),'1' ESTA B)↑'1',⎕AV[220]

      ∇POF[⎕]∇
[0]   r←POF
[1]    r←'' ⋄ 0 ⎕POKE 116
[2]   ⍝POF:'' ⋄ 0 ⎕POKE 116

      ∇PON[⎕]∇
[0]   r←PON
[1]    r←'' ⋄ 1 ⎕POKE 116
[2]   ⍝PON:'' ⋄ 1 ⎕POKE 116

      ∇POSICION[⎕]∇
[0]   Z←A POSICION B;A;B;Z;C
[1]    →2+∨/0=(+/A=A),+/B=B
[2]    →4 ⋄ Z←(C=+⌿((⍴,A),⍴,B)⍴(¯1+⍳C←+/A=A)⌽A∘.=B)⍳1
[3]    Z←1+⍴B
[4]    Z←Z×Z≤⍴B

      ∇PROGRAMAS[⎕]∇
[0]   PROGRAMAS;A;B;C;D;IMP;MENSAJE1;MENSAJE2;MEN1;MEN2
[1]    A←NOM ⋄ C←'' ⋄ CLS ⋄ MENSAJE1←(15⍴' '),'LISTADO DE FUNCIONES EXISTENTES EN EL SISTEMA SEA' ⋄ MEN1←(15
[2]    →3+2×0=⍴A
[3]    A←⍕A ⋄ B←⍴↑A
[4]    →Z ⋄ C←C,26↑⍕B
[5]    ⍎(3×¯1+2×'S'=IMP←DINKEY ⋄ ⍞←'POR IMPRESORA (S/N)?')↑'PON    ' ⋄ MENSAJE2←(10⍴' '),'NOMBRE INTERNO
```

```
[6]   ⎕←((D,14⍴' '),((E,5)⍴NOM)[; 1 2 3 4]),((D,13)⍴'  '),(D,26)⍴C),⍕((1⍴NUM),1)⍴NUM ⋄ C←⎕←'  ' ⋄ C←MINZ
[7]   ⎕←(14⍴' '),('OBJE'),(13⍴' '),(27↑'OBJETIVO'),⍕'NUMDEJE'
[8]   ⍎(3×¯1+2/'S'=IMP)↑'POF   '

      ∇REEMPLAZA[⎕]∇
[0]   Z←L REEMPLAZA A;A;L;Z;E;C;D;E;J;F
[1]    B←L ⋄ E←0
[2]    →3+3×0=⍴B ⋄ E←E+1
[3]    B←(B⍳⎕AV[220])↓B ⋄ B←J↓B ⋄ C←B[⍳¯1+J←B⍳⎕AV[220]]
[4]    →2+3×0≠D←C POSICION A
[5]    →4 ⋄ A←A[⍳D-1],⎕AV[220],'←',(⍕E),⎕AV[220],A[(D+¯2+J)↓⍳⍴A]
[6]    B←L ⋄ E←0
[7]    →8+3×0=⍴B ⋄ E←E+1
[8]    E←J↓B ⋄ C←B[⍳¯1+J←B⍳⎕AV[220]]⋄ B←(B⍳⎕AV[220])↓B ⋄ F←⎕AV[220],'←',(⍕E),⎕AV[220]
[9]    →7+3×C≠D←F POSICION A
[10]   →9 ⋄ A←A[⍳D-1],C,A[(D+¯1+⍴F)↓⍳⍴A]
[11]   Z←A

      ∇REGLA1[⎕]∇
[0]   Z←REGLA1 I;A;C;CONJA;I;Z
[1]    A←NOM,CONJA←Z←''
[2]    →2+0=⍴A ⋄ A←5↓A ⋄ CONJA←CONJA,C,(0≠⍴C←I REGLA11 4↑A)↑⎕AV[220]
[3]    →10-6×0≠⍴CONJA
[4]    Z←Y REGLA12 CONJA

      ∇REGLA11[⎕]∇
[0]   Z←I REGLA11 A;A;I;Z
[1]    →2+1×0=∧/((⍕I,'1')IGUAL⍕(⍕A),'1'),(⍕I,'3')IGUAL⍕(⍕A),'3'
[2]    →4 ⋄ Z←A
[3]    Z←''

      ∇REGLA12[⎕]∇
[0]   Z←Y REGLA12 A;A;I;Z
[1]    Z←Y REGLA121 4↑A
[2]    →1+20×∨/(0≠⍴Z),0=⍴A←5↓A

      ∇REGLA121[⎕]∇
[0]   Z←Y REGLA121 A;I;Z;A;B;C;D;G;H;I;J;K;L;M;N;O;P;Q;R;T;CONJA;ECU;ECUA;FLAG;LL
[1]    CLS ⋄ ECU← 0 75 ⍴Z←'' ⋄ FLAG←0
[2]    D←B←¯4+NUM[⍕¯3↑A] ⍝ D+B+No. CASILLAS DE LA FN-4
[3]    →(15<C←1+⍎'NUM',I)/ALFA ⍝ C+1+No. CASILLAS DE LA F.O.
[4]    →FIN ⋄ Z←⍎I,'6←',(⍕A),'6' ⋄⍎I,'8←',(⍕A),'8' ⋄⍎I,'7←',(⍕A),'7'
[5]   ALFA:→6+'R' ESTA⍎I,⍕C←C-6
[6]    →5+11×C<16
[7]    →9 ⋄ E←D↑6
[8]    →3+∧/('F' ESTA⍎(⍕A),⍕B-1),(⍎I,⍕C+1)ESTA⍎(⍕A),⍕5
[9]    →150-142×10<E←B-6
[10]   →9+2×(⍎I,⍕C+2)FORMA⍎(⍕A),⍕B+1
[11]   →9+3×(⍎I,⍕C+4)TABLACV⍎(⍕A),⍕B+3
[12]   ECU←ECU,[1] 1 75 ⍴75↑(⍎I,⍕C+4),' + + SOI(',(⍕(⍕A),⍕B+3),')'
[13]   ECU←ECU,[1] 1 75 ⍴75↑⍎I,⍕C+2
[14]   ECU←ECU,[1] 1 75 ⍴75↑⍎Y,⍕C+1
[15]   →6 ⋄ ECU←ECU,[1] 1 75 ⍴75↑⍎Y,⍕C+5
[16]   ⎕←'' ⋄ ⎕←'VARIABLES A DESPEJAR:' ⋄ ⎕←'' ⋄ ⎕←'SOI(',(⍕(⍕A),'6'),')' ⋄ ⎕←'RESTRICCIONES  ' ⋄ ⎕←' ' ⋄ G←EC
```

```
[17]  G←0,4↑E↓↑6↓I-5 ⋄ I←I←''
[18]  →19↓6×G=⍴G←1↓G
[19]  H←,⍕(I←↑,⍕1↑G
[20]  →21←ι.J=⍴H
[21]  →23-1×'A'≠1↑K←H[ι⁻1↓J←H⍳⎕AV[220]]
[22]  →20 ⋄ H←J↓H
[23]  →24-6×K IGUAL I
[24]  →18 ⋄ L←L,'SO1(',K,'/',⎕AV[220],(⍕1↑G),⎕AV[220]⋄ I←K
[25]  →26-22×C=⍴L
[26]  →('S'≠1↑⍞⋄ C←'TIENE SOLUCION (S/N)?' ⋄ D←'' ⋄ 'SO1' SUJETOA A ⋄ C←'' ⋄ D←DIBUJA L/SUB[3]
[27]  L←SOLUCION L
[28]  Z←⍎,'6←L REEMPLAZA ⍕(⍴A), '6'''
''∩   ⍝ ' ' '
[31] SUBOBJ:J←52 ⋄ M←L ⍝ FORMA EL SUBOBJETIVO ⋄ FLAG←1
[32]  →33↓J=0
[33]  →32 ⋄ ⎕EX 'SUBO',⍕J←J-1
[34]  →(0=⍴M ⋄ B←10),FIN
[35]  M←J↓M ⋄ I←⍎M[ι⁻1↓J←M⍳⎕AV[220]]⋄ M←J↓M ⋄ N←M[4↓ι⁻2↓J←M⍳⎕AV[220]]
[36]  SUBO3←⍕(⍴A),⍕I+5 ⋄ SUBO1←⍕(⍴A),⍕I+2 ⋄ N←I
[37]  →38↓7×N/D ⋄ N←N+6
[38]  O←,⍕(⍴A),⍕N
[39]  →40↑5×0=⍴O
[40]  →41↓M IGUAL O[ι⁻1↓J←O⍳⎕AV[220]]
[41]  →39 ⋄ O←J↓O
[42]  ⍎'SUBO',(⍕B+2),'←',(⍕A),⍕N+2 ⋄⍎'SUBO',(⍕B+1),'←',(⍕A),⍕N+1 ⋄⍎'SUBO',(⍕B),'← '5' ,⎕AV[220]
[43]  ⍎'SUBO',(⍕B+4),'←',(⍕A),⍕N+4 ⋄⍎'SUBO',(⍕B+5),'←',(⍕A),⍕N+5
[44]  →37 ⋄ B←B+6
[45]  P←NOM,CONJA←'' ⍝ EMPIEZA A RESOLVER EL SUBOBJETIVO
[46]  →46↑C=⍴P ⋄ P←5↓P ⋄ CONJA←CONJA,((T←4↑P)),(T←/(C=⍴A ORIGEN ⍎P),C←⍴C←'SUBO' REGLA1 ⍎P↓⎕AV[220]
[47]  →34↑14×C=⍴CONJA
[48]  →47↑2/~F IGUAL A ⋄ CONJA←5↓CONJA ⋄ T←A ORIGEN F←4↑CONJA ⋄ ECUA← 0 75 ⍴''
[49]  C←B ⋄ ECUA←ECUA,[1] 1 75 ⍴75↑'SO1(',M,'/ ← ← SO2(',⍕(⍴P),⍕6),')'
[50]  →(10?B-1 ⋄ ECUA←ECUA,[1] 1 75 ⍴75↑⍕(⍴P),⍕1 ⋄ ECUA←ECUA,[1] 1 75 ⍴75↑⍕(⍴P),⍕3),SOL
[51] BETA1←52↑'B' ESTA⍎'SUBO',⍕Q←Q-5
[52]  →51↑11↑Q<16
[53]  →55 ⋄ B←2↑NUM(⍎⁻3↑P)
[54]  →55↑~/('B' ESTA⍕(⍴P),⍕B-1),(⍎'SUBO',⍕Q+1)ESTA⍕(⍴P),⍕B
[55]  →47↑7×10<B↑J-6
[56]  →55↑2×(⍎'SUBO',⍕Q+2)FORMA⍕(⍴P),⍕B+1
[57]  →55↑3×(⍎'SUBO',⍕Q+4)TABLAC5⍕(⍴P),⍕B+3
[58]  ECUA←ECUA,[1] 1 75 ⍴75↑'SO1(',(⍎'SUBO',⍕Q+4),'/ ← ← SO2(',(⍕(⍴P),⍕B+3),')'
[59]  ECUA←ECUA,[1] 1 75 ⍴75↑⍎'SUBO',⍕Q+2
[60]  ECUA←ECUA,[1] 1 75 ⍴75↑⍎'SUBO',⍕Q+1
[61]  →52 ⋄ ECUA←ECUA,[1] 1 75 ⍴75↑⍎'SUBO',⍕Q+5
[62] SOL:D←'' ⋄ D←ECUA ⋄ D←'' ⋄ D←ECU ⋄ CLS
[63]  D←'' ⋄ D←'VARIABLES A DESPEJAR:'
[64]  →47↑16×'S ≠1↑⍞⋄ D←'TIENE SOLUCION S/N?' ⋄ D←'' ⋄ 'SO2' SUJETOA F ⋄ 'SO1' SUJETOA A ⋄ D←'' ⋄ D←'SO2/'.
SUBVAR P ⋄ D←DIBUJA L
[65]  LL←SOLUCION LL ⋄ L←SOLUCION L
[66]  Z←⍎,'6←L REEMPLAZA ⍕(⍴A),''6'''
[67]  ⍎1,'5←(L REEMPLAZA ⍕(⍴A),''6''),⎕AV[220],LL REEMPLAZA ⍕(⍴F),''6'''
[68]  ⍎1,'7←T'
[69] FIN:

      ⎕TOFF
```

```
      ∇SOLUCION[⎕]∇
[0]   Z←SOLUCION L;L;Z;J;K
[1]    Z←'' ⋄ ⍞←'TECLEE UNA A UNA LAS VARIABLES PEDIDAS '
[2]    →3+2×0=⍴L
[3]    D←K←L[⍳¯1+J←L⍳⎕AV[220]]
[4]    Z←Z,(¯1↓4↑K),⎕AV[220],'(',D,')',⎕AV[220]
[5]    L←J↓L
[6]    J←L⍳⎕AV[220]
[7]    →2 ⋄ L←J↓L

      ∇SUBVAS[⎕]∇
[0]   Z←SUBVAP P;A;B;C;I;IMPRIME;J
[1]    →((1↑A←1+NUM[⍴¯3↑P]⋄ IMPRIME←I←Z←'' ⋄ B←4)/FIN
[2]   LOOP:→(A=B←B+6)/FIN
[3]    C←,⍕(⍴P),⍕B
[4]   LAST:→(0=J←'A' POSICION C)/LOOP
[5]    →((C[J+¯1+⍳3]IGUAL I)/LOOP
[6]    →LOOP ⋄ IMPRIME←IMPRIME,'SO2(',I,')',⎕AV[220]⋄ Z←Z,'SO2(',I,')',⎕AV[220],'S',⎕AV[220]⋄ I←C[J+¯1+⍳3]
[7]   FIN:⍞←IMPRIME

      ∇SUJETOA[⎕]∇
[0]   G←B SUJETOA A;A;B;Z;CASILLAS;MAT;C;TEMP;CLAVE;IMP;D;ESP
[1]    →FIN ⋄ Z←A ⋄ G←''
[2]    →3+3×Z IGUAL 'OBJE' ⋄ ESP←' '
[3]   ⍝ →1+3×∧/('A' IGUAL 1↑Z),(¯3↑Z)∊'1234567890'
[4]    Z←⍎Z ⋄ CASILLAS←NUM[⍎¯3↑Z]
[5]    →7
[6]    CASILLAS←⍎'NUM',Z
[7]    MAT[80+⍳6]←8↑B ⋄ MAT←,MATRIZ1 ⋄ CLAVE← 327 1117 1512 350 745 1140 1535 768 1552
[8]    ⍞(3×¯1+2×'S'=IMP)↑'FOR    ' ⋄ C←10 ⋄ IMP←'N'
[9]    →10+2×0=C←C-1
[10]   TEMP←88↑((22×67)⍴' ',⍕Z,⍕C/⍕' '),⍕Z,⍕C
[11]   →11-2×0=⍴TEMP←22↓TEMP ⋄ MAT[(CLAVE[C]+79×4-⍳⍴TEMP)+22)+¯1+⍳22]←22↑TEMP
[12]   D← 24 75 ⍴MAT
[13]   CLAVE← 90 485 880 1275 1670 2065 113 508 903 1298 1693 2088 136 531 926 1321 1716 2111
[14]   C←0
[15] ⍝ ⍞(7×¯1+2×'S'=IMP)↑'D←D←ESP DINKE;
[16]   →17+5×CASILLAS≤C+9
[17]   MAT[8C+⍳5]←8↑B ⋄ MAT←,MATRIZ2 ⋄ D←1+CASILLAS⌊C+27
[18]   →19+2×(C+9)=I←D-1
[19]   TEMP←88↑((22×67)⍴' ',⍕Z,⍕D/⍕' '),⍕Z,⍕D
[20]   →20-2×0=⍴TEMP←22↓TEMP ⋄ MAT[(CLAVE[D-9+C]+79×4-⍳⍴TEMP)+22)+¯1+⍳22]←22↑TEMP
[21]   →16 ⋄ C←C+10 ⋄⍞(7×¯1+2×-/('S'=IMP),(C+36)≤⍳C+36)↑'    D←⍕D←D←ESP' ⋄ D← 31 75 ⍴MAT
[22]   ⍞(3×¯1+2×'S'=IMP)↑'FOF    '
[23] FIN:

      ∇TABLACV[⎕]∇
[0]   Z←A TABLACV B;W;B;Z
[1]    →2+3×Z←VARIABLE B ⋄ A←⍎(¯1+2×∧/(VARIABLE B),0=⍴A)↑ B←
[2]    →4-0=VARIABLE A
[3]    →10 ⋄ Z←A IGUAL B
[4]    Z←∧/('0'=1↑A),~VARIABLE 1↑A

      )TOFF
```

```
      ∇VARIABLE[⎕]∇
[0]   Z←VARIABLE A;A;Z
[1]    Z←0=∧/('12',A)∈'-1234567890.'

      ∇a[⎕]∇
[0]   Z←A a B
[1]    Z←A

      FOF
```

Bibliografia

1 .- M. Andreka, T. Gergely, I. Nemeti:
"Definition Theory as Basis for a Creative Problem Solver",
Advanced Papers of the 4th IJCAI,Georgia,URSS 3-8 sep. 1975.

2 .- R. Balzer, N. Goldman, D. Wile:
"Meta-Evaluation as a Tool for Program Understanding",
Advanced Papers of the 5th. IJCAI 1977.

3 .- D. Barstow:
"A Knowledge-Based System for Automatic Program Construction",
Advanced Papers of the 5th IJCAI 1977.

4 .- R. Brooks:
"A Model of Human Cognitive Behavior in Writing Code for Computer Programs",
Advanced Papers of the 5th IJCAI 1977.

5 .- F. Calo, G. Guida, M. Somalvico:
"Problem solving as a basis for Program Synthesis: Design and Experimentation of the SIS System",
Int.J. Man-Machine Studies (1982)17,173-188.

6 .- A.N. Campbell, V.F. Hollister, R.O. Duda, P.E. Hart:
"Recognition of a Hidden Mineral Deposit by an Artificial Intelligence Program",
SCIENCE Vol. 217, 3 sep. 1982. 927-929.

7 .- J. G. Carbonell:
"Learning by Analogy: Formulating and Generalizing Plans From Past Experience",
Machine Learning. An Artificial Intelligence Approach.
R.S. Michalski, J.G. Carbonell, T.M. Mitchell (editors)
Tioga Publishing Company,Palo Alto,California 1983.
ISBN 0-935382-03-4

8 .- K. Clark, S. Sickel:
"A Calculus for Deriving Programs".
Advanced Papers of the 5th IJCAI 1977.

9 .- R. Davis:
"Interactive Transfer of Expertise: Acquisition of New Inference Rules".
Advanced Papers of the 5th. IJCAI 1977.

10.- R. Davis, B. G. Buchanan:
"Meta-Level Knowledge: Overview and Applications",
Advanced Papers of the 5th. IJCAI 1977.

11.- N. Dershowitz:
"Automatic Program Annotation".
Advanced Papers of the 5th. IJCAI 1977.

12.- G. W. Ernst, M. M. Goldstein:
"Mechanical Discovery of Classes of Problem-Solving Strategies",
Journal of the Association for Computing Machinery Vol. 29 No. 1 January 1982. pp 1-23.

13.- E. A. Feigenbaum:
"Innovation and Symbol Manipulation in 5th Generation Computer Systems",
"La Computadora de la Quinta Generacion",
UNAM agosto 1984.

14.- T. Furugori:
"Knowledge and its use in a Program for Going from one Place to Another",
Advanced Papers of the 4th IJCAI,Georgia URSS 3-8 sep. 1975.

15.- K. Furukawa, R. Nakajima, A. Yonezawa, S. Goto.:
"Problem Solving an Inference Mechanisms",
"La Computadora de la Quinta Generacion",
UNAM agosto 1984.

16.- P. J. Gloess:
"Understanding Lisp",
Alfred Publishing Co.,Inc. 1982.,
ISBN 0-88284-218-8

17.- G. A. Goldin, G. F. Luger:
"Problem Structure an Problem Solving Behavior",
Advanced Papers of the 4th IJCAI,Georgia URSS 3-8 sep. 1975.

18.- C. Green:
"Application of Theorem Proving to Problem Solving",
Proceedings of IJCAI, Washington D.C. pp 219-239 (1969)

19.- C. Green, D. Barstow:
"Some Rules for the Automatic Synthesis of Programs":
Advanced Papers of the 4th IJCAI,Georgia URSS 3-8 sep. 1975.

20.- C. Green, D. Barstow:
"A Hypothetical Dialogue Exhibiting a Knowledge Base for a Program-Understanding System",
Machine Intelligence 8. E. W. Elcock & D. Michie (editors) 1977.

21.- C. Green:
"A Summary of The Psi Program Synthesis System",
Advanced Papers of the 5th. IJCAI 1977.

22.- G. Guida, M. Guida, S. Gusmeroli, M. Somalvico:
Milan Polytechnic Artificial Intelligence Project,
Politecnico di Milano. Milano, Italy,
"ESAP: An Expert System for Automatic Programing",
ICAI-1984. Advances in A.I..
T'Oshea (editor),
Elsevier Science Publisher B.V. (North Holland),
ECCAI, 1984.

23.- C. Hewitt:
"How to Use What You Know",
Advanced Papers of the 4th IJCAI,Georgia URSS 3-8 sep. 1975.

24.- J.-F. Jouannaud, B. Guino, J.-F. Treuil:
"An Interactive System Able to Synthesize Functions from Examples",
Advanced Papers of the 5th. IJCAI 1977.

25.- H. Karatsu:
Matsushita Communication Industrial Co.,Ltd.,
Yokonama, Japan,
"What is required of the 5th Generation Computer - Social Needs en its Impact",
"La Computadora de la Quinta Generacion",
UNAM agosto 1984.

26.- S. Katz, Z. Manna:
"Logical Analysis of Programs",
Studies in automatic programming Logic. A.I. Series.,
Nilson J.N. editor. Elsevier North Holland Inc. Publishing,
Co. 1977.

27.- Yu. I. Klykov, V.N. Pushkin:
"Semantic Language and the Problem of Goal Formation Modelling in Human Thinking ",
Advanced Papers of the 4th IJCAI,Georgia URSS 3-8 sep. 1975.

28.- J. Knapman:
"Some Principles of Artificial Learning That Have Emerged from Examples",
Advanced Papers of the 4th IJCAI,Georgia URSS 3-8 sep. 1975.

29.- P. M. Kolers, W. E. Smythe:
"Symbol Manipulation: Alternatives to the Computational View of Mind",
Learning and Verbal Behavior 23, pp 289-314 (1984).

30.- R.C.T. Lee, C.L. Chang, R.J. Waldinger:
"An Improved Program-synthesizing Algorithm and its Correctness",
Communications of the ACM,April 1974, Vol. 17, No. 4 211-217

31.- Z. Manna, R. Waldinger:
"Knowledge an Reasoning in Program Synthesis",
Advanced Papers of the 4th IJCAI,Georgia URSS 3-8 sep. 1975.

32.- Z. Manna, R. Waldinger:
"The Automatic Synthesis of Systems of Recursive Programs",
Advanced Papers of the 5th. IJCAI 1977.

33.- Z. Manna, R. Waldinger:
"Synthesis: Dreams => Programs",
IEEE Transactions on Software Engineering, vol. se-5,
no. 4, julio 1979.

34.- W. S. Mark:
"The Reformulation Approach to Building Expert Systems",
Advanced Papers of the 5th. IJCAI 1977.

35.- J. McCarthy, P.J. Hayes:
"Some Philosophical Problems from the Standpoint of
Artificial Intelligence",
Machine Intelligence 4. D.Michie, B. Meltzer (editors),
1969.

36.- T. Moto-oka et al.:
"Keynote speech: Challenge for knowledge Information
Processing Systems",
Preliminary report on fifth generation computer systems.
"La Computadora de la Quinta Generacion",
UNAM agosto 1984.

37.- N. J. Nilson:
"Theorem Proving in the Predicate Calculus":
Problem Solving Methods in A.I., Nilson N. J.,
Mc. Graw Hill 1971.

38.- M. Rychener, C. Forgy, P Langley, J. McDermott, A. Newell,
R. Ramakrishna:
"Problems in Building an Instructable Production System",
Advanced Papers of the 5th. IJCAI 1977.

39.- H. Samet:
"Toward Automatic Debugging of Compilers",
Advanced Papers of the 5th. IJCAI 1977.

40.- C. B. Schwind:
"A State Logic for the Representation of Natural Language
Based Intelligent Systems",
Advanced Papers of the 5th. IJCAI 1977.

41.- D. E. Shaw, W. R. Swartout, C. Green:
"Inferring Lisp Programs from Examples",
Advanced Papers of the 4th IJCAI,Georgia URSS 3-8 sep. 1975.

42.- L. Stillossy, D.A. Sykes:
"Automatic Program Synthesis from Example Problems",
Advanced Papers of the 4th IJCAI,Georgia URSS 3-8 sep. 1975.

43.- R. G. Smith, T. M. Mitchell, R. Chestek, B. G. Buchanan:
"A Model for Learning Systems".
Advanced Papers of the 5th IJCAI 1977.

44.- N.S. Sridharan, F. Hawrusik:
"Representation of Actions That have Side-Effects",
Advanced Papers of the 5th. IJCAI 1977.

45.- N. Suwa, K. Furukawa, A. Nakinouchi, T. Mizoguchi, F. Mizoguchi, H. Yamasaki.:
"Knowledge Base Mechanisms",
"La Computadora de la Quinta Generacion",
UNAM agosto 1984.

46.- O.K. Tikhomirov:
"Philosophical and Psychological Problems of Artificial Intelligence",
Advanced Papers of the 4th IJCAI,Georgia URSS 3-8 sep. 1975.