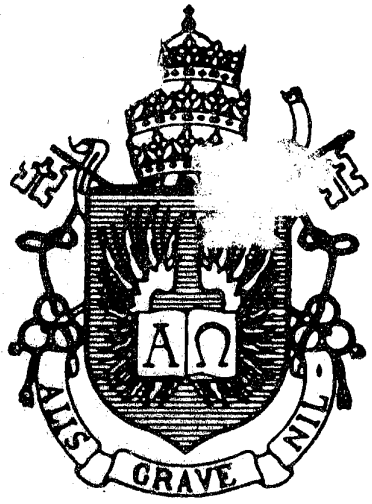

PUC

Series: Monografias em Ciência da Computação
Nº 3/87

HACIA UN METAMODELO DEL PROCESO DE DESARROLLO DE SOFTWARE BASADO EN TEORIA DE PROBLEMAS

Armando M. Haeberer
Paulo A. S. Veloso
Gabriel Baum

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO
A MARQUÊS DE SÃO VICENTE, 225 – CEP 22453
RIO DE JANEIRO – BRASIL

Series: Monografias em Ciência da Computação
Nº 3/87

HACIA UN METAMODELO DEL PROCESO DE DESARROLLO DE SOFTWARE BASADO EN TEORIA DE PROBLEMAS

Armando M. Haeberer
Paulo A. S. Veloso
Gabriel Baum

Departamento de Informática

PUC/RJ - Departamento de Informática

Series: Monografias em Ciência da Computação
Nº 3/87

October 1987

Series Editor: Paulo A. S. Veloso

HACIA UN METAMODELO DEL PROCESO DE DESARROLLO DE SOFTWARE BASADO EN TEORIA DE PROBLEMAS

Armando M. Haeberer[1]
Paulo A. S. Veloso[2]
Gabriel Baum[1]

1 ESLAI - Escuela Superior Latinoamericana de Informática
Argentina

2 PUC/RJ - Pontifícia Universidade Católica do Rio de Janeiro
Brasil

* Research partly sponsored by the ETHOS project of the Argentinian Brazilian Program for Research and Advanced Studies in Computer Sciences

TOWARDS A METAMODEL OF THE SOFTWARE DEVELOPMENT PROCESS BASED ON A THEORY OF PROBLEMS

ABSTRACT

The need for a formal model of the software development process has been felt for some time now. Such a model is needeed in order to understand the deep nature of this process and for the design and implementation of an environment to support software development. This work is based on the viewpoint that the software development process may be regarded as consisting of a series of linguistic transformations starting from the requirements of a problem and ending with a solution for it in the form of an efficient program. First, some fundamental notions on the concept of model as well as Lehman's PM model are reviewed. Then, some methodologies for program construction are examined, which suggests a first view of the software process as solving problems. Some concepts from an algebraic theory of problems are briefly introduced and used to provide a framework for the analysis of the software process. Finally, a more precise description of our metamodel is presented, emphasizing the distinctions between executable and non-executable, formal and informal specifications, as well as between verification and validation by testing.

Key words : Software development process, metamodel, algebraic problem theory, programming methodologies, linguistic transformations, problem solving, verification, testing.

HACIA UN METAMODELO DEL PROCESO DE DESARROLLO DE SOTWARE BASADO EN TEORIA DE PROBLEMAS

RESUMEN

Desde hace algùn tiempo se ha sentido la necesidad de un modelo formal del proceso de desarollo de /software./ Tal modelo es necesario para la compreensiòn de la naturaleza profunda de dicho proceso asi como para el diseño e implementaciòn de un ambiente para soportar el desarrollo de software. Este trabajo se basa en el punto de vista de que el proceso de desarrollo de software puede considerarse como compuesto por una serie de transformaciones lingüisticas que comienza con los requerimientos de un problema y termina con una soluciòn para el mismo en la forma de un programa eficiente. Se revisan inicialmente algunas nociones fundamentales sobre el concepto de modelo asi como tambièn sobre el modelo FW de Lehman. Luego, se examinan algunas metodologias para la construcciòn de programas, lo que sugiere una primera visiòn del proceso de software como uno de resoluciòn de problemas. Se introducen suscintamente algunos conceptos de una teoria algebraica de problemas, los cuales se utilizan para proveer un marco para el anàlisis del proceso de software. Se presenta finalmente una descripciòn màs precisa de nuestro metamodelo, enfatizando la distinciòn entre especificaciones ejecutables y no ejecutables, formales e informales asi como entre verificaciòn y validaciòn.

Palabras clave : Proceso de desarrollo de /software/ metamodelo, teoria /algebraica/de problemas, /metodologias de programaciòn/ transformaciones lingüisticas, resoluciòn de problemas, verificaciòn, testeo.

PARA UM METAMODELO DO PROCESSO DE DESENVOLVIMENTO DE PROGRAMAS BASEADO EM TEORIA DE PROBLEMAS

RESUMO

A necessidade de um modelo formal do processo de desenvolvimento de programas se faz sentir já há algum tempo. Um tal modelo é necessário para se entender a natureza profunda deste processo bem como para o projeto e implementação de um ambiente para suportar o desenvolvimento de programas. Este trabalho se baseia no ponto de vista de que o processo de desenvolvimento de programas pode ser considerado como consistindo de uma série de transformações lingüísticas começando com os requisitos de um problema e terminando com uma solução para este na forma de um programa eficiente. Inicialmente, são passadas em revista algumas noções fundamentais sobre o conceito de modelo bem como o modelo FW de Lehman. Então, algumas metodologias para construção de programas são examinadas, sugerindo uma primeira maneira de ver o processo de programação como um de resolução de problemas. Alguns conceitos de uma teoria algébrica de problemas são informalmente introduzidos e empregados para fornecer um arcabouço para a análise do processo de programação. Finalmente, uma descrição mais precisa de nosso metamodelo é apresentada, enfatizando a distinção entre especificações executáveis e não executáveis, formais e informais bem como entre verificação e validação por testes.

Palavras chave : Processo de desenvolvimento de programas, metamodelo, teoria algébrica de problemas, metodologias de programação, transformações lingüísticas, resolução de problemas, verificação, teste.

CONTENIDO

1 - Introducción

En los últimos diez años se ha comenzado a desarrollar la idea de la necesidad de modelizar el proceso de desarrollo de software tanto para comprender la naturaleza profunda del mismo así como para obtener la estructura conceptual necesaria para el diseño e implementación de una herramienta integrada que cubra dicho proceso de desarrollo, que -los autores concuerdan con Lehman [6]- solamente puede apoyarse en una base teórica, una estructura conceptual unificadora y un modelo coherente del proceso en cuestión.

Así la modelización del proceso de desarrollo de software ha pasado desde el modelo intuitivo del ciclo de vida de Boehm [34], al modelo PW de Lehmann [6], gracias a contribuciones como las de Wirth [33], Turski [7], Balzer [32], y Maibaum [8].

Este trabajo parte de la concepción de que el proceso de desarrollo de software puede ser descompuesto en una serie de transformaciones entre las distintas representaciones linguísticas del problema a ser resuelto así como de su solución hasta llegar al programa buscado [6]. Por lo tanto nos basaremos, para comenzar la exposición de nuestras ideas, sobre el modelo PW de Lehman.

2-Algunas nociones fundamentales sobre el concepto de "modelo"

En adelante se utilizará la palabra "modelo" en dos sentidos que será fácil diferenciar por el contexto. El primero de ellos es el sentido ingenieril de tal término, según esta acepción un modelo es una estructura o mecanismo utilizado para interpretar fenómenos o procesos naturales o artificiales. El segundo de ellos es el sentido lógico que dada la importancia que tiene este concepto de modelo en el desarrollo de este trabajo creemos necesario detenernos un instante en él ([35]y [36]).

Comenzaremos por analizar un ejemplo, la geometría euclídea es un sistema formal (lenguaje más axiomas y teoremas) basado en un lenguaje de primer orden donde las nociones de "punto", "recta", "se cortan", etc., no tienen ningún significado. Ahora bien, si interpretamos a la noción de recta como la ecuación que representa una recta en el espacio, la de punto como una terna, etc., estaremos dando un significado a las mismas y obtendremos por ejemplo la geometría analítica, que es un modelo de la geometría euclídea. Si interpretamos, en cambio, a la noción de recta como rayo a la de punto como foco, "pasa por un punto" como "pasa por el foco" etc. obtendremos la óptica gaussiana que es otro modelo de la geometría euclídea.

Observemos que la expresión "las rectas X e Y se cortan " no es verdadera ni falsa en geometría euclídea en cambio es fácil ver que una interpretación de la misma como "la arista formada por el piso y cualquier pared se corta con la formada por el techo y cualquier pared" es obviamente falsa, es decir, no es modelo de la primera. Una interpretación da contenido semántico a las fórmulas del lenguaje.

Un poco más formalmente un lenguaje de primer orden con igualdad es un conjunto infinito de símbolos distribuídos de la siguiente manera:

A) Símbolos Lógicos:

i) Paréntesis: (,).

ii) Conectivos proposicionales: $\rightarrow, \neg$.

iii) Variables V_i (Una para cada entero positivo).

iv) Un símbolo predicativo binario denominado de igualdad: $\approx$.

B) Símbolos extra lógicos (parámetros):

i) Símbolo de cuantificación: $\forall$.

ii) Símbolos predicativos: para cada entero positivo un conjunto (posiblemente vacío) de símbolos, denominados símbolos predicativos n-arios.

iii) Constantes: algún conjunto (posiblemente vacío) de símbolos.

iv) Símbolos funcionales: para cada entero positivo un conjunto (posiblemente vacío) de símbolos denominado símbolos funcionales n-arios.

Se denominan términos a las variables, las constantes y al resultado de aplicar reiteradamente símbolos funcionales del lenguaje a variables o constantes.

Por ejemplo, en el lenguaje de la teoría de números existen variables X_i , un par de constantes denominadas 0 y 1 , un predicado binario $<$, un símbolo funcional unario Suc

(sucesor de) y dos símbolos funcionales binarios + y *. Entonces (x_i+x_j) (que debería escribirse $+(x_i,x_j)$), (x_i*x_j), (x_i+0), (x_i*1), (x_i+1), $((x_i+x_j)*(x_k+x_h))$, $Suc(x_i)$, etc., son términos.

Se denominan fórmulas atómicas a las expresiones resultantes de aplicar los símbolos predicativos n-arios a términos del lenguaje. Por ejemplo en la teoría de números $(x_i \approx x_j)$ (que debería escribirse $\approx(x_i,x_j)$), $(x_i \approx 1)$, $((x_i+x_j)\approx(x_k+x_h))$, $((x_i+x_j)\approx(x_k+1))$, etc, son fórmulas atómicas.

Se denominan fórmulas bien formadas a aquellas expresiones obtenidas de la siguiente manera: i) todas la fórmulas atómicas son fórmulas bien formadas, ii) si ψ y δ son fórmulas bien formadas, entonces $(\psi \rightarrow \delta)$ y $\neg(\psi)$ también lo son; iii) si ψ es una fórmula bien formada y x_i una variable entonces $(\forall x_i)(\psi)$ también es una fórmula bien formada.

Volviendo a nuestro ejemplo de la teoría de números, $(x_i \approx Suc(x_j))$, $((x_i+x_j)<(x_k*x_h))$, $(\forall x_i)(1<x_i)$, etc., son fórmulas bien formadas.

Una interpretación $\mathcal{H}$ para un lenguaje de primer orden es una función cuyo dominio es el conjunto de símbolos extra lógicos tal que:

i) $\mathcal{H}$ asigna al símbolo de cuantificación $\forall$ un conjunto no vacío $\Delta_{\mathcal{H}}$ denominado el "universo" o dominio de interpretación de $\mathcal{H}$.

ii) $\mathcal{H}$ asigna a cada símbolo predicativo n-ario una relación n-aria en $\Delta_{\mathcal{H}}$.

iii) $\mathcal{H}$ asigna a cada símbolo constante un miembro del universo $\Delta_{\mathcal{H}}$.

iv) $\mathcal{H}$ asigna a cada símbolo funcional n-ario una operación n-aria en $\Delta_{\mathcal{H}}$.

Es decir, $\mathcal{H}$ asigna significado a los símbolos extra lógicos del lenguaje; así, $\forall$ significará $\Delta_{\mathcal{H}}$, el símbolo constante C será un elemento de $\Delta_{\mathcal{H}}$, etc. Una valuación V es una función que aplica a cada variable del lenguaje un elemento de $\Delta_{\mathcal{H}}$.

Diremos entonces que dada una fórmula bien formada Φ de nuestro lenguaje, $\mathcal{H}$ satisface a Φ con la valuación V (que se denota $\models_{\mathcal{H}}\Phi[V]$) si y solo sí la traducción de Φ determinada por $\mathcal{H}$ -donde las variables no cuantificadas x_i se reemplacen por $V(x_i)$- es verdadera.

Por último, una interpretación es un modelo (en el sentido lógico) de una fórmula bien formada Φ sí y solo sí satisface a Φ para cualquier valuación (lo que se escribe $\models_{\mathcal{H}} \Phi$).

3-Descripción suscinta del modelo PW de Lehman

Volviendo ahora al proceso de desarrollo de software, el modelo PW de Lehman parte de la base de que tal proceso puede verse como una transformación de la primera verbalización de un problema a ser resuelto (denominada concepto de la aplicación) en un programa de computadora y de que dicha transformación es tan compleja como para que deba ser descompuesta en subtransformaciones para poder ser llevada a cabo exitosamente.

Así, una primera descomposición de tal transformación es la que se logra dividiéndola en dos partes: una de abstracción de dicha primera verbalización en una especificación formal y otra (denominada de reificación) de concretización de dicha especificación formal en un programa de computadora.

Este proceso puede verse como la V invertida representada en la figura 1:

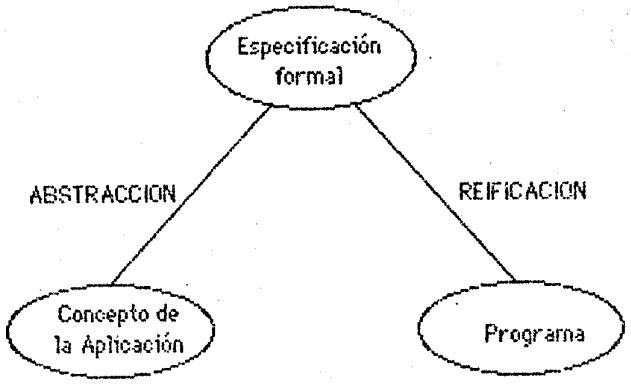

figura 1

Así la especificación formal es una abstracción tanto del concepto informal de la aplicación, en el sentido en que se han eliminado los detalles irrelevantes de la misma, como del programa, ya que para poder llegar de la especificación a este último se deben agregar nuevos detalles, esta vez no referentes a la aplicación sino al lenguaje o arquitectura en la que estará escrito o sobre la que deba ser ejecutado dicho programa. Así la especificación formal puede ser considerada como una teoría de la cual tanto el concepto de la aplicación como el programa son modelos [7].

Sin embargo, tanto el proceso de abstracción como el de reificación son también transformaciones suficientemente complejas como para deber a su vez ser descompuestas en subtransformaciones. Por lo tanto una descripción más detallada aunque simplificada del modelo PW sería la de la figura 2.

La explicación de esta figura es la siguiente : el proceso de reificación es un proceso compuesto por transformaciones linguísticas cuya propiedad fundamental es que dos representaciones cualesquiera relacionadas por una transformación son logicamente equivalentes (cada una a su nivel de abstracción). Por lo tanto dicho proceso de transformación debe comenzar obligatoriamente en una representación formal inicial que, en algún sentido, sea correcta o satisfactoria.

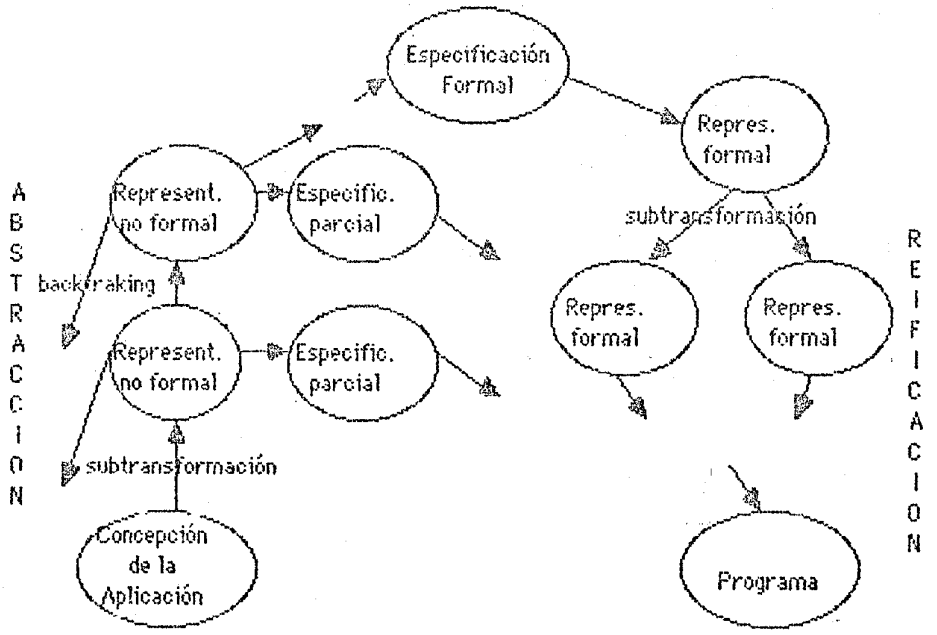

figura 2

Dicha representación inicial , no es otra cosa que la especificación formal.

¿Cómo conseguir que tal especificación formal sea correcta o satisfactoria?, comparando frecuentemente -validando- durante el proceso de su construcción con el objeto por ella descripto, con el objeto real y corrigiendo o ajustando los defectos encontrados. Este proceso de abstracción (pierna izquierda de la V) también se resuelve por descomposición en subtransformaciones que producen lo que podríamos denominar especificaciones parciales.

Asi, según Lehman, la especificación formal es un objeto ideal formado en realidad por una colección estructurada de subespecificaciones cuyos elementos proveen los puntos de partida para aquellas partes del proceso basadas en el desarrollo formal y representadas en el modelo por la pierna derecha de la V.

Por su parte, el proceso de reificación de la pierna derecha es también un proceso de transormaciones linguísticas entre modelos representacionales de la subespecificaciones y composiciones de los mismos hasta llegar a una representación de la colección estructurada de subespecificaciones que compone la especificación formal en el lenguaje en que será definitivamente escrito el programa.

La diferencia del proceso de reificación con el de abstracción estriba en que en el primero, los modelos representacionales antes y después de los pasos de transformación son descripciones formales del problema que esta siendo transformado en el programa final, mientras que en el segundo las primeras representaciones, en la cadena de transformaciones, son poco precisas, informalmente descriptas, ambiguas e incompletas.

El proceso de reificación en cuestión se basa fundamentalemente, siempre según Lehman, en la aplicación de tres principios enunciados por Turski ([6] y [7]), a saber:

1) Las porciones formales del proceso de desarrollo de software pueden verse como una secuencia de transformaciones linguísticas de modelos representacionales.
2) En cualquier paso de esta secuencia se puede aplicar creatividad al diseño tanto del lenguaje en que se representará el codominio de la transformación como al del proceso de transformación en sí mismo.
3) Que en el diseño de un proceso específico se debe restringir tal creatividad a uno de los dos diseños expuestos en lugar de seguir un proceso híbrido aplicándola a ambos.

Dicho proceso de reificación, según lo propuesto por Lehman, está formado por la aplicación reiterada de un paso canónico cuyo núcleo es la tranformación formal de una representación del problema expresada en un sistema linguístico base, en una versión refinada de la misma expresada en un sistema linguístico que podríamos denominar destino. Una vez aplicada esta transformación se debe verificar formalmente la misma para asegurar la equivalencia lógica de ambas representaciones.

Dados el lenguaje base y el destino de una transformación y una representación del problema en el primero de ellos, la selección de una representación del mismo en el segundo -de entre todas las producidas en los procedimientos de transformación alternativos- solo puede basarse en un juicio y no en un proceso de cálculo.

Una vez aceptada la representación en el lenguaje destino -posiblemente por verificación y validación- ésta se convierte en la base de un nuevo paso canónico de transformación. Puede entonces tomarse este paso canónico como el paradigma del proceso formal de transformación.

La figura 3 es una representación esquemática [6] del proceso de transformación formal a partir de la especificación para llegar al programa.

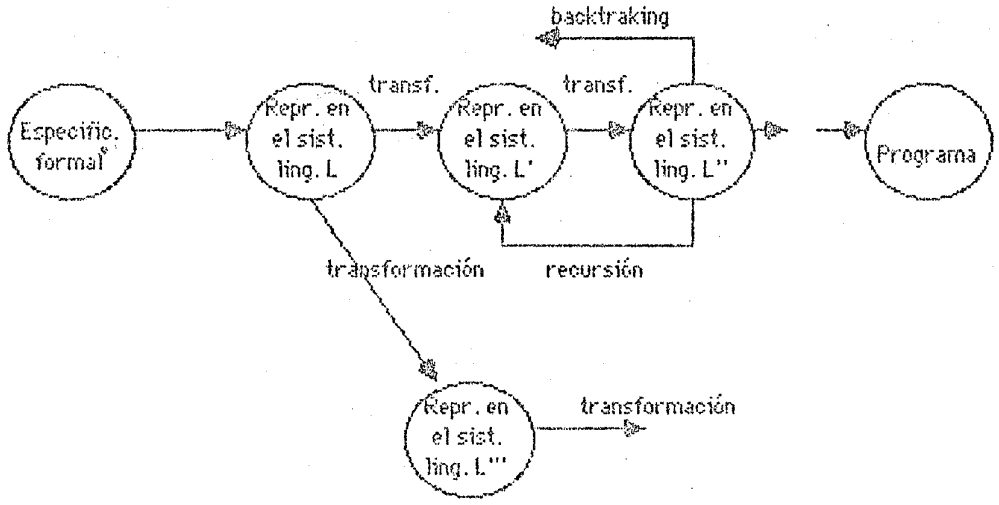

figura 3

Como última reflexión también debida a Lehman, sobre el modelo PW, deberíamos acotar que a pesar de que el proceso de abstracción representado por la pierna izquierda de la figura 2 supone una actividad mental no modelizable en los términos aqui descriptos, una vez verbalizado el problema, el proceso de dicha pierna puede también ser descompuesto en una serie de pasos que no deberían diferir demasiado del paso canónico arriba descripto.

fórmulas de un lenguaje de primer orden que tienen modelos entre las descripciones recursivas de funciones son aquellas que son a su vez modelo de alguna fórmula de la lógica constructiva.

Dicho en otras palabras, con la primera transformación de Bauer uno "construye" una solución expresada mediante descripciones recursivas de funciones del problema expresado mediante fórmulas de un lenguaje de primer orden, mientras que, con la segunda trasnformación simplemente se traduce (por supuesto refinando y optimizando) la expresión recursiva de dicha solución en un programa escrito en lenguaje "do-while".

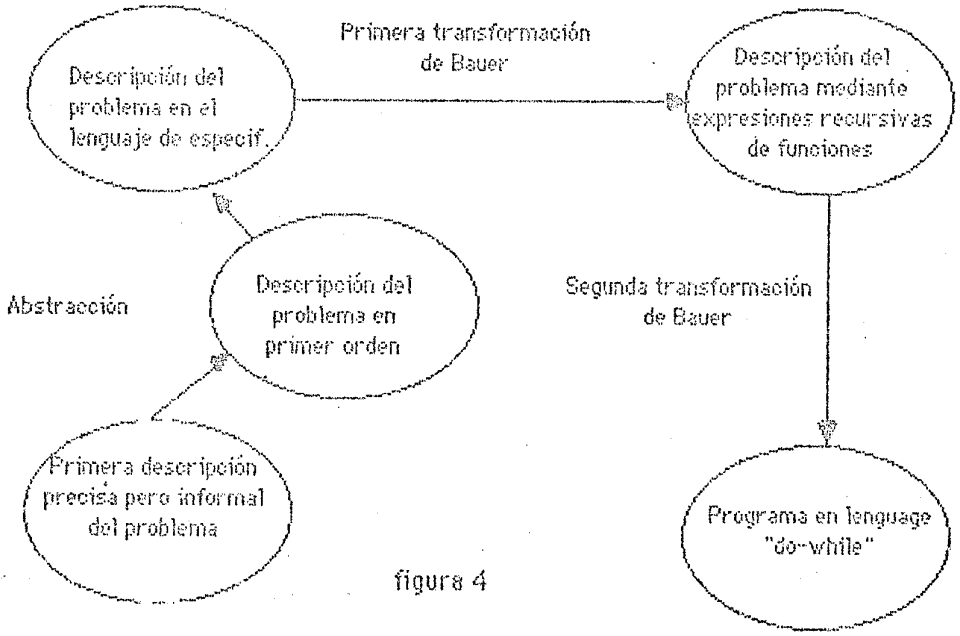

figura 4

Ambas tranformaciones son, entonces, de naturaleza distinta , no solamente debido a los sistemas liguísticos que relacionan sino respecto a su naturaleza misma, dado que la segunda es una traducción mientras que la primera además de traducir "construye" una solución.

Analicemos ahora el caso de la metodología de Jackson. El primer paso de esta metodología es "describir" las estructuras de datos de entrada y salida del programa mediante ciertos tipos de datos primitivos; se relacionan luego estas representaicones mediante relaciones funcionales; a continuación se construye mediante un procedimiento dado la estructura del programa mediante dichos tipos primitivos traduciéndose luego esta estructura en un programa escrito en un lenguaje "do-while". Por último se completa dicha estructura con el código necesario para describir las acciones semánticas de los subproblemas no modelizados mediante los tipos primitivos arriba mencionados. Una representación de esta metodología es la expuesta en la figura 5.

Obsérvese que en este caso no existe una especificación del problema, sino directamente una especificación de una solución del mismo. La especificación de las estructuras de entrada y salida del programa -descriptas mediante los tipos de datos primitivos de Jackson- y relacionadas mediante las relaciones funcionales correspondientes, ya constituyen una solución "ejecutable" del problema si se cuenta con el código correspondiente a las "hojas" (subproblemas) del árbol del programa que es la representación de éste último mediante los tipos primitivos de Jackson.

4-Una primera descripción de un metamodelo del proceso de desarrollo de software

Como ya se dijo, el proceso de desarrollo de software ha sido visto como un proceso de transformación de una descripción informal del concepto de la aplicación en una descripción sumamente formal y detallada de la misma que es el programa.

Desde nuestro punto de vista un tanto diferente, el proceso de desarrollo de software es el de, dada una descripción informal del problema a ser resuelto, construir una solución del mismo descripta en un determinado lenguaje a la que denominaremos programa; solución que debe obviamente poseer como mínimo la propiedad de ser una función recursiva -es decir Turing-computable-.

La diferencia estriba en que, en el metamodelo del proceso de desarrollo de software que intentamos construir, haremos una estricta distinción entre la especificación formal de un problema y la especificación formal de una solución del mismo. Distinción ésta que se expresa algunas veces hablando de especificaciones no ejecutables y ejecutables.

Una especificación es una expresión de la relación que une datos con resultados en el ámbito de un problema determinado, mientras que un programa es una función que para cada dato "calcula" el resultado correspondiente, lo que supone algo más que dar sencillamente propiedades de dicho resultado que permitan realizar una elección apropiada del mismo.
Para aclarar ideas analicemos el caso de dos metodologías de programación muy conocidas, la de Bauer [28] y la de Jackson [29].

En la metodología de Bauer, éste define un lenguaje de amplio espectro compuesto por un nivel muy alto o de especificación que es un lenguaje de primer orden, un nivel intermedio compuesto por descripciones recursivas de funciones y un nivel de implementación que es un lenguaje algorítmico del tipo "do-while".

Se proveen ademas dos conjuntos de transformaciones, cada una de las cuales esta demostrada formalmente, uno para transformar la especificación del problema en el nivel muy alto en una solución escrita en el nivel de descripciones recursivas de funciones y otro para transformar este último en un programa escrito en el lenguaje de implementación. De esta manera y gracias a las demostraciones que acompañan a dichas transformaciones, cuando se obtiene el programa en cuestión también se cuenta con la demostración de que éste satisface a su especificación en primer orden.

Ahora bien, siempre se puede transformar una especificación del segundo nivel en un programa escrito en el lenguaje del tercero, en cambio no siempre es posible lograr una transformación del primer nivel en el segundo. Una representación de esta metodología es la expuesta en la figura 4.

Podemos decir, entonces, que los sistemas linguísticos involucrados en la metodología de Bauer son: un subconjunto del lenguaje natural, un lenguaje de primer orden, las descripciones recursivas de funciones y un lenguaje "do-while".

El problema de la diferencia de comportamiento entre las dos transformaciones estriba en que la segunda es siempre posible pues cualquier expresión expresable en un lenguaje "do-while" es modelo de una expresión escrita mediante descripciones recursivas de funciones, en cambio en el caso de la primera transformación no ocurre lo mismo ya que las únicas

Como Turski y Lehman, creemos que el proceso de desarrollo de software se basa fundamentalmente, como parece ser el caso del "problem solving" en general, en el método conocido como "divide and conquer", así como en una serie de transformaciones lingüísticas.

Este trabajo es solo una descripción informal de las ideas y metas que motivan los trabajos que estan desarrollando los autores respecto de la teoría algebraica de problemas y su utilización como modelo standard de un sistema formal que sirva para construir un metamodelo del proceso del desarrollo de software y cuyo cálculo deductivo sirva como un cálculo de diseño y programación del estilo del propuesto por Sintzoff ([9] a [13]) solo que a diferencia de aquél el desarrollo de dicho cálculo será llevado a cabo a partir de una semántica formalmente establecida por la mencionada teoría algebraica.

Las dos transformaciones de Jackson son siempre posibles -si pudo ser construída la estructura arborescente- ya que tanto el subdominio de los tipos de datos construíbles con los tipos primitivos de Jackson como el subdominio de los programas derivables utilizando esta metodología son modelo del dominio de los lenguajes regulares [30].

Entonces, describir la solución de un problema mediante tipos primitivos de Jackson y relaciones funcionales, es equivalente a describirla mediante un lenguaje regular y como sabemos no todo programa es describible mediante ese tipo de lenguaje.

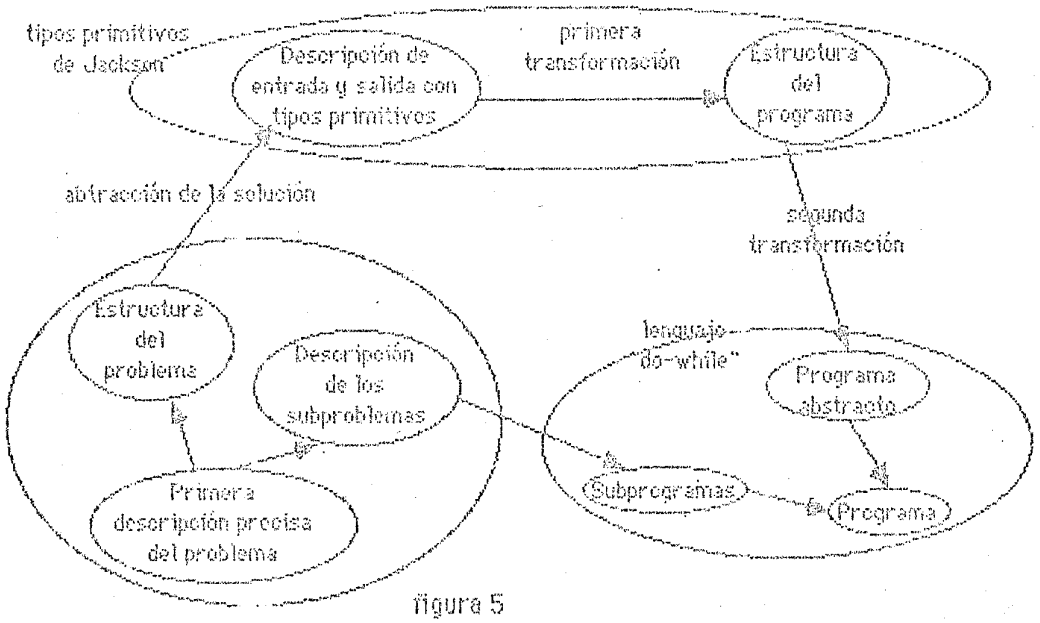

figura 5

Es decir, debido a la expresividad restringida de los lenguajes regulares los problemas de la metodología de Jackson respecto de la no derivabilidad de programas a partir de especificaciones del problema se situan antes del proceso mismo de derivación. Utilizando la metodología de Jackson, uno debe resolver de antemano el problema y escribir dichas soluciones en forma de programa -como estructuras arborescentes pero programas al fin-.

Distinguimos así en el proceso de desarrollo del software, aparte del paso previo a la primera descripción precisa del problema que consta de actividades que podrían ser descriptas epistemológicamente como de interpretación y elucidación, tres tipos de transformaciones: una -que puede incluir descomposición- de traducción entre representaciones de la especificación del problema expresadas en sistemas linguísticos distintos, otra de construcción de soluciones a partir de especificaciones -que también puede involucrar traducción- y por último una tercera -que puede incluir composición- de traducción entre representaciones de soluciones expresadas en sistemas linguísticos distintos.

Para comprender en profundidad la diferencia existente entre la especificación de un problema y una solución del mismo deberemos presentar -por lo menos informalmente- una teoría algebraica de problemas que está siendo desarrollada por los autores [1] para explicar formalmente los procesos de descomposición, composición, abstracción, refinamientos sucesivos, etc., comunes a toda metodología de programación y por lo tanto subyacentes en el proceso de desarrollo de software.

5- Descripción informal de la Teoría Algebraica de Problemas.

La teoría algebraica de problemas está basada en las ideas desarrolladas por Polya y en trabajos sobre una teoría matemática de problemas desarrollada por P. A. S. Veloso.

Un problema para esta teoría, es una cuadrúpla $P=\langle D,R,q,I\rangle$ donde D es el dominio de los datos y R el de los resultados, mientras que q es una relación binaria de D en R que no es otra cosa que la especificación del problema, es decir, un elemento d del dominio de datos D y un elemento r del dominio de resultados R estan en la relación q si y solo si r es el resultado esperado para d en ese problema. En otras palabras q es lo que Polya denomina la condición del problema.

Por ejemplo si se trata de calcular las raíces de polinomios de segundo grado, el dominio de datos serán polinomios de la forma ax^2+bx+c y el dominio de resultados será el conjunto Z de los números complejos y la relación q relacionará a cada polinomio d perteneciente a D con un par de números de Z cada uno de los cuales utilizado como valor de x en d lo anule.

Graficamente un problema se podría representar, entonces, de la siguiente manera:

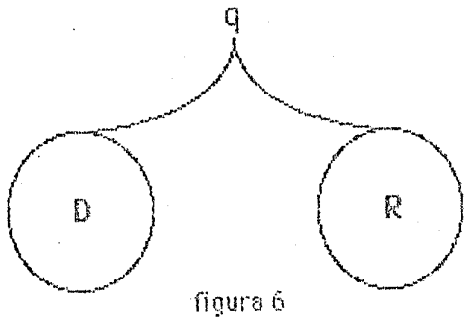

figura 6

Ahora bien, puede interesarnos por ejemplo calcular las raíces solo para polinomios de segundo grado con coeficientes reales que es obviamente un subconjunto del dominio de todos los polinomios de segundo grado.

En este caso diremos que el subconjunto de los polinomios de segundo grado con coeficientes reales es el "Dominio de instancias de interés" de nuestro problema.

Dicho dominio de instancias de interés no es otra cosas que el cuarto elemento de nuestra cuadrupla y que ha sido simbolizado mediante I.

En nuestro diagrama tendremos entonces:

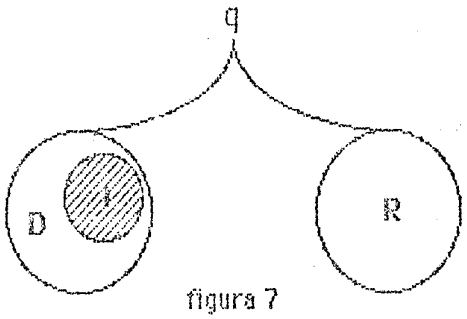

figura 7

Diremos que un problema es viable si y solo sí para cada dato d del dominio I de instancias de interés existe por lo menos un elemento r perteneciente al dominio de resultados R tal que el par $\langle d,r\rangle$ pertenezca a la condición q. Es decir q debe estar definida para todo el dominio de instancias de interés.

En nuestro ejemplo del cálculo de raíces para polinomios de segundo grado la situación será: D es el dominio de todos los polinomios de segundo grado, I el subconjunto de D de los polinomios de segundo grado con coeficientes reales, R el conjunto Z de los números complejos y q una relación que tendrá dos arcos $\langle d,r\rangle$ y $\langle d,r'\rangle$ para cada polinomio d de I que le "asignarán" cada una de sus dos raíces r y r'.

Formalmente se puede escribir la condición de viabilidad de la forma:

$$(\forall d)(d\in I\rightarrow(\exists r)(r\in R\wedge q(d,r)))$$

Otra forma de ejemplificar un problema podría ser la siguiente : tomando un sistema de coordenadas cartesianas ortogonales se puede tomar como D a un subconjunto de las abscisas; como R un subconjunto de las ordenadas e I será entonces algún conjunto de puntos del eje de las abscisas contenido en D.

En esta representación, q será una región cuya proyección sobre las abscisas deberá contener a I, si el problema es viable. Por ejemplo el problema representado en la figura (8-a) es evidentemente un problema viable mientras que los representados en la figuras (8-b) y (8-c) no lo son.

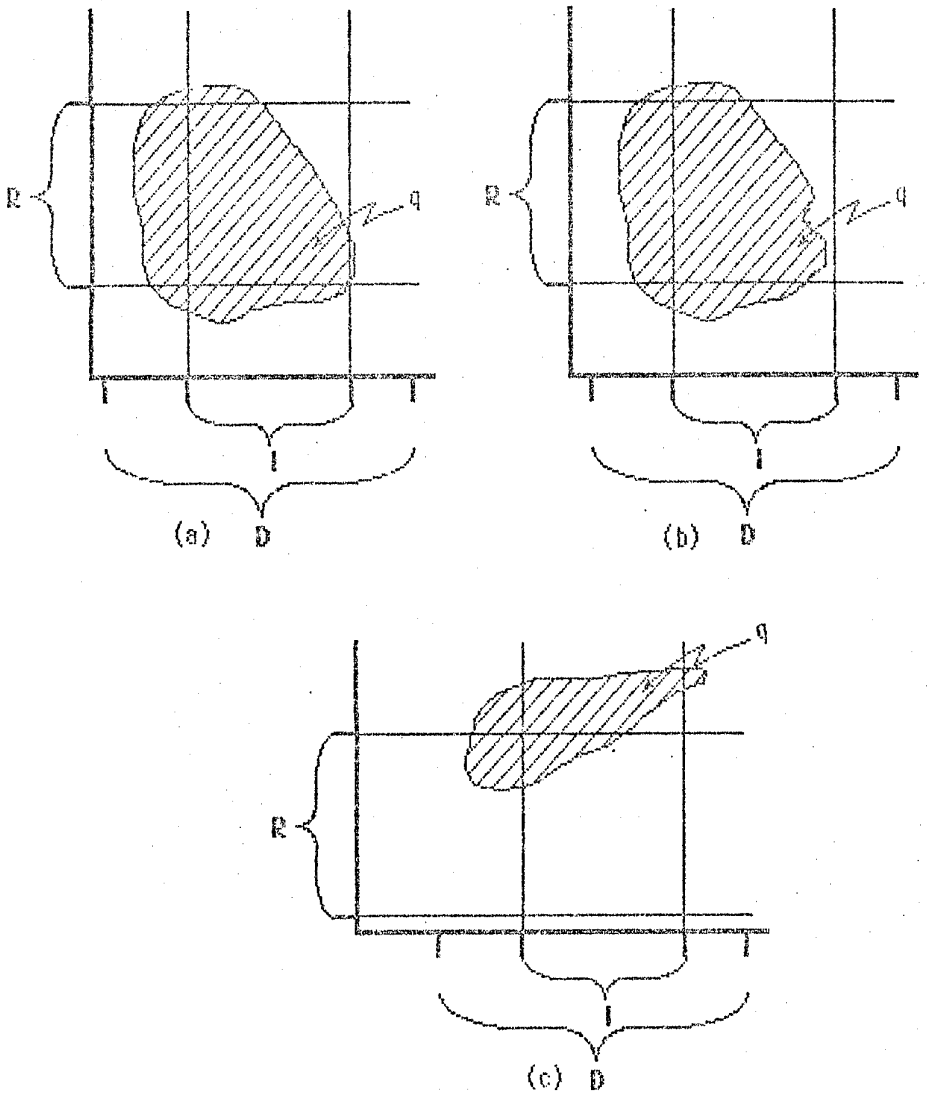

figura 8

Ahora bien, en general, para cada dato d de I habrá más de un resultado r de R relacionado con éste por la condición q, es decir más de un par de la condición q tendrá como primer elemento al mismo elemento d de I.

Obviamente, en nuestra representación, todos los pares de la relación q que tienen como primer elemento a un mismo dato d son aquellos representados por el segmento de la recta paralela a las ordenadas que pasan por el punto d, definido por la intersección de esta última con la región que representa a q (figura 9).

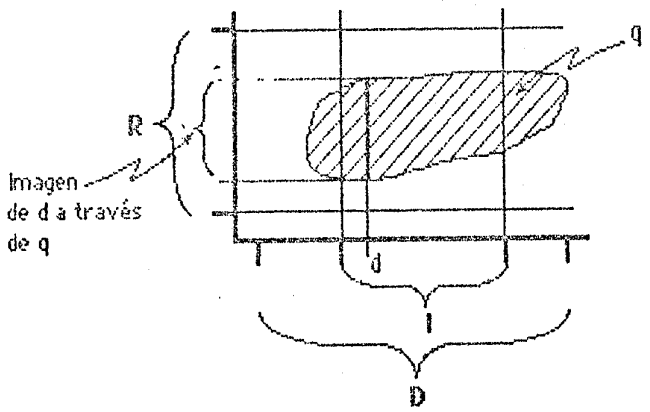

figura 9

Ahora bien, ¿que significa una solución para el problema P? Parece natural pensar que es una subregión de q tal que a cada punto d en I le asigne uno o más puntos r del dominio de resultados, por ejemplo a cada polinomio de segundo grado con coeficientes enteros le asigne sus dos raíces.

Sin embargo hay en esta definición de solución un hecho un tanto molesto:
q misma es una solución. ¿Qué significa esto?, que todo problema viable enunciado ya está resuelto.

Por otra parte es fácil observar que la región que representa a q es la de los puntos del producto cartesiano de $D \times R$ que cumplan con la condición de viabilidad $(\forall d)(d \in I \rightarrow (\exists r)(r \in R \wedge q(d,r)))$. ¿Qué significa el cuantificador existencial en esta región? que para un dato dado existe algún resultado relacionado con él por la condición, o lo que es lo mismo, para un dato dado d existen pares $\langle d,r \rangle$ en la condición del problema, y uno debe elegir cuales le interesan.

En el caso del problema del cálculo de raíces de polinomios de segundo grado, uno debe elegir cuál raíz le interesa, por ejemplo si se tratara de una aplicación a problemas de tiro vertical solo interesarían las raíces positivas.

Ahora bien, la elección de algún arco de la condición que contenga a un dato dado no parece poder generalizarse a un método efectivo (en algun sentido "razonable") de obtención de soluciones para problemas.

Pensemos nuevamente en el caso de los polinomios de segundo grado, un típico algoritmo del tipo "British Museum" como el de tomar elementos de Z (números complejos) e ir probando cuales anulan a un determinado polinomio no parece el método más feliz para resolver el problema del cálculo de raíces aún cuando dicho algoritmo fuera aplicable sobre un conjunto no recursivamente numerable como Z.

Pensemos por un momento en que dado un problema, para cada dato d en I, aplicamos el algoritmo del British Museum para elegir un par $\langle d,r\rangle$ de q que lo contenga. En nuestra representación (figura 10) esto significa que para cada punto del segmento I elegiremos como resultado la proyección de un punto del segmento de perpendicular a I en d definido por la intersección de la misma con la región que representa a q.

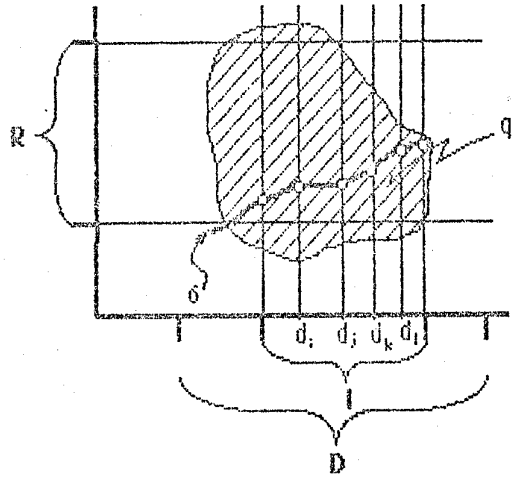

figura 10

Obsérvese que las elecciones de dichos puntos sobre tales segmentos definen una función de D en R. Ahora bien, poseer una función es muy distinto de tener que elegir puntos en una relación.

En nuestro ejemplo de cálculo de raíces, es muy distinto utilizar el algoritmo del British Museum, a saber que para cualquier polinomio de segundo grado las funciones $(-b+\sqrt{(b-4ac)})/2a$ y $(-b-\sqrt{(b-4ac)})/2a$ calculan las dos raíces posibles.

Diremos entonces que una solución es una función total de D en R tal que todo punto de I satisface a la condición q. En otras palabras, si denominamos σ a una solución, se debe cumplir que para todo d en I que $q(d,\sigma(d))$. Es decir que σ pertenece al conjunto de las funciones de D en R tales que para todo punto d en I estan contenidas en la relación q. Un lector familiarizado con la lógica notará inmediatamente que una solución no es otra cosa que una de las funciones de Skolem de la condición de viabilidad.

Formalmente diremos que un problema es soluble si y solo sí se satisface la fórmula $(\exists\sigma)(\sigma\in R_P^{D_P} \wedge (\forall d)(\, d\in I_P \rightarrow q_P(d,\sigma(d))))$ que no es otra cosa que la skolemización de la condición de viabilidad.

Como es sabido el axioma de elección asegura la existencia de tales funciones de manera que la aserción: "un problema es viable" es equivalente a: "un problema es soluble". Dicho en otras palabras, todo problema viable tiene por lo menos una solución.

Sin embargo, en "problem solving" el cálculo efectivo de una solución es un proceso tipicamente constructivo más que de elección. Más aún, el método del British Museum , cuando fuese aplicable, deberá en general ser rechazado por elementales razones de eficiencia. Es decir, ni elección arbitraria ni inspección exhaustiva resultan reconstrucciones racionales del proceso de obtener una solución de un problema. Sin embargo el problema del "problem solving" es más complicado de lo que parece.

Supongamos el problema de encontrar la superficie de un círculo, o como lo enunciaría un jónico: dado un círculo, encontrar el cuadrado cuya superficie es igual a la de aquel; se trata del problema clásico de la cuadratura del círculo.

Dado que podemos especificarlo, el axioma de elección asegura que existe una solución -es decir, dado que es un problema viable es también soluble- y nosotros conocemos esa solución que es obviamente: "el cuadrado cuya superficie es igual a la de un círculo de radio r es aquel cuyo lado es $l=+(\sqrt{\pi})r$".

Desgraciadamente para los jónicos esta solución no tenía sentido pués sencillamente desconocían los irracionales. Por lo tanto y siguiendo un paradigma de la época, intentaron resolver este problema mediante una construcción geométrica utilizando regla no graduada y compás que, como sabemos, no existe.

Entonces si los jónicos hubieran conocido el axioma de elección se hubieran sentido más frustrados de lo que se sintieron al no encontrar una solución al problema de la cuadratura del círculo.

El problema reside en que el axioma de elección no distingue diferencias constructivas en cuanto a las funciones incluídas en una relación. Si los jónicos hubieran encontrado una cuadratriz (como la de Hipías para el caso de la trisección del ángulo), dado que para cada círculo existe un solo cuadrado con superficie igual a la de él, la solución obtenida con esa supuesta cuadratriz sería extensionalmente la misma que la obtenida mediante el número π. Sin embargo las dos serían diferentes en el sentido intensional.

Como en el caso de la cuadratura del círculo, las herramientas de que disponemos para encontrar o calcular una solución pueden hacer a un problema insoluble a pesar de lo enunciado por el axioma de elección.

Volviendo a nuestra teoría algebraica de problemas, lo expuesto anteriormente se puede tratar formalmente distinguiendo dos conceptos de solubilidad.

Como hemos visto, un enunciado equivalente del axioma de elección, en el ámbito de la tería de problemas es: "Dado un problema, decir que este es viable es equivalente a decir que es soluble". Este es el concepto no restricto de solubilidad, basados en el mismo denominaremos espacio de soluciones de un problema P -que denotaremos Ω_P- a la clase de todas las funciones de Skolem de su condición Q.

El otro concepto de solubilidad, que es el concepto restricto, es el siguiente: denominemos $\mathcal{A}_\alpha$ a la subclase de las funciones de $\mathfrak{U}^{\mathfrak{U}}$ que posean la propiedad α y $\mathcal{A}_{\alpha,P}$ a la clase de funciones obtenida restringiendo a cada función de $\mathcal{A}_\alpha$ al dominio de datos del problema P. Aquí $\mathfrak{U}$ es un conjunto denominado "universo del discurso" que esta formado por la unión de los dominios de datos y resultados de todos los posibles problemas de que se esta tratando.

Diremos entonces que P es α-soluble si tiene alguna solución que pertenezca a la clase de funciones $\mathcal{A}_{\alpha,P}$. Denominaremos a la clase $\mathcal{A}_\alpha$ "Contexto de admisibilidad" y a la clase $\mathcal{A}_{\alpha,P}$ "clase de funciones admisibles" del problema P.

A la luz del concepto de α-solubilidad el problema de la cuadratura del círculo puede enunciarse de la siguiente manera: La cuadratura del círculo es un problema viable y por lo tanto soluble pero no es α-soluble para el caso en que el contexto de admisibilidad sea la clase de las construcciones geométricas con regla no graduada y compás.

Estas ideas nos llevan a la definición del espacio de soluciones admisibles de un problema P para un contexto dado de admisibilidad $\mathcal{A}_\alpha$, que denotaremos como $\Psi_{\alpha,P}$. Obviamente será $\Psi_{\alpha,P}=\mathcal{A}_{\alpha,P}\cap\Omega_P$.

Es importante notar que la propiedad α puede ser extensional, como por ejemplo: "continua", "derivable", "creciente", "calculable", "construible con regla no graduada y compás", etc.; o intensional como: "eficiente", "elegante", "fácil", etc.

Podemos citar otro ejemplo de α-solubilidad mucho más interesante para nosotros. Si nuestro objetivo es que una solución de un problema dado sea calculada por una computadora -es decir sea un programa-, las funciones pertenecientes al contexto de admisibilidad deberán ante todo poseer la propiedad de calculabilidad (Turing-computabilidad) pero, si además pretendemos eficiencia, necesitamos que tales programas tengan una complejidad aceptable -por ejemplo polinómica- propiedad esta que también deberían poseer las funciones pertenecientes al contexto de admisibilidad.

Este contexto de admisibilidad será utilizado en las secciones siguientes, lo denominamos el de las "feasible algorithmic Skolem functions" y se dirá que toda función que pertenezca a él es **fas**.

Para consuelo -o desvelo- de los diseñadores de algoritmos, el axioma de elección no asegura la existencia de funciones **fas** en cualquier relación Q, es decir no necesariamente todo problema viable es **fas**-soluble.

Volvamos ahora sobre el concepto de problema, podemos formalizarlo un poco más diciendo que un problema es una cuadrupla de la forma: $P=< D_P, R_P, q_P, I_P >$, donde: $I_P \subseteq D_P$, $D_P \subseteq \mathcal{U}$.

Supongamos tres problemas $P=< D_P, R_P, q_P, I_P >$, $Q=< D_Q, R_Q, q_Q, I_Q >$ y $S=< D_S, R_S, q_S, I_S >$, diremos que S es la "suma" de P y Q si y solo si $D_S=D_P \cup D_Q$, $R_S=R_P \cup R_Q$, $q_S=q_P \cup q_Q$ e $I_S=I_P \cup I_Q$.

Graficamente:

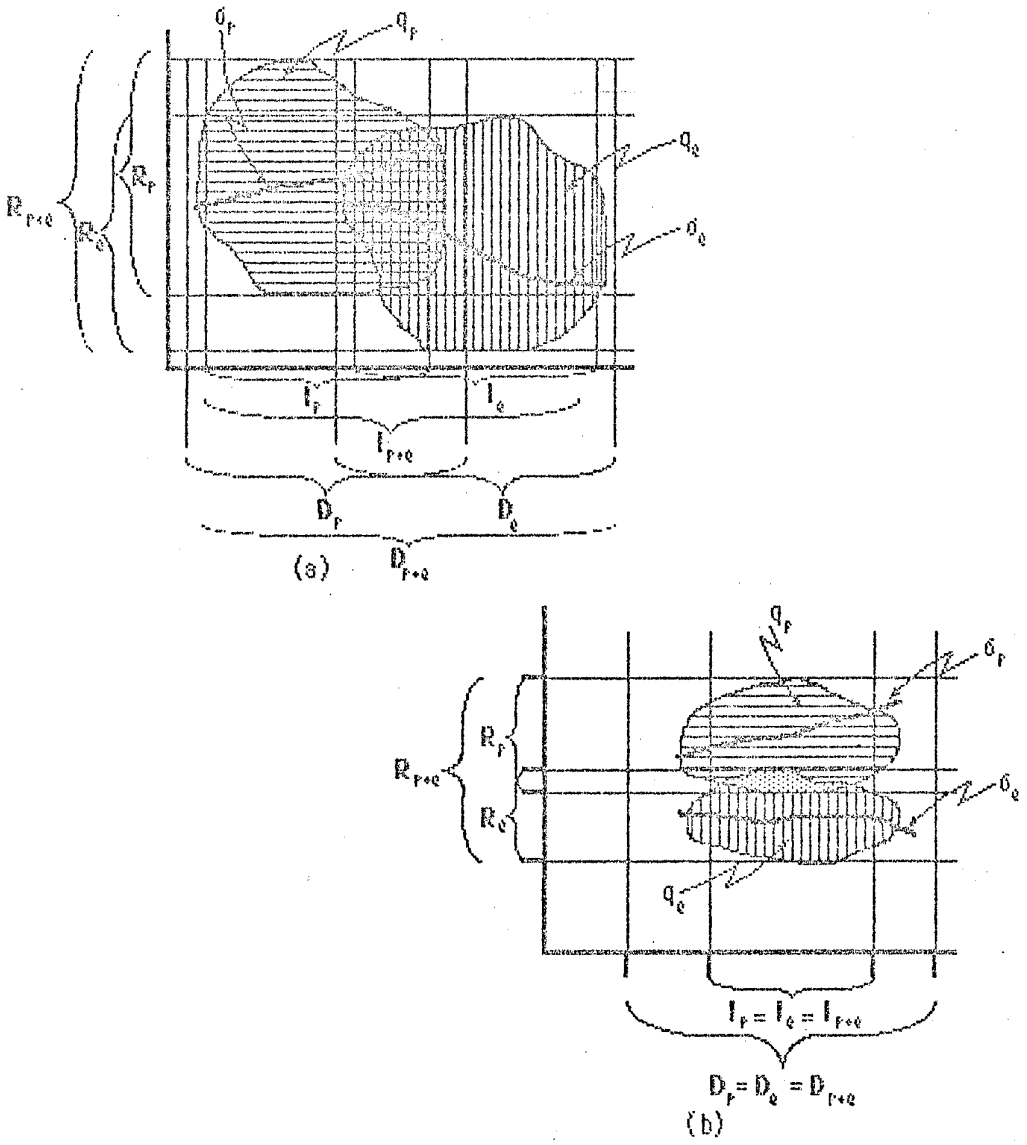

figura 11

La figura 11 muestra dos ejemplos de suma de problemas, en el caso (a) las problemas difieren en sus dominios de datos, resultados e instancias de interés, así como en sus condiciones, si bien ninguno de estos conjuntos son disjuntos; en el caso (b), en cambio, los problemas solo difieren en su relación y en su dominio de resultados.

Como se demuestra en [1] la suma de problemas es: conmutativa ($P+Q=Q+P$), asociativa ($P+(Q+R)=(P+Q)+R$) e idempotente ($P+P=P$). Se demuestra también en [1] la existencia de un elemento neutro para esta operación que denominaremos 0 y que no es otra cosa que el problema que tiene los dominios de datos, resultados e instancias de interés y la condición vacíos: $0=\langle\emptyset,\emptyset,\emptyset,\emptyset\rangle$.

Es obvio que se puede extender el concepto de suma al de sumatoria -que denotaremos $\Sigma_{x\in\mathcal{L}}$- de los problemas pertenecientes a un conjunto $\mathcal{L}$ dado, extendiendo sencillamente las uniones correspondientes a uniones generalizadas.

Definiremos ahora el concepto de "subproblema aditivo", diremos que un problema P es subproblema aditivo de otro S - que denotaremos $P\trianglelefteq S$ - si y solo sí existe un tercer problema Q tal que $P+Q=S$.

Obsérvese que en la figura 11 tanto en el caso (a) como en el (b) los problemas P y Q son evidentemente subproblemas aditivos de la suma $P+Q$ (por la definición de subproblema aditivo). Sin embargo existe una diferencia importante entre ambos casos, en el caso (a) ninguna de las soluciones de P ni de Q son soluciones de $P+Q$, en cambio en el caso (b) toda solución de P o de Q es solución de $P+Q$.

Diremos que en el caso (b) P y Q son subproblemas aditivos completos de $P+Q$ -hecho este que denotaremos $P\trianglelefteq_c P+Q$ y $Q\trianglelefteq_c P+Q$ -. Entonces un problema P es subproblema aditivo completo de otro S si y solo sí el espacio de soluciones de P no es vacío y esta incluido en el espacio de soluciones de S y los dominios de instancias de interés de P y S son iguales.

Si tomamos ahora el conjunto P_+ de todos los subproblemas aditivos de un problema P y analizamos sus propiedades veremos que dicho conjunto con la operación suma es un semigrupo, dado que la suma de todo par de problemas pertenecientes a P_+ también pertenece a P_+, que dicho semigrupo es abeliano puesto que la suma de problemas es conmutativa y que, por último, dado que la suma de problemas es idempotente dicho semigrupo es un semireticulado superior [1].

Se puede deducir también que el supremo del semireticulado $< P_{+},+>$ no es otro que el problema P y que por lo tanto $\Sigma_{x \in P+} X=P$.

La interpretación gráfica del conjunto P_{+} es imposible de realizar en diagramas como los de la figura 11 ya que deberíamos dibujar todas las posibles subregiones de Q con sus proyecciones sobre ambos ejes, en particular aquella formada por cada uno de los puntos de la región que representa a Q.

Ahora bien, observemos que no solo la sumatoria de los problemas pertenecientes al conjunto P_{+} tiene como resultado al problema P, también la de ciertos subconjuntos propios de P_{+} produce P, por ejemplo los conjuntos de los problemas de la figura 11 que obviamente son un subconjunto de $(P+Q)_{+}$ también producen por suma el problema P que es el supremo del semireticulado superior $< P_{+},+>$.

Existen ciertos subconjuntos de P_{+} tales que la sumatoria de sus elementos es igual a P y que tienen además la propiedad de que si uno elimina un elemento de los mismos dicha sumatoria deja de producir P. Diremos que todo subconjunto de P_{+} que posea esa característica es una "base" de P_{+} y lo denotaremos mediante el símbolo $\mathfrak{B}_{P+}$.

Otro punto interesante es que si tomamos el conjunto P_{C+} de los subproblemas completos de P, éste también forma un semireticulado con la operación suma que es, a su vez, un subsemireticulado de $< P_{+},+>$ y que este nuevo semireticulado también posee bases.

Dentro de las bases de P_{+} y P_{C+} existen algunas que poseen la propiedad de que ninguno de sus elementos posee subproblemas en P_{+} o en P_{C+} respectivamente y que denotaremos ${}^{u}\mathfrak{B}_{P+}$ y ${}^{u}\mathfrak{B}_{Pc+}$.

Si analizamos P_+ detenidamente veremos que ${}^{\mu}\mathfrak{B}_{P_+}$ es la base compuesta por problemas cuya q está representa por un solo arco, es decir que graficamente ${}^{\mu}\mathfrak{B}_{P_+}$ estará representada por el conjunto formado por tales problemas compuestos de la siguiente manera. El dominio de datos de cada uno de estos será uno de los puntos del segmento $\mathbb{D}$, el dominio de resultados será uno de los puntos del segmento $\mathbb{R}$ que está relacionado con el punto correspondiente al dominio de datos por algún punto de la región que representa q, la condición estará formada por ese único punto de q y el dominio de instancias de interés será igual al dominio de datos. Es fácil ver que esta es la única base de P_+ cuyos elementos no poseen subproblemas que pertenezcan a P_+ y que por lo tanto ${}^{\mu}\mathfrak{B}_{P_+}$ es única.

¿Qué ocurre con las bases mismas del conjunto P_{C+} de los subproblemas completos de P?. Por definición un subproblema completo pertenecerá a una base ${}^{\mu}\mathfrak{B}_{P_{C+}}$ si y solo si no tiene subproblemas que pertenezcan al conjunto P_{C+} de los subproblemas completos de P.

Evidentemente, los únicos subproblemas completos que poseen esa propiedad son aquellos cuya relación q es una función ya que cualquiera de sus subproblemas no podrá ser subproblema completo de P pues éstos tendrán también condiciones funcionales pero es obvio que a cualquiera de ellas les faltará algún arco para poder contener una función que sea solución del problema P.

Por otra parte es evidente que existe más de una base de estas características, es decir que ${}^{\mu}\mathfrak{B}_{P_{C+}}$ no es única.

En el ejemplo del problema del cálculo de las raíces de polinomios de segundo grado con coeficientes reales, si tomamos los subproblemas: P_1="cáculo de las raíces de polinomios de segundo grado con coeficientes pares ", P_2="cálculo de las raíces de polinomios de segundo grado con coeficientes impares", P_3="cálculo de las raíces de polinomios de segundo grado con coeficientes racionales", P_4="cálculo de las raíces de polinomios de segundo grado con coeficientes irracionales", P_5="cálculo de las raíces obtenidas sumando el discriminante" y P_6="cálculo de las raíces de segundo grado obtenidas restando el discriminante", tendremos por ejemplo: P_1, P_2 y P_3 no son subproblemas completos, en cambio P_5 y P_6 si lo son; el conjunto $\{P_1, P_2, P_3, P_4\}$ no es una base de P_+, pues si quitamos del mismo P_1 o P_2 o P_1 y P_2 el conjunto $\{P_3, P_4\}$ sigue produciendo por sumatoria P_+, en cambio $\{P_3, P_4\}$ si es una base de P_+

y el conjunto $\{P_5, P_6\}$ es una base del conjunto P_{C+} de los subproblemas completos de P.

Hasta aquí hemos tratado la suma de problemas teniendo en cuenta las condiciones Q de los sumandos y de la suma, analicémosla, ahora, respecto de las soluciones.

El tratamiento del comportamiento de las soluciones en la operación suma es muy importante pues la idea de la definición de esta operación así como la del producto, que se hará más adelante, es la de explicar la solución de problemas por descomposición. Si descomponemos por ejemplo un problema P en sus subproblemas aditivos P_1, P_2 y P_3 que por suma producen P, éste será un método de resolución si, conocidas soluciones para aquéllos, se puede calcular alguna solución para este último.

¿Qué se puede decir del espacio de soluciones de la suma a partir de los espacios de soluciones de los sumandos?.

Tomemos tres problemas P, Q y S tales que P+Q = S como los representados en la figura 12.

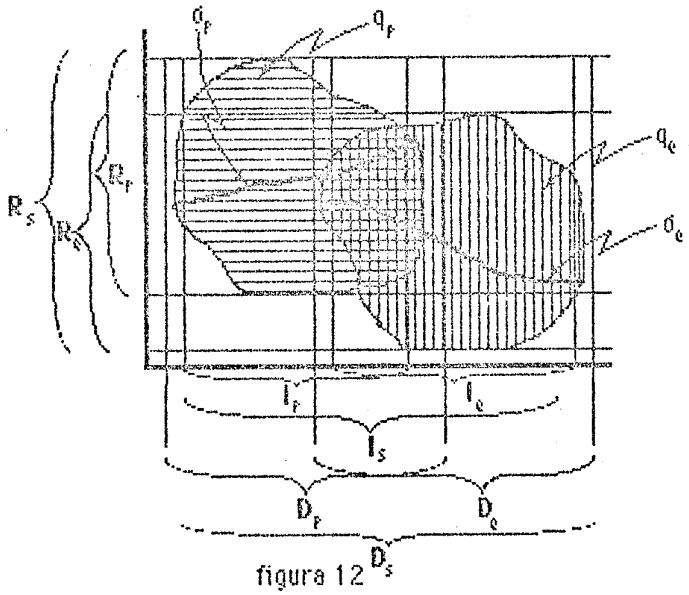

figura 12

Si tomamos un par de soluciones $\sigma_P \in \Omega_P$ y $\sigma_Q \in \Omega_Q$, veremos que si realizamos su unión -tomando a las funciones σ_P y σ_Q como conjuntos de pares ordenados- el resultado no es una función, ya que en la región $q_P \cap q_Q$ se tendría para cada punto de $I_P \cap I_Q$ dos valores, uno dado por σ_P y otro por σ_Q. Por lo tanto si se extendiera el concepto de suma a las soluciones, $\sigma_P + \sigma_Q$ no es necesariamente una solución de S si no más bien la condición $\sigma_P \cup \sigma_Q$ de un subproblema completo R de S cuyas soluciones, que también son soluciones de S, son las funciones de Skolem

de su condición de viabilidad. Por lo tanto si dejamos las cosas como hasta aquí, aún conociendo soluciones para P y para Q y sabiendo que P+Q=S, para encontrar soluciones de S a partir de las de P y Q deberíamos resolver algún subproblema completo de S obtenido como R a partir de la unión de una solución de P y una de Q.

Una primera observación interesante es que si las primeras proyecciones de σ_P y σ_Q, es decir sus preimagenes no se intersecan dentro del segmento $I_P \cup I_Q$ -para lo cual una condición necesaria pero no suficiente es que $I_P \cap I_Q = \emptyset$- la unión $\sigma_P \cup \sigma_Q$ será una función que es solución del problema S=P+Q.

Denominemos, ahora, $\tau(\sigma_P, \sigma_Q)$ al conjunto de funciones de Skolem de la relación $\sigma_P \cup \sigma_Q$ y $\eta(\sigma_P, \sigma_Q)$ al subconjunto de $\tau(\sigma_P, \sigma_Q)$ que cumplan con la propiedad α del contexto de admisibilidad $\mathcal{A}_\alpha$. Evidentemente la solución de un problema S por descomposición del mismo en dos problemas P y Q tales que P+Q=S, de los cuales se conoce un par de soluciones σ_P y σ_Q respectivamente, y que poseen la propiedad α del contexto de admisibilidad, será una de las funciones contenidas en el conjunto $\eta(\sigma_P, \sigma_Q)$.

Si las soluciones σ_P y σ_Q a las que nos estamos refiriendo son tales que sus preimagenes no se intersecan dentro del segmento $I_P \cup I_Q$, el conjunto $\eta(\sigma_P, \sigma_Q)$ será unitario y por lo tanto estaremos ante un caso ideal de solución del problema S por descomposición en dos subproblemas P y Q y posterior "combinación" de dos α-soluciones conocidas σ_P y σ_Q de P y Q respectivamente.

Ahora bien, ¿cómo tratar el caso en el que la intersección de la preimagenes de las α-soluciones σ_P y σ_Q no sean disjuntas sobre el segmento $I_P \cup I_Q$?. Una posibilidad es elegir un subconjunto K de $D_P \cap D_Q$ y definir la operación $\sigma_P +_K \sigma_Q$ de la siguiente manera: $\sigma_P +_K \sigma_Q = \sigma_P$ para todo punto en $D_P - K$ y $\sigma_P +_K \sigma_Q = \sigma_Q$ para todo punto en $(D_P - D_Q) \cup K$.

Evidentemente no nos hemos escapado de la necesidad de "elegir" nuevamente entre la solución para P y la solución para Q en el conjunto $D_P \cap D_Q$. Sin embargo dado que conocemos una solución σ_P para P y otra σ_Q para Q, el problema de la elección queda restringido a elegir un subconjunto K de $D_P \cap D_Q$. Si $K=\emptyset$ entonces para todo D_P se utiliza como solución σ_P y para

$D_P - D_Q$, σ_Q ; si $K = D_P \cap D_Q$ entonces se utiliza σ_P en $D_P - D_Q$ y σ_Q en todo D_Q.

Entonces, dada una descomposición de un problema, si contamos con soluciones de los subproblemas obtenidos, lo que hemos hecho es explicar la construcción de funciones de Skolem de la suma.

Graficamente, podemos representar esta situación de la siguiente manera:

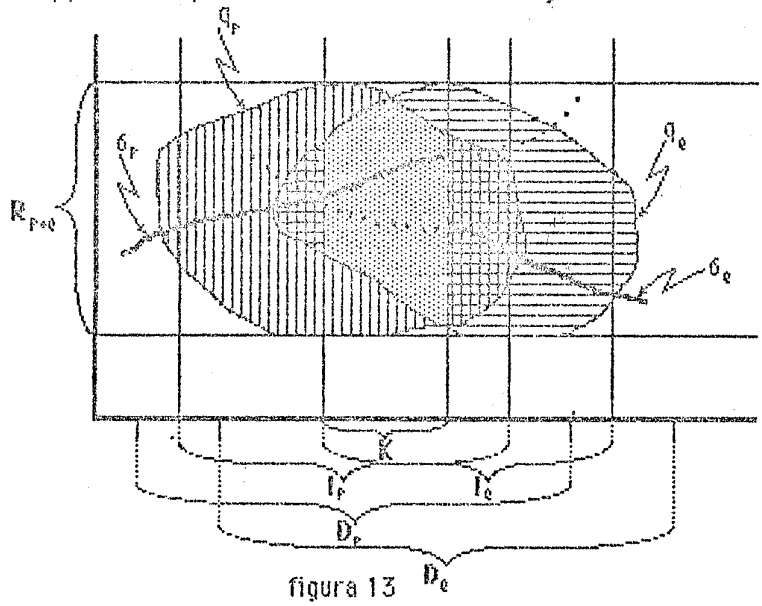

figura 13

Resumiendo: si los subproblemas, producto de la descomposición, son disjuntos respecto de sus dominios de datos, entonces, dadas soluciones para los mismos la solución por composición es simplemente la unión de dichas soluciones. Si los subproblemas en cuestión no son disjuntos respecto de sus dominios de datos nos encontramos que la unión de las soluciones de aquellos es una solución por composición, fuera de la intersección, dentro de la misma dicha unión tiene como resultado una relación que, si bien es una subrelación de la condición del problema original, no se ajusta a nuestra definición de solución; digamos entonces que la solución no está determinada en la intersección ya que puede ser cualquiera de las funciones de Skolem de dicha subrelación. En este caso, o bien contamos con una solución para el subproblema cuyo dominio de datos es la intersección de los dominios de datos de los productos de la descomposición, su dominio de resultados la unión de los dominios de resultados de los mismos, su condición la relación obtenida uniendo una solución por cada subproblema producto de la descomposición y su dominio de instancias de interés la intersección de los dominios de instancias de interés de los mismos; o bien debemos construir una que sea obviamente una de las funciones de Skolem de la relación obtenida mediante la unión de una solución para cada uno de los subproblemas producto de la descomposición.

Obsérvese que este subproblema puede ser vacuamente viable, en caso extremo, por el hecho de que la intersección de los dominios de instancias de interés de los productos de descomposición sea vacío. Si este es el caso, la solución de la composición sobre el dominio de instancias de interés de la suma estará determinada en todos sus puntos y por lo tanto, a los fines prácticos estaremos en el primer caso.

Si el subproblema arriba definido es viable, entonces tendremos dos opciones: o decimos que para cierto dominio la solución no esta determinada (o en otras palabras existe pero es no determinística) o elegimos un conjunto K ad-hoc.

Un resultado interesante de la teoría algebraica es que, como se demuestra en [1], la unión de todos los conjuntos de soluciones obtenidas eligiendo cada una de las posibles K de la intersección de los dominios de datos de los sumandos es igual al espacio de soluciones de la suma.

Obsérvese que lo que hemos hecho hasta aquí es descomponer un problema en un conjunto de subproblemas aditivos cuya suma tenga como resultado dicho problema y para cada uno de los cuales o bién conozcamos o, lo que es lo mismo, sepamos contruir una solución. Luego analizamos como se puede obtener una solución del problema original por unión de las soluciones de dichos subproblemas.

Ahora bien, una forma interesante de ver este resultdo es la siguiente: dado un problema P, se induce un conjunto $\mathbb{L}$ de subproblemas aditivos mediante un cierto predicado $\delta(\mathbb{L})$, que denominamos predicado de descomposición.

Definamos ahora un problema abstracto P_δ de la siguiente manera: por cada subproblema Q en $\mathbb{L}$ definamos un par ordenado de la condición q_{P_δ} de P_δ cuyo primer elemento será la preimagen de q_Q y cuyo segundo elemento será la imagen de q_Q. El dominio de datos D_{P_δ} será obviamente el conjunto de los primeros elementos de los pares ordenados que componen q_{P_δ} y el dominio R_{P_δ} el de los segundos elementos de los mismos.

Entonces, conocidas soluciones para cada subproblema $Q_i \in \mathbb{L}$ de P y si las q_i de los problemas de $\mathbb{L}$ constituyen una cobertura de q, su unión, éligiendo o no algún K para cada intersección de dominios de datos, produce una solución, deterministica o no de P.

¿Por que definimos el problema abstracto P_δ?.

Analicemos el caso representado en la figura 14, en donde no hemos representado por simplicidad los dominios, sino solo las condiciones y donde suponemos que la suma de los subproblemas incluídos en $\mathbb{L}$, $Q_1+Q_2+Q_3+Q_4+Q_5=P$. La condición del problema abstracto P_δ estará representada aquí por un punto para cada Q_i en $\mathbb{L}$.

Observemos que en este caso Q_5 no aporta nada a la solución de P, si tenemos en cuenta Q_2 y Q_3 o viceversa.

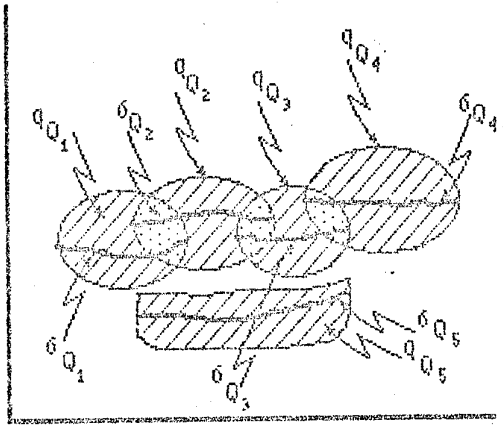

figura 14

Es decir deberemos elegir a Q_2 y Q_3 o Q_5 cosa que es evidentemente la elección que define cada una de las funciones de Skolem de la condición q_{P_δ} del problema abstracto P_δ.

En otras palabras basta con unir las soluciones correspondientes a los subproblemas de P pertenecientes a $\mathbb{L}$ representados por aquellos arcos de la condición q_{P_δ} de P_δ que pertenezcan a una solución de este último.

Ahora bien, ¿ se cumple lo hasta aquí expuesto para cualquier conjunto de subproblemas aditivos de P?.

Como lo demuestra un teorema de esta teoría [1] esto se cumple si el conjunto $\mathbb{L}$ contiene una base $\mathbb{B}_{P_+}$ del conjunto de subproblemas aditivos de P.

Entonces, y permitiéndonos algunas libertades en el lenguaje, la unión de las soluciones de los subproblemas de P representados por cada uno de los arcos de una solución del problema abstracto P_δ es una solución de P, si el conjunto de los subproblemas de P representados por cada uno de los arcos de la condición q_δ contiene una base $\mathbb{B}_{P_+}$ del conjunto de subproblemas aditivos de P.

Si meditamos un instante sobre este resultado, suponiendo que toda descomposición es aditiva, veremos que esto es lo que ocurre cuando por ejemplo escribimos un programa utilizando tipos abstractos de datos. Este programa representa una solución abstracta que es una solución del problema real cuando proveemos los "clusters" - es decir las soluciones- de cada uno de los tipos abstratos -es decir los subproblemas- utilizados.

Lo que estamos realmente haciendo, presentando resultados como el anterior, es explicar bajo que condiciones se puede construir una solución -función de Skolem- de la estructura abstracta de un problema dado y como -siempre que se conozcan soluciones para los subproblemas involucrados- se obtienen soluciones para un problema dado por descomposición de su especificación y posterior composición -en el sentido de combinación- de sus soluciones guiada por la solución de la estructura abstracta definida en la descomposición.

Es decir estamos dando una explicación rigurosa del metodo de resolución de problemas conocido como "descomposición por refinamientos sucesivos".

En realidad esta explicación es parcial pues, como dijimos en un párrafo anterior, estamos suponiendo que toda descomposición es aditiva y este no es necesariamentes el caso.

Supongamos que descomponemos un problema en dos subproblemas tales que el dominio de datos del primero de ellos es el dominio de datos de dicho problema, el dominio de resultado del otro es el dominio de resultados del mismo y su dominio de datos, el dominio de resultados del primer subproblema. Este es también un caso de descomposición muy común que es imposible explicar mediante el concepto de subproblema aditivo.

Si denotamos S al problema original y P y Q a los dos subproblemas descriptos en el párrafo anterior diremos que $P \bullet Q = S$ y que P y Q son subproblemas multiplicativos de S.

Definimos entonces al producto de problemas como una operación tal que: la condición es el producto relativo -en el sentido de teoría de conjuntos [14]- de las condiciones de los factores, el dominio de datos del mismo la preimagen de la condición del producto, el dominio de resultados el rango de tal condición, mientras que el dominio de instancias de interés de tal operación será el dominio del producto relativo de la condición del primer factor, restringida a su dominio de instancias de interés, por la condición del segundo factor.

Como se demuestra en [1], el producto de problemas no es conmutativo, es asociativo y distributivo por la izquierda y por la derecha con respecto a la suma y posee un elemento neutro $1=\langle \mathfrak{U}, \mathfrak{U}, \mathfrak{i}_{\mathfrak{U}}, \mathfrak{U} \rangle$ donde $\mathfrak{i}_{\mathfrak{U}}$ es la relación identidad sobre el universo del discurso $\mathfrak{U}$.

Se demuestra también que el producto de cualquier problema P por 0 (tanto por la derecha como por la izquierda) es igual a 0.

Para aclarar ideas tomemos dos problemas P y Q cuyas representaciones gráficas son las de la figura 15:

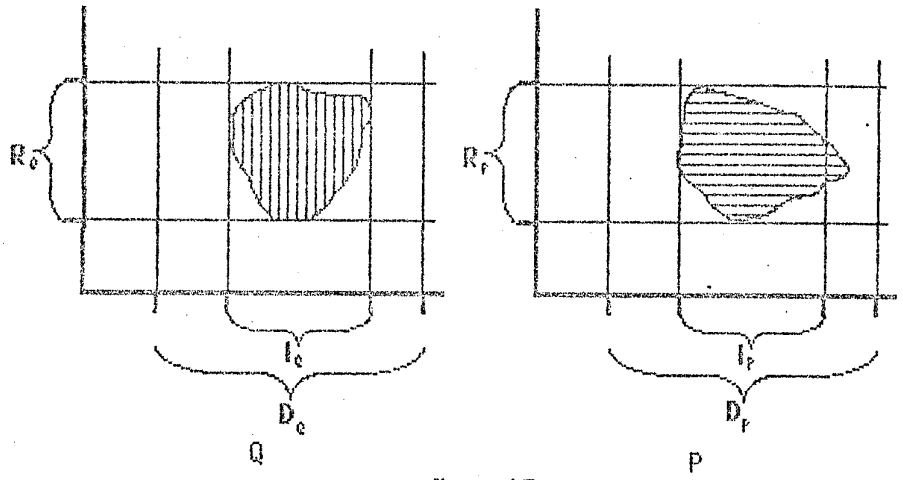

figura 15

a fin de producir el producto relativo de las condiciones rotemos 90º a la representación de Q.

El diagrama de la figura 16 será una representación gráfica del producto P∘Q.

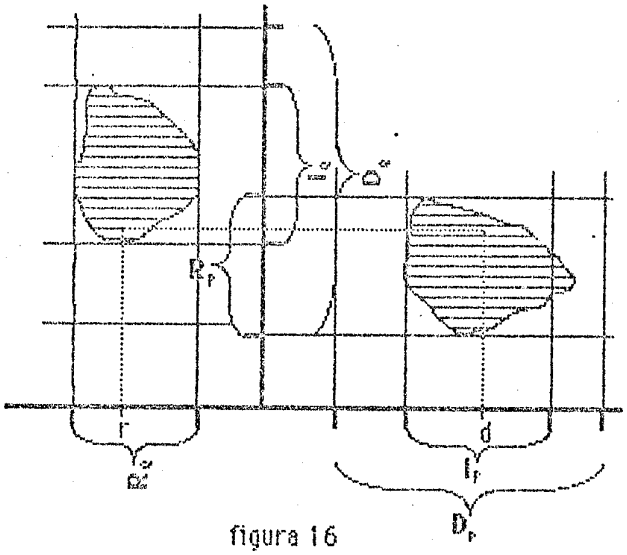

figura 16

Esta es una operación todavía no totalmente desarrollada en la teoría algebraica (ver [1]).

Los desarrollos que se estan llevando a cabo actualmente lo estan siendo en el mismo sentido que los realizados para la operación suma a fin de que estudiando las propiedades de los subproblemas multiplicativos se pueda llegar a explicar este nuevo tipo de descomposición.

Para simplificar la discusión que sigue denominaremos: ϕ_+ a la condición ya explicada que debe cumplirse para que la unión de las soluciones de los subproblemas de P representado por cada uno de los arcos de una solución del problema abstracto P_δ sea una solución de P (o sea ϕ_+ será: "el conjunto de los subproblemas de P representados por cada uno de los arcos de la condición q_{P_δ} contiene una base $\mathcal{B}_{P_+}$ del conjunto de subproblemas aditivos de P"), $\phi_\circledast$ a una condición similar para la solución por descomposición multiplicativa y por último $\phi = \phi_+ \vee \phi_\circledast$, a la condición de lo que denominaremos solución por descomposición generalizada o simplimente solución por descomposición.

Vemos que, cuando contemos con $\phi_\circledast$ y por lo tanto con ϕ, estaremos explicando bajo que condiciones se puede construir una solución -función de Skolem- de la estructura abstracta de un problema dado y como -siempre que se conozcan soluciones para los subproblemas involucrados- se obtendrán soluciones para un problema dado por descomposición (aditiva o multiplicativa) de su especificación y posterior composición -en el sentido de uniones y composiciones (de funciones)- de sus soluciones guiada por la solución de la estructura abstracta definida con la descomposición.

Denominemos ahora $\Delta_{P_\delta} P_i$ a la descomposición del problema P en una estructura abstracta P_δ formada por un conjunto L tal que $\delta(L)$ de subproblemas aditivos y multiplicativos P_i de dicho problema y $\nabla_{\sigma_\delta} \sigma_i$ a la composición -unión o composición de funciones- de soluciones σ_i de los P_i representados por arcos de una solución abstracta σ_δ del problema P_δ abstracto de P inducido por el predicado de descomposición δ.

Denotemos mediante $\sigma_i \leftarrow P_i$ el hecho de que σ_i sea solución o resuelva a P_i.

Entonces podremos escribir:

$$Si\ P = \Delta_{P_\delta} P_i \wedge \phi(q_{P_\delta}) \wedge \sigma_\delta \leftarrow P_\delta \wedge \sigma_i \leftarrow P_i \ \textit{para todo}\ P_i\ \textit{en}\ P_\delta\ \textit{entonces}\ \nabla_{\sigma_\delta} \sigma_i \leftarrow P$$

que será el enunciado del teorema fundamental de la resolución de problemas por descomposición por refinamientos sucesivos.

Si se medita un momento sobre la operación de descomposición de problemas Δ y la composición de soluciones ∇, se verá que existe una diferencia fundamental entre las dos. Esta diferencia se verá con más claridad en el caso de la suma.

En la descomposición Δ el único requerimiento necesario para asegurar que $P = \Delta_{P\delta} P_i$ será una descomposición "apropiada" para la construcción de soluciones de P, se refiere a una propiedad del conjunto $\mathcal{L}$ de subproblemas P_i inducido por el predicado de descomposición y no es otra que la ya mencionada de que tal conjunto contenga una base del conjunto P'_+ de los subproblemas aditivos de P.

Este requerimiento refleja el hecho de que una base de P_+ es un definiens suficiente de P ya que su suma tiene como resultado a P.

En realidad y con un criterio reduccionista, el teorema de la descomposición podría establecer que si un conjunto $\mathcal{L}$ de subproblemas aditivos P'_i de P es una base, entonces, tomando una solución para cada P_i en $\mathcal{L}$ se obtiene por unión discriminada de las mismas una solución para P.

¿Porqué entonces no es esa la hipótesis del teorema de la descomposición?, sencillamente por el hecho de que la hipótesis que hemos escogido tiene mayor contenido heurístico.

Obsérvese que al definir un problema abstracto sustituyendo cada subproblema del conjunto $\mathcal{L}$ inducido por un arco de su condición tendremos nuevamente un problema y siempre que el conjunto de condiciones q_i de los subproblemas P_i pertenecientes a $\mathcal{L}$ sea una cobertura de la condición q de P -o, lo que es una consecuencia, $\mathcal{L}$ contenga una base de P_+ -, una solución de dicho problema abstracto, una vez realizada la sustitución inversa de la arriba mencionada, no es otra cosa que una base de P_+.

La diferencia en contenido heurístico entre ambas hipótesis estriba en que en la elegida la elección de una base es el resultado de la resolución de un problema (el abstracto) y que cuando se define el mismo uno trata de que éste sea tal que su solución sea "fácil" , por ejemplo porque se conocen soluciones para problemas análogos.

En el caso de la operación de composición ∇, la sencilla unión de las soluciones de los problemas P_i no siempre tiene como resultado una función y por lo tanto no siempre produce una solución P.

Para comprender esta situación basta con analizar los subproblemas P_i pertenecientes a $\mathcal{L}$ por pares. Si dados dos problemas P_i y P_j y una solución σ_i y otra σ_j respectivamente, la unión $\sigma_i \cup \sigma_j$ será una solución para $P_i + P_j$ (y por lo tanto una función) si y sólo si $Im^{-1}\sigma_i \cap Im^{-1}\sigma_j = \emptyset$ (donde $Im^{-1}\sigma$ denota la preimagen de σ) dado que, en caso contrario para la región cuya proyección es dicha intersección, la unión producirá una relación, salvo que en la

misma σ_i sea igual a σ_j.

Un análisis de la figura 17 aclarará estas ideas.

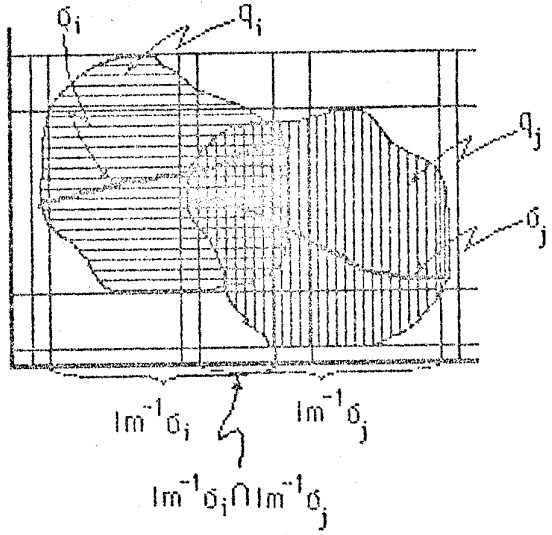

figura 17

Ahora bien, uno podría utilizar una unión discriminada en la que se podría tomar σ_i para la región cuya proyección es $Im^{-1}\sigma_i - Im^{-1}\sigma_j$, σ_j para la región cuya proyección es $Im^{-1}\sigma_j - Im^{-1}\sigma_i$ y alguna de las dos en subregiones distintas de la región cuya proyección es $Im^{-1}\sigma_i \cap Im^{-1}\sigma_j$ (por ejemplo mediante un subconjunto K de esta última región, en forma análoga a la tratada anteriormente).

Obsérvese que si la intersección de las preimagenes de las condiones q_i de los subproblemas P_i de la base de P_+ solución de P_δ son disjuntas dos a dos, la operación de composición $\nabla_{\sigma \in \sigma_i}$ sería, en lugar de una unión discriminada, sencillamente una unión de funciones. Este tipo de bases se denominarán bases ortogonales.

En el caso del desarrollo de software, si reformamos la condición Φ_+ del teorema de la descomposición exigiendo no solo que $\mathcal{L}$ incluya una base sino también que ésta sea ortogonal estaremos modelizando un caso de moldularización que posee la propiedad que los ingenieros de software denominan acoplamiento débil entre modulos.

6- El proceso de desarrollo de software a la luz de la teoría algebraica de problemas

Según lo expuesto en los puntos 3 y 4 , el proceso de desarrollo de software puede ser visto como una transformación que aplica a descripciones de problemas en programas que las satisfacen.

Desde el punto de vista de la teoría algebraica de problemas, la construcción de un programa que satisfaga la descripción de un problema puede ser visto como un problema.

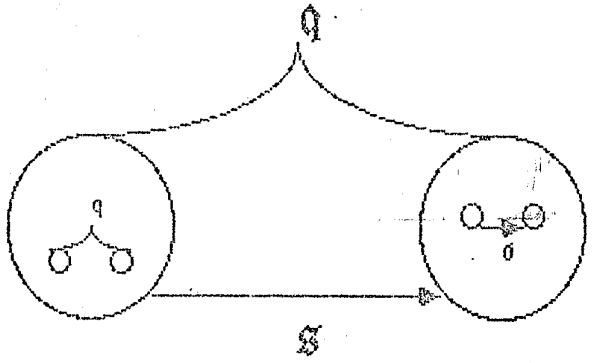

figura 18

Como puede deducirse de la figura 18, el problema de la construción de programas se puede definir de la siguiente manera: el dominio de datos está compuesto por problemas, el dominio de resultados por α-soluciones que pertenecen al contexto de admisibilidad de las "feasible algorithmic skolem functions" según lo expuesto en el punto 4; la condición será "ser una fas que satisfaga a la especificación q" y el dominio de instancias de interes será el de los problemas de algún tipo determinado.

Una solución $\mathcal{S}$ de este problema de la programación elegirá para cada problema uno y sólo un programa de entre todos los que satisfacen a su condición (especificación)q.

Tenemos, entonces, dos niveles de problemas, el primero: el de los problemas para los que deseamos construir programas (aplicaciones) y el segundo: el problema de la programación.

Si representamos al primer nivel, es decir a las aplicaciones, en un plano perpendicular al del papel y al segundo, es decir al problema de la programación, en el plano del papel y aplicamos las ideas de: descomposición, construcción de funciones de Skolem (soluciones) y composición según la teoría algebraica de problemas, obtendremos una figura como la 19.

En dicha figura, en los pasos de la izquierda, se descompone el problema P, cuya condición es q, en un conjunto $\{P_1, P_2, P_3, P_4, P_5\}$ cuyas condiciones son q_1, q_2, q_3, q_4, q_5 de manera tal que se cumple que: $P = (P_1 * P_2) + (P_3 * P_4 * P_5)$.

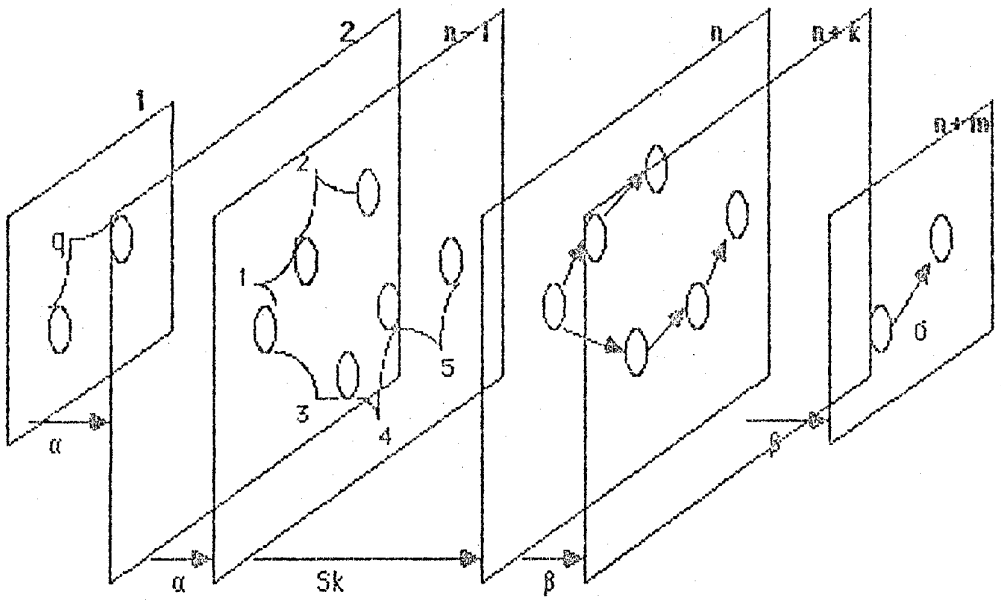

figura 19

Ahora bien, si suponemos que lo que denominamos P en el plano 1, es un modelo representacional de P [6] en un lenguaje A_1 y la expresión $(P_1 * P_2) + (P_3 * P_4 * P_5)$ en el plano n-1 es un modelo representacional de una descomposición de P en el lenguaje A_{n-1}, entonces los pasos α son traducciones (con descomposición) entre ambos modelos representacionales.

El paso entre el plano n-1 y el plano n puede incluir también una traducción entre sistemas linguísticos diferentes pero involucra además para cada q_i de un P_i la construcción de una función de Skolem .

Los pasos a partir del plano n son nuevamente traducciones, sólo que esta vez con composiciones como las explicadas por la operación ∇.

Obsérvese que:

.Los pasos α forman parte del proceso de abstracción.
.Los pasos β forman parte del proceso de reificación.
.El paso Sk es el de construcción de soluciones para los subproblemas P_i,

es decir $\sigma_i \leftarrow P_i$

La diferencia fundamental entre este modelo y el PW de Lehman [6] estriba en el hecho de que en el aquí presentado no existe un paso canónico para todo el proceso de desarrollo. Existe uno para la parte de abstracción y uno para la parte de reificación que, analizados con un grado de abstracción suficiente, podrían resultar iguales. Pero el paso Sk es un paso totalmente distinto

ya que en el mismo, aparte de la traducción, no tenemos descomposiciones ni composiciones sino construcciones de funciones de Skolem- que pertenecen al contexto de las "feassible algorithmic Skolem functions"- de las subespecificaciones q_i y de la condición q_δ del problema abstracto que representa a la descomposición Δ en el plano n-1.

El paso Sk es el corazón del proceso de programación y difiere del paso general representado en la figura 18 sólo en el hecho de que, por alguna razón, "sabemos construir" o "conocemos" las soluciones fas σ_i para cada uno de los subproblemas P_i y contamos con una solución fas abstracta para la descomposición.

Existen además otras diferencias respecto al modelo PW, que se refieren a la aplicación del principio de Turski [6]y [7], en los pasos α,Sk y β. De acuerdo a tal principio, en los pasos α,Sk y β se puede aplicar creatividad tanto a la elección de la transformación como a la del lenguaje destino de la misma, pero en nuestro modelo queda claro que dicha creatividad está restringida de la siguiente manera:

.En el caso de las tranformaciones α, la restricción estriba en que la descomposición debe satisfacer al predicado ϕ.

.En el caso de transformaciones β, la restricción estriba en que la composición debe satisfacer la " discriminación entre soluciones" de la operación ∇.

.Los lenguajes Λ_1, Λ_2,.... Λ_{n+m} deben ser tales que aquellas de sus construcciones linguísticas que representen la descomposición o composición (ej: concatenación, if then else, if con comandos guardados; etc) deben tener como semántica standard, la de las operaciones suma o producto de la teoría algebrsica de problemas.

7-Una descripción más profunda de un posible metamodelo del proceso de desarrollo de software

Supongamos que construimos un sistema formal cuya semántica standard este dada por la teoría algebraica de problemas, en otras palabras, del cual la teoría algebraica de problemas sea un modelo standard.

Este sistema formal estará basado en un lenguaje multisorted de primer orden con igualdad con dos sorts de interes, un sort x_i cuya interpretación serán problemas y un sort y_i cuya interpretación serán funciones.

Para el sort x_i existirán un par de constantes **0** y **1** cuyas interpretaciones serán el problema **0** y el problema **1** respectivamente y para el sort y_i existirá un par de constantes 0 y 1 cuyas interpretaciones serán la función con dominio vacio y la función identidad respectivamente.

Los términos t_i del sort x_i serán el resultado de aplicar reiteradamente los símbolos funcionales binarios $+$ y $\circ$, cuyas interpretaciones serán la suma y el producto de problemas respectivamente.

Los términos del sort y_i serán el resultado de aplicar reiteradamente los símbolos funcionales binarios $+_f$ y $\circ_f$ cuyas interpretaciones serán respectivamente la unión discriminada y el producto relativo de funciones, según lo discutido para la composición ∇.

Por último existirá un símbolo predicativo $\leftarrow(\tau_i, t_i)$ cuya interpretación será: "la composición ∇ aditiva y/o multiplicativa de soluciones resultado de la interpretación del término τ_i es una solución de la descomposición de problemas Δ resultado de la interpretación del término t_i.

Entonces:

- Cada uno de los modelos representacionales de una descomposición Δ expresada mediante construcciones linguísticas de un lenguaje en los planos 1 a n-1 de la figura 19 deberá ser modelo de un término t_i del sistema formal.
- Cada uno de los modelos representacionales de una composición ∇ expresada por construcciones linguísticas de un lenguaje A_j en los planos n a n+m de la figura 19 deberá ser modelo de un término τ_i del sistema formal.
- Cada construcción $O_i \leftarrow P_i$ y la construcción $O_s \leftarrow P_s$ en el paso Sk entre los planos n-1 y n de la figura 19 será modelo de una fórmula $\leftarrow(\tau_i, t_i)$ del sistema formal.

. Los teoremas de la descomposición y sus corolarios serán interpretacions de las reglas de inferencia del sistema formal.

Obsérvese además que, si denominamos α^* y β^* a la composición multiplicativa de pasos α y β respectivamente, las composiciones multiplicativas seran modelo del sistema formal.

Analicemos ahora las reglas de inferencia del sistema formal que en conjunto con el hecho de que las traducciones α y β deben ser homomórficas justifican esta última aserción, las que dada la informalidad de esta exposición será mejor tratar en el metalenguaje.

Tendremos dos lemas a saber:

$$Si\ P^{\Lambda_{i+1}} = \alpha(P^{\Lambda_i})\ \ y\ \sigma^{\Lambda_{i+2}} \leftarrow P^{\Lambda_{i+1}}\ \ entonces\ \sigma^{\Lambda_{i+2}} \leftarrow P^{\Lambda_i}$$

y

$$Si\ \sigma^{\Lambda_i} \leftarrow P^{\Lambda_{i-1}}\ \ y\ \sigma^{\Lambda_{i+1}} = \beta(\sigma^{\Lambda_i})\ entonces\ \sigma^{\Lambda_{i+1}} \leftarrow P^{\Lambda_{i-1}}$$

El primero de estos lemas nos dice que si un modelo representacional de P en el lenguaje Λ_{i+1} es el resultado de una traducción de un modelo representacional de P en el lenguaje Λ_i y si las construcciones linguísticas del lenguaje Λ_{i+2} son modelo de los términos τ_j del sistema formal y se da que una solución $\sigma^{\Lambda_{i+2}}$ resuelve al problema $P^{\Lambda_{i+1}}$ entonces se puede afirmar que $\sigma^{\Lambda_{i+2}}$ resulve a P^{Λ_i}.

Nótese que, tanto en este lema como en el siguiente y en general en todo el metamodelo, el hecho de que P^{Λ_j} sea un modelo representacional del problema P en el lenguaje Λ_j significa que P^{Λ_j} será una expresión Δ compuesta por sumas y productos de subproblemas de P y que esta expresión de descomposición será la interpretación de un término t_j del sistema formal. Lo que estamos tratando de decir es que P representa, en los distintos sistemas linguísticos cuyas construcciones linguísticas sean modelo de términos t_j, cierta expresión Δ , formada por las interpretaciones de los símbolos funcionales + y $\circ$ en dichos sistemas linguísticos, cuyo resultado sea P.

Lo mismo puede decirse de σ^{Λ_k} respecto de las expresiones formadas por las interpretaciones de los símbolos funcionales $+_f$ y $\circ_f$ en los distintos sistemas linguísticos cuyas construciones linguísticas sean modelo de los términos τ_j del sitema formal.

Volviendo a los lemas, el segundo de ellos asegura que si σ^{Λ_i} resuelve a la versión $P^{\Lambda_{i-1}}$ del problema y $\sigma^{\Lambda_{i+1}}$ es una traducción en el sistema linguístico Λ_{i+1} del modelo representacional σ^{Λ_i} de esa solución entonces se puede afirmar que $\sigma^{\Lambda_{i+1}}$ resuelve a la versión $P^{\Lambda_{i-1}}$ del problema.

La aplicación reiterada de ambos lemas produce un resultado mucho más general que denominaremos teorema de la verificación y cuyo enunciado será:

$$Si\ P^{\Lambda i+k}=\alpha^{*}(P^{\Lambda i})\ \ y\ \sigma^{\Lambda i+k+1}\leftarrow P^{\Lambda i+k}\ \ y\ \sigma^{\Lambda i+k+1}=\beta^{*}(\sigma^{\Lambda i+k+1})$$

$$entonces\ \sigma^{\Lambda i+k+1}\leftarrow P^{\Lambda i}$$

¿Qué es lo que asegura este teorema?, sencillamente que si $P^{\Lambda i+k}$ es el resultado de un número cualquiera de traducciones entre distintos sistemas linguísticos de un modelo representacional $P^{\Lambda i}$ en el lenguaje Λi y luego de un paso Sk la solución $\sigma^{\Lambda i+k+1}$ resuelve a $P^{\Lambda i+k}$ y a su vez $\sigma^{\Lambda i+k+1}$ es un modelo representacional en el lenguaje $\Lambda i+k+1$ resultado de un número cualquiera de traducciones entre distintos sistemas linguísticos de dicha solución $\sigma^{\Lambda i+k+1}$, entonces podemos afirmar que $\sigma^{\Lambda i+k+1}$ resuelve a $P^{\Lambda i}$.

Es decir cualquier función, expresada en un lenguaje dado, resultado de la composición mediante construcciones linguísticas que sean modelo de términos τ_i y de un número cualquiera de traducciones homomórficas de la solución obtenida luego de un paso Sk es solución de cualquier expresión de P en un lenguaje dado que por descompocisiones que sean modelo de términos t_i y un número cualquiera de traducciones homomórficas produzca la expresión de P a la que se aplica dicho paso Sk.

Ahora bien, como dijimos en el punto 4, existe antes de poder contar con una primera descripción del problema (aplicación) en un primer sistema linguístico Λ_1, un paso que consta de actividades de interpretación (en el sentido epistemológico), elucidación, etc., que producirá una primera verbalización del problema en un lenguaje informal Λ_0 y un paso de formalización que transformará esa primera verbalización en la descripción de la aplicación en el sistema linguístico Λ_1. A los fines de esta exposición denominaremos γ a esta última tranformación y asumiremos que el problema real es el descripto por la primera verbalización del mismo en el lenguaje informal Λ_0. Diremos entonces que $P^{\Lambda_1}=\gamma\,(P^{\Lambda_0})$.

Ahora bién dado que P^{Λ_0} es una descripción informal del problema no tiene ningún sentido pensar aquí en teoremas como el de la verificación.
¿Cómo hacemos entonces para asegurar que $P^{\Lambda_1}=\gamma\ (P^{\Lambda_0})$? desgraciadamente, dada la informalidad de P^{Λ_0} esta igualdad tiene el mismo status que una hipótesis en ciencias naturales y sólo puede ser testeada como tal.

¿Cuál es el problema de las hipótesis en ciencias naturales?. Para comprenderlo expondremos brevemente una versión simplificada del método hipotético-deductivo.

Supongamos que una teoría científica determinada se basa en una hipótesis H y deseamos saber si ésta es o no correcta. Lo que se hace es agregar a H un conjunto de hipótesis auxiliares $H_1,---,H_n$ que definen una instancia de H; de este conjunto de hipóteis se deduce formalmente

-utilizando la teoría y la lógica subyacente- una consecuencia observacional que puede contrastarse con la realidad, es decir se realiza una predicción o un retrodicción [31].

Si la concecuencia observacional resulta ser falsa estaremos en el caso:

$$(H \wedge H_1 \wedge \ldots\ldots \wedge H_n \rightarrow C) \wedge \neg C$$

aplicando el modus tollens ($p \rightarrow q \wedge \neg q$ *entonces* $\neg p$) tendremos:

$$(H \wedge H_1 \wedge \ldots\ldots \wedge H_n \rightarrow C) \wedge \neg C \text{ } \textit{entonces} \text{ } \neg(H \wedge H_1 \wedge \ldots\ldots \wedge H_n)$$

Operando sobre la consecuencia tendremos:

$$(H \wedge H_1 \wedge \ldots\ldots \wedge H_n \rightarrow C) \wedge \neg C \text{ } \textit{entonces} \text{ } \neg H \vee \neg(H_1 \wedge \ldots\ldots \wedge H_n)$$

Si por alguna razón podemos asegurar la veracidad de las hipótesis auxiliares $(H_1, \ldots\ldots, H_n)$ entonces aplicando modus ponendo tollens a $((\neg a \vee \neg b) \wedge b$ *entonces* $\neg a)$ habremos refutado la hipótesis H.

Si la consecuencia observacional resulta ser verdadera lo único que podremos afirmar es que no hemos refutado a la hipótesis H ya que el simétrico del modus tollens ($p \rightarrow q \wedge q$ *entonces* p) es una falacia conocida como " falacia de afirmación del consecuente".

Volviendo a nuestro caso, decíamos que $P^{\Lambda_1} = \gamma(P^{\Lambda_0})$ tiene el status de hipóteis, ¿como podremos realizar un experimento que refute o nó dicha hipótesis? sencillamente construyendo por restricción un subproblema P'^{Λ_1} de P^{Λ_1} que sepamos "por ser muy sencillo" que es el resultado de la aplicación de la transformación γ a un problema P'^{Λ_0} y testeando si este último es o no un subproblema de P^{Λ_0}. Entonces tendremos:

Hipótesis principal: $P^{\Lambda_1} = \gamma(P^{\Lambda_0})$

Hipótesis auxiliares: P'^{Λ_1} es subproblema de P^{Λ_1}

$$P'^{\Lambda_1} = \gamma(P'^{\Lambda_0})$$

Concecuencia observacional: P'^{Λ_0} es subproblema de P^{Λ_0}

Entonces lo que denominaremos "criterio de aceptabilidad" podrá expresarse como:

$$P^{\Lambda_1} = \gamma(P^{\Lambda_0}) \wedge P'^{\Lambda_1} \text{ es subproblema de } P^{\Lambda_1} \wedge P'^{\Lambda_1} = \gamma(P'^{\Lambda_0})$$
$$\rightarrow P'^{\Lambda_0} \text{ es subproblema de } P^{\Lambda_0}$$

Como en el caso de las ciencias naturales sólo podemos refutar o no la hipótesis en cuestión. Si se cumple que P'^{Λ_0} es subproblema de P^{Λ_0} no habremos refutado a la conjunción que constituye el antecedente y por lo tanto tampoco habremos refutado a la hipótesis principal.

En caso contrario, aplicando el modus tollens tendremos:

$(P^{\Lambda 1}=\gamma(P^{\Lambda 0}) \wedge P'^{\Lambda 1}$ es subproblema de $P^{\Lambda 1} \wedge P'^{\Lambda 1}=\gamma(P'^{\Lambda 0})$

$\rightarrow P'^{\Lambda 0}$ es subproblema de $P^{\Lambda 0})$ *y* $\neg(P'^{\Lambda 0}$ es subproblema de $P^{\Lambda 0})$

entonces: $\neg(P^{\Lambda 1}=\gamma(P^{\Lambda 0}) \wedge P'^{\Lambda 1}$ es subproblema de $P^{\Lambda 1} \wedge P'^{\Lambda 1}=\gamma(P'^{\Lambda 0}))$

Operando sobre la conclusión tendremos:

$\neg(P^{\Lambda 1}=\gamma(P^{\Lambda 0})) \vee \neg(P'^{\Lambda 1}$ es subproblema de $P^{\Lambda 1} \wedge P'^{\Lambda 1}=\gamma(P'^{\Lambda 0}))$

Como la conjunción es verdadera ya que el primer conyunto es verdadero por construcción y el segundo " por simplicidad ", entonces tendremos:

$$\neg(P^{\Lambda 1}=\gamma(P^{\Lambda 0}))$$

y habremos refutado a nuestra hipótesis.

Denominaremos a este experimento "test de aceptabilidad" y lo denotaremos mediante el símbolo t.

Escribiremos entonces $P^{\Lambda 1} \div t(\gamma(P^{\Lambda 0}))$ que significa que si t refuta la hipótesis se debe realizar backtracking hasta lograr que no lo haga; cuando t no refuta la hipótesis, la aceptamos por el momento, es decir aceptamos por el momento la expresión $P^{\Lambda 1}=\gamma(P^{\Lambda 0})$ como verdadera.

Cuando éste es el caso, de lo hasta aquí expuesto se puede decir que:

1) $P^{\Lambda i}=\alpha^*(\gamma(P^{\Lambda 0}))$ para todo $1 \leq i \leq n-1$

2) $\sigma^{\Lambda n} \leftarrow \gamma(P^{\Lambda 0}))$ por (1) y el primer lema de la verificación

3) $\sigma^{\Lambda n+j} \leftarrow \gamma(P^{\Lambda 0}))$ por (1) y el teorema de la verificación

El resultado (1) nos autoriza a aplicar el test de aceptabilidad t a cualquier hipótesis del tipo $P^{\Lambda i}=\alpha^*(\gamma(P^{\Lambda 0}))$, es decir $P^{\Lambda i} \div t\alpha^*(\gamma(P^{\Lambda 0}))$ es equivalente a $P^{\Lambda i} \div \alpha^*(t(\gamma(P^{\Lambda 0})))$.

Los resultados (2) y (3) nos autorizan a diseñar un nuevo test de aceptabilidad, a saber:

Hipótesis general: $P^{\Lambda 1}=\gamma(P^{\Lambda 0})$

Hipótesis auxiliares: $\sigma^{\Lambda n+j}=\beta^*(Sk(\alpha^*(\gamma(P^{\Lambda 0}))))$

$\sigma'^{\Lambda n+j}$ es una restricción de $\sigma^{\Lambda n+j}$

$$\sigma'^{\Lambda_{n+j}} = \beta^{*}(Sk(\alpha^{*}(\gamma(P'^{\Lambda_0}))))$$

Consecuencia observacional: P'^{Λ_0} es un subproblema de P^{Λ_0}

Es decir, este experimento consiste en determinar si el problema P'^{Λ_0} satisfecho por una subsolución de una solución de la especificación formal del problema P^{Λ_0} es un subproblema de este último. Si este no es el caso se habrá refutado la hipótesis $P^{\Lambda_1} = \gamma(P^{\Lambda_0})$ ya que el teorema de la verificación asegura que la primera y tercera hipótesis auxiliares son verdaderas siempre que toda α y toda β -y por lo tanto α^{*} y β^{*}- sean homomorfismos y la segunda hipótesis auxiliar es verdadera por construcción.

Este experimento será obviamente un nuevo test de aceptabilidad que denotaremos mediante el símbolo T.

Obsérvese que el teorema de la verificación asegura que validar un programa resultado de la aplicación de las transformaciones formales α, Sk y β a una especificación formal de un problema P en un lenguaje Λ_i tiene el mismo status lógico que validar dicha especificación formal.

Es decir $\sigma^{\Lambda_{n+j}} \div T(\beta^{*}(Sk(\alpha^{*}(\gamma(P^{\Lambda_0})))))$ es equivalente a escribir

$$\sigma^{\Lambda_{n+j}} = \beta^{*}(\sigma^{\Lambda_{n+1}}) \wedge \sigma^{\Lambda_n} \leftarrow P^{\Lambda_{n-1}} \wedge P^{\Lambda_{n-1}} \div \alpha^{*}(t(\gamma(P^{\Lambda_0}))).$$

Ahora bien, si denominamos ξ a un metasímbolo que represente la aplicación del test de aceptabilidad t o al T según el contexto, una o ninguna vez.

La expresión:

$$\sigma^{\Lambda_p} = ((\xi\beta)^{*}.(\xi Sk((\xi\alpha)^{*}(\xi\gamma(P^{\Lambda_0}))^{\Lambda_1})^{\Lambda_n})^{\Lambda_{n+1}})^{\Lambda_p}$$

Representa un metamodelo del proceso de construcción de un programa σ^{Λ_p} en el lenguaje de programación Λ_p para una aplicación P^{Λ_0} informalmente descripta en el lenguaje Λ_0.

En el caso de la metodología de Bauer, esta expresión se instancia en:

$$\sigma^{\Lambda_p} = (\beta(Sk(\xi\alpha(\xi\gamma(P^{\Lambda_0}))^{\Lambda_1})^{\Lambda_2})^{\Lambda_3})^{\Lambda_p}$$

donde: σ^{Λ_p} será el programa derivado escrito en el lenguaje de muy bajo nivel (o nivel de implementación) Λ_p, para una aplicación P^{Λ_0} descripta en el lenguaje informal Λ_0 cuya especificación formal escrita en el lenguaje de primer orden Λ_1 es el resultado del paso de abstracción γ. La traducción α traduce esta especificación formal en otra escrita en el lenguaje

Λ_2 que es el lenguaje de muy alto nivel (nivel de especificación). Sk es la primera transformación de Bauer entre dicha especificación formal y un programa escrito en el lenguaje intermedio de expresiones recursivas de funciones (que es el paso de construcción de una solución). β es la segunda tranformación de Bauer estre dicho programa y un programa escrito en el lenguaje "do-while" Λ_p.

Para poder explicar de esta manera la metodología de Jackson debemos admitir que la expresión presentada del metamodelo está demasiado simplificada, ya que se asume que los modelos representacionales P^{Λ_i} de la aplicación y de su solución σ^{Λ_j} son modelos de los términos del sistema formal que explican las descomposiciones y composiciones. En realidad esto ocurre también al explicar la metodología de Bauer solo que en esta metología se aplica el mismo tipo de transformaciones a los distintos productos de la descomposición.

En el caso de Jackson debemos, antes que nada, descomponer el problema P^{Λ_0} en una suma de subproblemas $P_1^{\Lambda_0} + P_2^{\Lambda_0} + \ldots\ldots + P_n^{\Lambda_0}$. $P_1^{\Lambda_0}$ será el problema que se derivará por la metodología de Jackson y el conjunto $\{P_2^{\Lambda_0}, \ldots\ldots, P_n^{\Lambda_0}\}$, es el conjunto de subproblemas que representen las "hojas" del árbol de Jackson.

Entonces para $P_1^{\Lambda_0}$, la instanciación de la expresión del metamodelo será:

$$\sigma_1^{\Lambda_p} = (\xi\beta'(\xi\beta((\xi\gamma(\sigma_1^{\Lambda_0}))^{\Lambda_1})^{\Lambda_2})^{\Lambda_p}$$

donde Λ_1 es un lenguaje regular donde se representan los tipos primitivos de Jackson extendido con relaciones funcionales y por lo tanto el paso de abstracción $\gamma(\sigma_1^{\Lambda_0})$ tendrá como resultado la expresión regular de la entrada y la salida relacionadas mediante relaciones funcionales. β será la traducción de ese modelo representacional en otro en un lenguaje Λ_2 que será el lenguaje Λ_j sin las relaciones funcionales.

Por último β' será la traducción de este último modelo representacional en un programa abstracto escrito en el lenguaje "do-while" Λ_p.

En el caso de los subproblemas $P_i (i>1)$, la instanciación de la expresión del metamodelo será:

$$\sigma_1^{\Lambda_p} = (\xi\beta(\sigma_1^{\Lambda_0}))^{\Lambda_p}$$

donde β será la traducción del modelo representacional descripto en el lenguaje informal Λ_0 en un programa escrito en el lenguaje "do-while" Λ_p.

El último paso será un paso de composición ∇ de los subprogramas $\sigma_1^{\Lambda_p}$ $(i>1)$ según la

estructura del programa abstracto $\sigma_1^{\Lambda_p}$.

Nótese que, con esta metodología, partimos de σ^{Λ_0} para llegar a σ^{Λ_p} es decir el paso de construcción de la función de Skolem (programas) fue llevado a cabo antes de la aplicación de dicha metodología

En realidad, en la expresión del metamodelo estamos expresando que α, Sk y β son funciones y por lo tanto dicha expresión modeliza una metodología a posteriori de su aplicación ya que al ser α, Sk y β funciones, significa que ya hemos elegido las transformaciones a aplicar, no solo el tipo, si no la transformación precisa.

Para simplificar la discusión supongamos que tenemos un caso con una sola transformación, α, una Sk y una β.

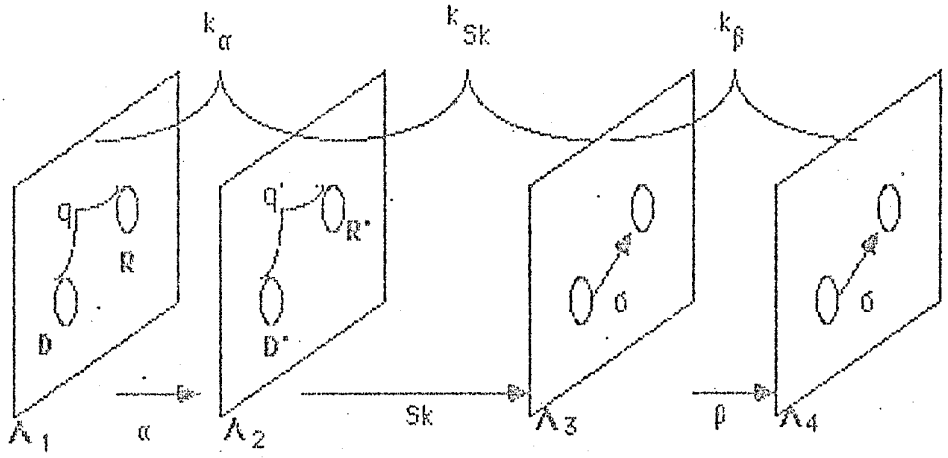

figura 20

Lo que ocurre realmente es que, según ya dijimos, tenemos dos niveles de problemas, uno -que es el representado en los planos perpendiculares al del papel- que es la, aplicación P_q cuya solución es un programa y el otro- representado en el plano del papel- que es el de construir una metodología, que hemos descompuesto en tres subproblemas multiplicativos: encontrar una transformación α, encontrar una transformación Sk y encontrar una transformación β.

Como suponemos que P_q es viable ¿cuál es la condición del primer subproblema de la izquierda en el plano del papel?

Ante todo respetar la semántica de q, lo cual esta asegurado pues como dijimos Λ_1 y Λ_2 deben ser modelos de términos del sistema formal cuya semántica esta dada por la teoría algebraica de problemas.

Luego, completar el proceso de abstracción iniciado por Y. ¿Cómo? modificando los dominios de datos y resultados de P_q a fin de, por ejemplo, plantear un subproblema cuyo dominio de datos sea igual al dominio de las instancias de interés $\bar{I}$, o restringiendo el dominio de

resultados de tal manera de plantear un nuevo problema que sea subproblema completo de P_q, o todo lo contrario extendiéndolos para generalizar el problema P_q y hacer más facil la construcción de una solución en el paso Sk.

Entonces k_α deberá garantizar la conservación de la viabilidad de P_q en el nuevo modelo representacional en el lenguaje A_2. Esto se logra definiendo k_α de manera que obligue a que cualquiera sea el nuevo dominio de datos se cumpla que el dominio de intancias de interés de P_q este contenido en el dominio de datos del nuevo modelo representacional y que el nuevo dominio de resultados no destruya la viabilidad del problema cuyo dominio de datos sea el de ese nuevo modelo representacional.

¿Que ocurre en el segundo subproblema en el plano del papel?. Ya que α asegura el mantenimiento de la viabilidad, el axioma de elección asegura que siempre existirá para $P_q^{A_2}$ una solución, pero como estamos programando pretendemos que éste sea una α-solución, donde la condición α sea "ser una algorithmic skolem function". Exigir de la solución de $P_q^{A_2}$ que cumpla la condición α es, obviamente, restringir el dominio de resultados del subproblema cuya condición es k_{Sk} y por lo tanto dicho subproblema no siempre será viable y entonces no siempre existirá una solución Sk para el mismo.

Esto es lo que ocurre en el método de Bayer con la primera transformación, para hacer que el problema de encontrar una sea viable se debe restringir el dominio de instancias de interés a aquellas especificaciones que sean un modelo de la lógica constructiva.

Respecto del tercer subproblema en el plano del papel, si lo único que se desea es traducir el programa (solución) del lenguaje A_3 al A_4 entonces k_β sólo obligará a conservar la semántica y podremos utilizar un compilador, pero si deseamos que el programa σ^{A_4} se acerque, por ejemplo, a una complejidad dada entonces estamos en el caso en que deseamos tranformar en la traducción una función as (algorithmic skolem function) en una fas (feasible algorithmic skolem function) para un cierto criterio de feasibility.

Esto es equivalente a restringir el dominio de resultados del subproblema cuya condición es k_β y nos arriesgamos a perder la viabilidad del mismo.

Por último, si agregamos descomposición y composición en los planos linguísticos, no basta con el hecho de que las construcciones linguísticas de los distintos lenguajes A_i involucradas en dichas descomposiciones y composiciones sean interpretaciones de los símbolos funcionales $+$ y $*$ o $+_f$ y $*_f$ del sistema formal.

Deberá ocurrir que, además de que la subexpresión del metamodelo:

$$\sigma^{A_p}=(\beta^* (Sk(\alpha^*(P^{A_1}))^{A_n})^{A_{n+1}})^{A_p}$$

sea modelo del teorema de la verificación, también satisfaga al teorema de la descomposición expresado en el sistema formal.

Volvamos ahora, a la observación de Turski ([6] y [7]) sobre el hecho de que en cualquier paso se puede aplicar creatividad al diseño del lenguaje destino o al de la tranformación, creemos haber demostrado que dicha creatividad está restringida por la semántica de la teoría algebrica de problemas, a saber:

- En lo que respecta al diseño del lenguaje destino, la restricción reside en que sus construcciones linguísticas de descomposición y composición, deben ser interpretaciones de los símbolos funcionales $+$ y $\otimes$ o $+_f$ y $\otimes_f$ del sistema formal cuya semántica standard esta dada por la teoría algebraica de problemas.

- En lo que respecta al diseño de la transformación, la restricción reside en que dichas tranformaciones deben ser soluciones (en el sentido de la teoría algebraica de problemas) de problemas cuyas condiciones sean del tipo k_α o k_{SK} k_β.

Para terminar, durante el "Software Process Workshop" llevado a cabo en Runnymede (Gran Bretaña) del 6 al 8 de Febrero de 1984 (referencia [6]) en la discusión general sobre "Specifying and Systems Analysis" Barry Boehm planteó la pregunta "...What is at the top of the two legs (of the inverted Y of Lehmann)...". Creeemos haber demostrado que entre ambas piernas se sitúa el paso crucial del proceso de desarrollo de software, el paso de construcción de funciones de Skolem (programas) para las condiciones (especificaciones) de los subproblemas y problemas abstractos resultado de las descomposiciones Δ.

REFERENCIAS-

[1] Haeberer A., M., Baum G., Veloso P., A., S., "Sobre uma Teoria Algebraica de Problemas", Anais do IV Encontro de Trabalho do Projeto ETHOS, Petrópolis, Brasil, 14 al 16 de Abril de 1987.

[2] Lehman M., M., "The Role of Executable Metric Models in the Programming Process", Proc. IEEE Workshop on Rapid Prototyping, Columbia, Maryland, Apr. 1982. Publ. Software Eng. Notes, Vol. 7, Nº 5, Dec. 1982, pp. 106-111.

[3] Lehman M., M., "Program Evolution", ICST Res. Rep. 82/1.

[4] Lehman M., M., Stenning V. and Turski W., M., "Another Look at Software Desing Methodology", ICST DoC Res. Rep., 83/13, London SW7 2BZ, Jun 1983, 25 p.

[5] Lehman M., M., "Program Evolution, Programming Process, Programming Support", Sonderdruck Attached to Programmierungmgebungen und Compier, Bericht 18a, Tagung 1/1984, German Chapter ACM, 2-4 May 1984, Munich, 12 p.

[6] Lehman M., M., "A Further Model of Coherent Programming Process", IEEE Proceedings of Software Process Workshop, UK Feb 1984, IEEE Comp. Soc., 1984.

[7] Turski, W., M., "The Design of Large Programms", Software Engineering Entwurf und Spezifikation, Floyd and Kopetz, Teubner Ver. 1981, pp. 94-126.

[8] Meibaum, T., S., E. and Turski, W., M., "On What Exactly is Going On When Software is Developed Step by Step", Proc. 7th ICSE, Orlando, FA. March 1984. Publ. IEEE Comp. Soc., Silver Spring, MA. IEEE cat. no. 84ch2011-5, pp 525-533.

[9] Sintzoff, M., "Exploratory Proposals for a Calculus of Software Development", Rep. 84/2, Unité d' Informatique, Univ. Louvain, 1984.

[10] Sintzoff, M., "Understanding and Expressing Software Construction", ASI F8. Springer, Berlin, 1984.

[11] Sintzoff, M., "Desiderata for a Design Calculus", UCL, T3 Memo, Unité d' Informatique, Univ. Louvain, June 1985.

[12] Sintzoff, M., "Playing With a Recursive Design Calculus", UCL, T3 Memo, Unité d' Informatique, Univ. Louvain, 1986.

[13] Jähnichen, S., Ali Hassein, F. and Weber, M., "Program Development Using a Design Calculus", GMD Forschungsstelle an der Universität Karlsruhe, 1986.

[14] Suppes, P., "Axiomatic Set Theory", D. Van Nostrand Company Inc., New York, 1961.

[15] Polya, G., "How to Solve it: a new Aspect of the Mathematical Method", Princeton Univ. Press, Princeton, 1957.

[16] Polya, G., "Mathematics and Plausible Reasoning", Princeton Univ. Press, Princeton, 1954.

[17] Polya, G., "Mthematical Discovery: on Understanding, Learning and Teaching Problem Solving", Wiley, New York, 1962.

[18] Veloso, P., A., S., "Divide and Conquer via Data Types", Anales de la VII Conferencia Latinoamericana de Informática, Caracas, Venezuela, Jan. 1980, pp. 530-539.

[19] Veloso, P., A., S., "A Problem-Theoretical Equivalent of AC", Spring Meeting of the Association for Symbolic Logic, San Francisco, USA, Jan. 1981.

[20] Veloso, P., A., S., "On a Mathematical Theory of Problems", 7th International Congress of Logic, Methodology and Philosophy of Science, Salzburg, Aust., Jun. 1983.

[21] Veloso, P., A., S., Lopes, M., A., "A Framework for Problem Solving: Theory and Methodology", G. E. Lasker Editor, Applied Systems and Cybernetics, Pergamon Press, Oxford, Engl., 1981.

[22] Veloso, P., A., S., Lopes, M., A., "Problem Solvability via Homomorphism and Analogy", Proc. 10th International Congress on Cybernetics, Namur, 1983.

[23] Veloso, P., A., S., Martins, R., C., B., "On Reductibilities Among General Problems", R. Trappl Editor, Cybernetics and Systems Research 2, North Holland, Amsterdam, 1984, pp. 21-25.

[24] Veloso, P., A., S., Veloso, S., R., M., "Problem Decomposition and Reduction: Applicability, Soundness, Completeness", R. Trappl, J. Klir, F. Pichler Editors, Progress in Cybernetics and Systems Research, Vol. VIII, Hemisphere, Washington, 1981, pp. 199-203.

[25] Veloso, P., A., "Aspetos de uma Teoria Geral de Problemas", Tech. Rep., Dep. Inf., Pontificia Universidad de Rio de Janeiro, Brasil, 1982.

[26] Shoenfield, J., R., "Mathematical Logic", Reading, Mass., Addison-Wesley, 1967.

[27] Burris, S., and Sankappanavar, H., P., "A Course in Universal Algebra", Springer Verlag, 1980.

[28] Bauer F., L. and Wossner H., "Algorithmic Language and Program Development", Springer Verlag, Berlin, 1982.

[29] Jackson M., A., "Principles of Program Design", Academic Press, London, 1980.

[30] Hughes J., W., "A formalization and explication of the Michael Jackson method of program construction", Software Practice and Experience, vol. 9, no. 3, March 1979, pp. 191-202.

[31] Hempel C., "Aspects of Scientific Explanation and Others Esseys in the Philosophy of Science", Free Press, New York, 1965.

[32] Balzer R., Cheatham T., E., Green C., "Software Technology in the 1990's: Using a New Paradigm", IEEE Proceedings of Software Process Workshop, UK Feb 1984, IEEE Comp. Soc., 1984.

[33] Wirth N., "Programming Development by Stepwise Refinement", CACM vol. 14, no. 4, April 1971, pp. 221-227.

[34] Boehm B., W., "Software engineering", IEEE Trnas. Comput., Dec. 1976, pp. 1226-1241.

[35] Enderton E., B., "A Mathematical Introduction to Logic", Academic Press, New York, 1970.

[36] Hamilton A., G., "Logic for Mathematicians", Cambridge University Press, Cambridge, 1978.