SISTEMA DE DEFINICION SEMANTICA DE PRIMITIVAS APL

Por Pedro Roa Medina
Trabajo desarrollado en el Centro Científico UAM-IBM. MADRID.

Introducción

El presente desarrollo surgió ante el proyecto de implementación de un compilador usual APL→S/360, escrito en el mismo lenguaje APL y que trabajaría bajo control de un intérprete APL→S/360.

APL considerado como lenguaje a compilar presenta las características:

a) Morfología y sintaxis elementales.
b) Objetos a manipular de talla variable, lo que implica gestión dinámica de memoria.
c) Gran número de funciones primitivas, agravado por las distintas significaciones de cada una de éstas según el o los tipos de operandos a que afecta la función.

Se ve así que la escritura de un tal compilador en lenguaje de alto nivel (concretamente APL) con gestión dinámica de memoria, si bien resuelve problemas como gestión de almacenamientos en compilación y otros, deja los puntos más ingratos (b y c) para ser trabajados en código objeto. Es en el punto c donde se ha incidido, auxiliándose de los logros sobre lenguajes extensibles, buscando una sistemática que nos facilite el proceso, a la vez que dota al lenguaje, en relación con las versiones actuales, de mayores posibilidades de adaptación al problema.

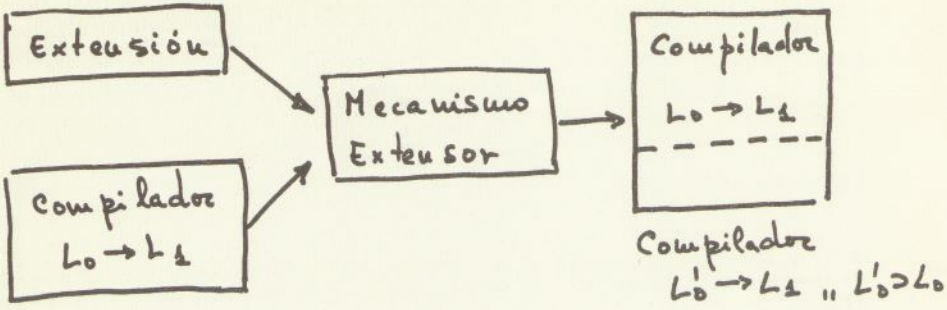
Mecanismo extensor para Definición de Funciones Primitivas tipo APL

Sea L_o un lenguaje definido por una gramática G_o de estructura de frase del tipo

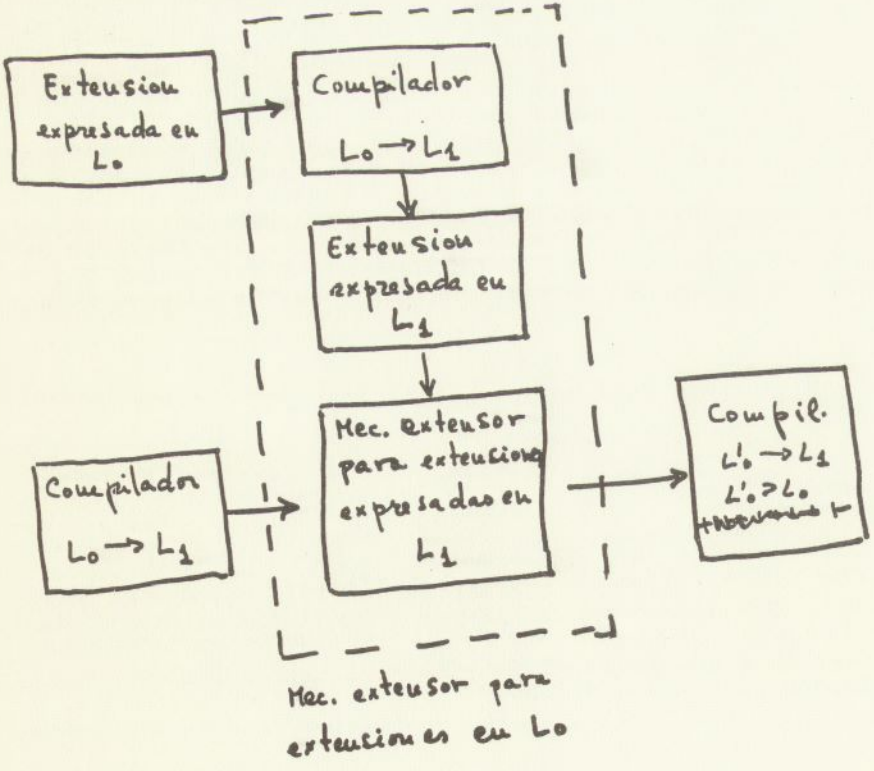
$$\langle frase\rangle ::= \langle arg.\rangle f_d \langle frase\rangle \mid f_m \langle frase\rangle \mid \langle arg.\rangle$$
$$\langle arg.\rangle ::= identificador \mid constante \mid (\langle frase\rangle)$$

donde $f_d \in F_d$, $f_m \in F_m$ siendo F_d y F_m subconjuntos particulares del alfabeto Σ de G_o. A los elementos de F_d los llamaremos símbolos funcionales diádicos y a los de F_m idem monádicos. $F = F_m \cup F_d$ será el conjunto de símbolos funcionales del lenguaje L_o.

Sea $F_o \subsetneq F$,, $F_o \neq \Phi$ el subconjunto de símbolos funcionales con una aplicación S_o de significación incorporada en un traductor $T: L_o \rightarrow L_1$ donde $S_o: F_o \rightarrow L_1$ es una correspondencia entre



Esquema 1.



Esquema 2.

los símbolos de F_o y las cadenas de código objeto asociadas como significado de los símbolos funcionales. La forma usual en la implementación de un compilador para extender S_o al conjunto $F_o \cup \{f\}$,, $f \in F - F_o$ consiste en tomar una cadena apropiada $y \in L_1$ y definir la extensión $S: F_o \cup \{f\} \rightarrow L_1$ como $S|_{F_o} \equiv S_o$, $S(f) = y$.

Otra posibilidad de definición de S(f) puede ser:

- Elegir una cadena x del lenguaje L_o.
- Señalar un identificador t si $f \in F_m$ o dos t_1, t_2 si $f \in F_d$ en la cadena x indicando como x(t) ó $x(t_1, t_2)$ la cadena x de L_o con los identificadores t ó t_1 y t_2 como argumento o valores indeterminados en el proceso de cálculo indicado por x.
- Definir la significación de f como

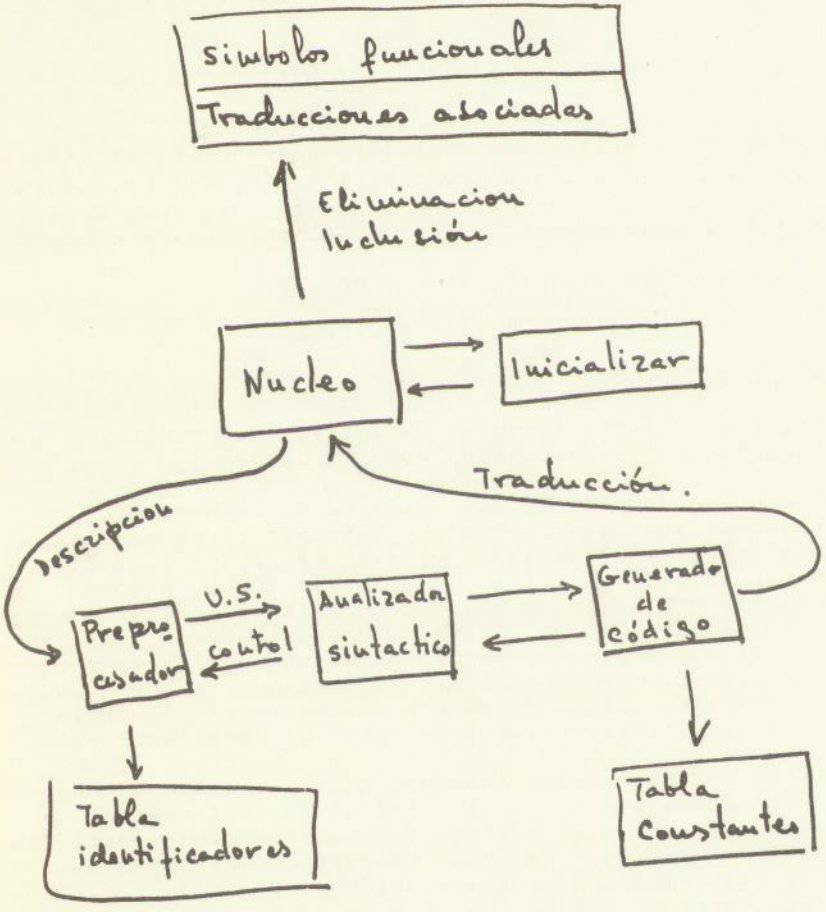
$$S(f) = T(x(t)) \quad \text{ó} \quad S(f) = T(x(t_1, t_2))$$

según $f \in F_m$ ó $f \in F_d$.

Obtenemos así una forma de definición de primitivas que se acerca a la idea de extensión de un lenguaje.

El esquema general de extensión de un lenguaje y por tanto del compilador que lo traduce (esquema 1) se nos condensa en el esquema 2, donde se ve que, al ser el lenguaje de expresión de extensiones igual al lenguaje fuente aceptado por el compilador; la extensión del compilador influye obviamente en la ampliación del lenguaje de expresión de extensiones, dado que es el propio compilador a extender quien realiza la traducción de la extensión. El compilador conseguido tras un proceso de extensión pasa a ocupar en el siguiente proceso las posiciones que ocupaba antes el compilador base.

El esquema definitivo adoptado como mecanismo extensor (esquema 3) ofrece además la posibilidad de supresión de un significado asociado a un símbolo funcional. Trabaja dirigido por un núcleo que ofrece de forma conversacional ambas posibilidades: eliminación de significación y extensión o definición de una nueva primitiva. En el caso de extensión, una definición de función primitiva APL es aceptada, traducida y almacenada, o bien se señala que un error se ha cometido en la descripción. Las eliminaciones de significados de símbolos funcionales se realizan incidiendo directamente sobre la estructura de almacén de traducciones, quedando los símbolos funcionales libres para un redefinición.



Esquema 3. Partes del mecanismo extensor

Conclusión

Situación actual, fines pretendidos y posibles desarrollos.

Este mecanismo extensor ha sido probrado para un lenguaje objeto restringido que no responde a ningún código ensamblador usual. Para que el sistema fuera realmente utilizable requeriría la adaptación a un ensamblador concreto. Revela imprescindible un optimizador de código. No se han previsto problemas de E/S en la fase de definición de funciones.

Se ofrece la posibilidad de conseguir una familia de lenguajes con sintaxis común al APL, pero con diversas semánticas, dirigidas cada una a un problema, partiendo de un compilador con un subconjunto mínimo de primitivas (lenguaje base) el cual sería extendido en distintas direcciones para conseguir lenguajes totalmente identificados con los diversos problemas.

Presentaría toda su potencia este tratamiento en un lenguaje con posibilidad de definición de tipos de objetos a manipular, donde es imprescindible una posibilidad de asociar dinámicamente la significación de las primitivas del lenguaje cuando afectan a un nuevo tipo definido.

Una publicación más detallada, con exposición de toda la implementación y pruebas está hecha en el Centro Científico UAM-IBM de Madrid.