SOL: Un lenguaje para el programador

Rafael O. Fontao, Claudio Delrieux, Guillermo Kalocai, Gustavo Goñi y Gustavo, Ramoscelli
Departamento de Ingeniería Eléctrica
Universidad Nacional del Sur
e-mail: fontao@uns.edu.ar

Resumen

En este trabajo se propone llevar a la práctica, a la luz de nuevas tecnologías informáticas, un ambiente de concepción de programas cuyos orígenes se toman de un trabajo publicado por el director del proyecto (1978) y desarrollado a través de diferentes implementaciones y aplicaciones de la metodología a lo largo de los últimos años.

La idea central consiste en brindar un lenguaje para el programador, que refleje la metodología de concepción de ideas, más que un lenguaje estándar para programar donde finalmente será escrito el problema a resolver.

En este modelo la estructura de control de un programa se define separadamente del resto de las instrucciones del lenguaje de programación, y es modelada a su vez por el comportamiento de una jerarquía de autómatas finitos.

El enunciado inicial de un problema puede concebírselo como un autómata de un solo estado. A cada estado podrá corresponderle hasta dos próximos estados acorde a la evaluación lógica de una condición (si existe) al final de su tarea.

A partir de aquí y en forma recursiva se analiza si un estado particular de un autómata puede ser sintetizado directamente por el lenguaje de programación disponible. En tal caso la descomposición se detiene, sino el comportamiento del estado particular se descompone en un nuevo autómata en un nivel inferior. Y así sucesivamente en un número finito de pasos se arribará a que todos los estados del autómata (jerarquía de autómatas) pueden ser sintetizados "razonablemente bien" por instrucciones del lenguaje de programación disponible.

La idea de concebir programas mediante este lenguaje (o metodología) resulta de interés como medio unificado de programación, ya que la descripción de la solución de un problema puede plantearse con cierto grado

de independencia del lenguaje final a utilizar (paradigma imperativo) y por consiguiente puede permitir la reutilización y portabilidad de programas más eficientemente.

Introducción

La enseñanza clásica de programación se realiza mediante la utilización de un lenguaje práctico disponible (Ej. Pascal) a problemas en un contexto académico limitado y aplicando metodologías de programación (típicamente Programación Estructurada) de modo tal de hacer que los programas sean eficientes, claros de interpretar, mantener y por supuesto que hagan o que se espera que hagan.

En este sentido, entre la escritura de las especificaciones y el listado del programa en que finalmente será ejecutado para cumplirlas, tradicionalmente no hay más que el relato de buenas intenciones y comentarios adecuados en el texto del programa a modo de documentación, pero de la forma de cómo el programador fue creando o concibiendo la solución de las especificaciones la mayoría de las veces quedan pocos vestigios. Los pasos intermedios en la concepción de un programa ceden su lugar al listado de la solución final y su historia suele desaparecer por completo.

Un lenguaje para el programador

El lenguaje **SOL** (por las siglas en Inglés de **Structured Oriented Languaje**) trata de llenar este hueco de historiar la concepción de un programa partiendo de la idea básica de que todo comportamiento secuencial puede modelarse por medio de un autómata finito.

La mera observación de cómo se realiza un trabajo, cualquiera sea su naturaleza, puede concebirse como una secuencia de aplicaciones de herramientas de trabajo, seguida de una herramienta de medida que chequea una condición lógica. Eventualmente la secuencia puede ser vacía y la condición lógica una tautología (es decir, puede obviarse si siempre es verdadera). Y a esta secuencia seguida de una condición, la bautizaremos con el nombre de **trabajo elemental**. Nótese que la noción de elemental corresponde al tipo de herramienta usada y no a la tarea en sí. Por ejemplo, si una herramienta es poderosa hacer una tarea con ella puede resultar elemental pero no si se realiza con herramientas más primitivas.

El nexo entre **estado** de un autómata finito y estado de la programación se dá precisamente en la noción de **trabajo elemental**.

El Modelo de descripción de un programa como Autómata finito

Nuestra experiencia docente nos indica que modelar un comportamiento secuencial por medio de autómatas finitos es una noción muy bien aceptada no solo por estudiantes de sistemas digitales sino también por otras disciplinas.

Este hecho está soportado por los trabajos de estudiantes donde autómatas relativamente complejos son diseñados e implementado eficazmente por circuitos digitales.

El comportamiento de un autómata finito (análogamente máquina secuencial) se especifica por medio de una tabla en la cual las filas son los estados (que indican alguna acción a tomar) y las columnas son los estímulos o entradas al sistema. Cada entrada a esta tabla representa el próximo estado que seguirá el autómata dependiendo del estímulo o entrada. Hay un estado inicial y uno o más estados finales

Para explorar las ventajas de modelar comportamiento secuencial mediante la concepción de autómatas finitos, se propone al siguiente esquema:

Sea L un lenguaje de escritura natural para expresar ideas.

i) Hay un estado inicial.

ii) Cada estado se representa por un programa elemental escrito en L. Es decir, una secuencia de aplicaciones de herramientas de trabajo seguida de una aplicación de medida o testeo.

iii) Hay dos entradas (columnas) que denominamos SI y NO (True and False) que representan los valores lógicos de la condición al final de cada programa elemental (por razones de simplicidad y sin perder generalidad, podemos considerar solo una condición lógica).

El estado actual es el que se está ejecutando y el próximo estado será determinado por el valor lógico de la condición al final del programa elemental.

Sintaxis de SOL

La definición del lenguaje SOL por producciones BNF es la siguiente:

FSA quiere decir: **Autómata de Estados Finito**.
LPD reemplaza a **Lenguaje Práctico Disponible**

<program>::= <control structure> <actions description>

<control structure>::=<state refinement>FIN |<state refinement><control structure>

<state refinement>::= REF <depth identifier><FSA definition> END <depth identifier>

<depth identifier>::= null | <identifier>
<identifier>::=<positive integer > | <positive integer> . <identifier>
<FSA definition>::=<state description>|<state description><FSA definition>
<state description>::= <positive integer> DO <task description> THEN <next state list>

<task description>::= es una descripción escrita del propósito de la tarea elemental

<next state list>::= <next state identifier> |<next state identifier> OR <next state list>
<next state identifier>::= <positive integer>| EXIT <positive integer>|STOP

<positive integer>::= <digit> |<digit><positive integer>
<digit>:: = 0 | l | 2 | 3 | 4| 5 | 6 | 7 l 8 l 9

<actions description>::= DEC <data declarations> < actions list >

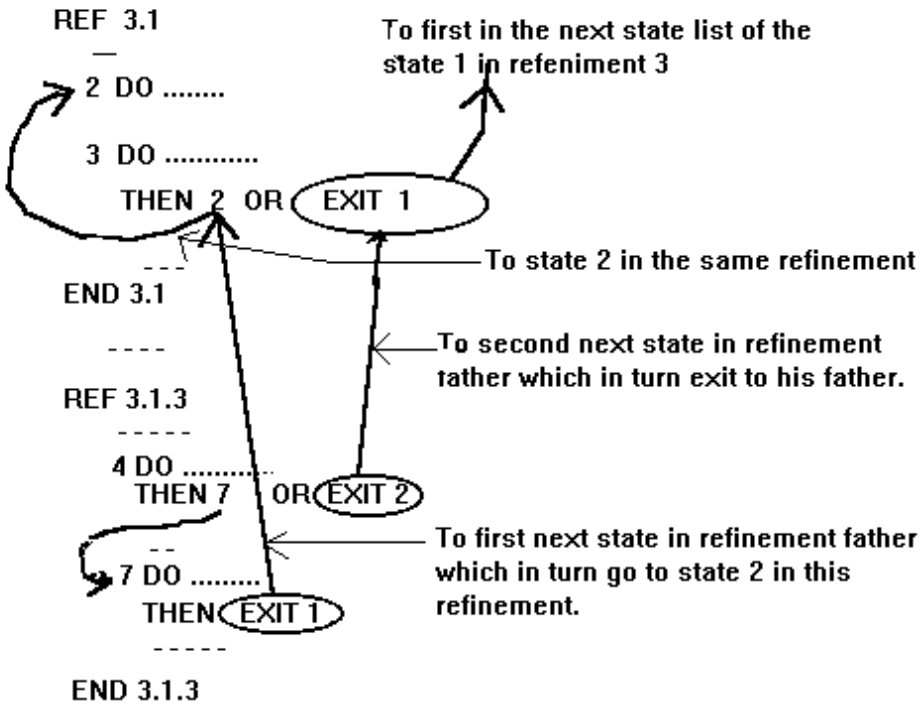
<data declarations>::= etapa de declaraciones (si es que existen) requeridas por el LPD.

<actions list>::= <action> FIN | <action> <action list>

<action>::= ACT <identifier> <output state> <exit condition>
<output state>::= sentencias (en LPD) que sintetizan la salida del estado
<exit condition>::= NEXT | NEXT (<logic condition>)
<logic condition> ::= expresión lógica permitida en el LPD.

Semántica de SOL

Como se establece en sección anterior, la estructura de control se define separadamente del resto de las sentencias que conforman un programa. La estructura de control a su vez, se compone de una sucesión de refinamientos La definición de una FSA se delimita por las palabras REF y END seguidas de un identificador formado por el identificador que refina, seguido de punto y el entero que identifica al estado que está siendo refinado.
Ver la siguiente figura.



El proceso de refinamientos

Los principios de la programación estructurada establecen que un programa debe concebirse en sucesivos refinamientos: En cada paso o refinamiento un módulo o parte del programa se refina explicitando su comportamiento en mayor detalle con las propias instrucciones del lenguaje. El proceso de refinamiento concluye hasta que prácticamente todo el programa esté escrito lo suficientemente claro en el lenguaje de programación

En forma análoga el proceso de concepción de un programa puede hacerse mediante autómatas finitos de la siguiente manera:

Sea LPD un lenguaje práctico disponible donde finalmente será escrito el programa.

Paso 0:
Elija un lenguaje de escritura L para expresar sus ideas y conciba a todo el enunciado del problema como si fuera un FSA de un solo estado y una sola entrada (SI).

Paso 1 a N:
Mientras que exista algún estado que no pueda ser escrito "razonablemente claro" en el LPD, concíbalo expresado en lenguaje L como un nuevo autómata.
Por consiguiente, un estado de un autómata es reemplazado por un nuevo autómata en un nivel inferior. Como regla práctica se aconseja que el nuevo autómata tenga pocos estados. Claro está que al menos deben ser 2 nuevos estados ya que de otro modo no se estaría refinando nada. Solo sería establecer un mero sinónimo o re expresión de lo mismo con otras palabras.

Paso N+1:
A esta altura todos los estados podría expresarse claramente en el LPD y en tal caso el paso último es precisamente escribir el código en LPD para todos los estados.

Características de SOL

La implementación actual se está llevando en DELPHI y el primer pre compilador será destinado a Pascal por ser este leguaje el utilizado en los cursos de programación.

La metodología de concebir mediante SOL permite aventurarnos en áreas como la portabilidad y reutilización de programas ya escritos en lenguajes convencionales y traducirlos, al menos parcialmente, en otros lenguajes disponibles.

Este trabajo de desarrollo e investigación es parte de un PGI de la Universidad Nacional del Sur