Revista Educación en Ingeniería
ISSN 1900-8260
Enero a Junio de 2012, Vol. 7, N°. 13, pp. 55-62 • © 2012 ACOFI • http://www.educacioneningenieria.org
Recibido: 08/03/2012 • Aprobado: 18/05/2012 • Versión final: 25/05/2012

MÁQUINAS ABSTRACTAS DE ESTADOS Y SUS APLICACIONES

ABSTRACT STATE MACHINES AND THEIR APPLICATIONS

Javier Mauricio Reyes Vera
Universidad del Valle, Cali (Colombia) • javier.reyes@correounivalle.edu.co

Resumen

Este artículo, busca dar a conocer de manera sucinta los orígenes de las Máquinas Abstractas de Estado (ASM), además propone una definición formal con base en sus características particulares. Por otra parte, muestra la relación que se puede dar con los *Autómatas de Pila* (AP) resaltando sus aspectos fundamentales. Finalmente, expone una contextualización generalizada de la aplicación de las ASM en las bases de datos, tomando como base tres importantes trabajos recomendados en el sitio web oficial de las ASM.

Palabras clave: álgebras evolucionadas, autómatas finitos, máquinas abstractas de estado, máquinas de estado finito, máquinas de Turing

Abstract

This paper aims to summarize the origin of the Abstract State Machine (ASM) and on the basis of its particular features proposes a formal definition. As well, it exposes the relationships between ASM and Pushdown Automaton (PA) emphasis on the fundamentals. Eventually presents a general application of the ASM in Data Base field, taking into account three remarkable works show in the official ASM web site.

Keywords: evolving algebras, finite automata, abstract state machines, finite state machines, Turing machines

Introducción

Las Máquinas Abstractas de Estado (ASM) surgen a partir de la necesidad de brindar mayores niveles de profundidad y expresividad respecto a la semántica de un programa. La base fundamental de estos planteamientos son las Máquinas de Estado Finito (FSM), que se han desarrollado con la finalidad de formalizar una cantidad de procesos que pueden ser modelados en estas máquinas.

A continuación se presentan algunos conceptos básicos que deben ser considerados para vislumbrar la importancia que revisten las ASM desde sus

orígenes hasta la definición formal y, posteriormente, su aplicabilidad en el área de las bases de datos.

Máquina de estados

Una máquina de estados se enmarca dentro de un modelo de comportamiento de un sistema que tiene entradas y salidas. Las salidas dependen tanto de las entradas actuales, como de las anteriores. Una máquina de estados tiene entonces un conjunto de estados que sirven de intermediarios en esta relación de entradas y salidas, dicha relación hace que el historial de señales de entrada, determine en cada instante un estado de la máquina, de tal manera, que la salida dependa únicamente del estado y las entradas actuales.

De esta manera, una máquina de estados se denomina Máquina de Estados Finitos (FSM, *Finite State Machine*) si el conjunto de estados de la máquina es conocido o finito. En la literatura frecuentemente se tienden a confundir las Máquinas de Estado y las Máquinas de Estado Finito, pues ambas pueden ser modeladas computacionalmente. Varios podrían ser los ejemplos en los cuales se podrían contextualizar las FSM, sin embargo, el más popular es la máquina de Turing, la cual se define como una cinta en la que se lee y también se escribe.

Existen dos tipos de FSM (Lee & Yannakakis, 1996):

- Las máquinas Mearly, que tienen un número finito de estados y producen salidas en las transiciones de estado después de recibir los insumos.
- Las máquinas Moore, en las que las salidas están determinadas únicamente por el estado actual y no dependen directamente de la entrada.

En concreto, una máquina de estados finitos M (Lee & Yannakakis, 1996) se define formalmente como una quíntupla M=(I,O,S, δ, λ)
Donde,

I: Es el conjuntos de símbolos de entrada finitos y no vacíos
O: Es el conjunto de salida
S: Son los estados
$\delta : S \times I \rightarrow S$ Es un estado de la función de transición y
$\lambda : S \times I \rightarrow O$ Es la función de salida

De esta forma, cuando una máquina está en el estado actual ***s*** en ***S*** recibe una entrada a de ***I*** y se mueve al siguiente estado especificado por $\delta(s,a)$ y produce una salida dada por $\lambda(s,a)$.

Evolving algebras

A mediados de los años 90, Guverich (Gurevich & Börger, 1995) propone la definición de la semántica de un programa o lenguaje de programación; en aquel momento, ya existía una definición formal de máquinas de Turing (MT), dicha definición afirmaba que todo era computable siempre y cuando fuese Turing computable, además se afirmaba de manera no formal que "todos los algoritmos pueden ser simulados por una máquina de Turing" (Gurevich & Rosenzweig, 2000).

Con base en esa afirmación, un programa podría ser simulado y además tener un significado preciso a partir de una MT. Sin embargo, para Guverich no era adecuado utilizar una MT para proveer semántica a un programa, ya que éstas se concentran únicamente en bits. El autor observó también que existe una gran brecha entre el nivel de abstracción de un programa y el lenguaje de la MT, lo que llevaría, en palabras del mismo autor, a observar el bosque y no los árboles (Gurevich & Rosenzweig, 2000).

Dado lo anterior, no se puede generalizar una MT para cualquier algoritmo, pues independiente de su nivel de abstracción, no puede ser modelado de manera fiel y acertada dada la diversidad de los algoritmos. Si existieran MT que pudieran modelar cualquier algoritmo, dadas las particularidades propias de éstos, no existiría claridad de cuáles podrían ser sus estados. La experiencia en lógica matemática conlleva a pensar que cualquier clase de realidad matemática estática puede ser representada fielmente como una estructura de primer orden, de esta manera, los estados podrían ser de primer orden considerando un nivel de abstracción, además, que reflejen toda la información pertinente relacionados con un conjunto de instrucciones particulares en todos los casos. Con esta idea, surge un nuevo planteamiento expuesto en Gurevich & Börger (1995).

Teniendo en cuenta que las estructuras de primer orden con vocabulario únicamente funcional, se denominan álgebras, el autor definió esas nuevas máquinas estructuras dinámicas, o álgebras dinámicas, o evolving algebras (algebras evolucionadas). Este último fue el nombre oficial por un buen tiempo,

sin embargo, la comunidad relacionada con evolving algebras le cambió el nombre a Maquinas Abstractas de Estado (ASM por sus siglas en inglés Abtract State Machines), (Gurevich, 2000).

Así, en el planteamiento de Börger & Stärk (2003), las ASM podían ser no determinísticas e interactuar con el ambiente. Posteriormente, en Gurevich, Veanes, & Wallace (2006), le fue agregado ASM paralelas y multiagentes, donde los ASM fueron utilizadas para dar semántica dinámica a varios lenguajes de programación. Las aplicaciones se propagaron en varias direcciones (Börger, 2004). Con el paso del tiempo, se añadió a las ASM la habilidad de simular algoritmos arbitrarios en sus niveles naturales de abstracción sin necesidad de implementarlos, esto hizo que las ASM se convirtieran en elementos apropiados para diseño y análisis de sistemas de alto nivel (Sinha & Patel, 2011).

Un aspecto que se percibe como relevante, es la independencia de la representación de los datos (Blass, Guverich & Brussche, 2002), pues en los modelos de cómputo convencionales se requiere que el grafo sea representado de una forma u otra (por ejemplo matriz de adyacencia), aun cuando el algoritmo sea independiente del grafo de representación. En cambio, usando una ASM (en especial un ASM paralela) se podrían programar esos algoritmos en una representación de manera independiente (Dershowitz, 2010).

Máquina Abstracta de Estados (ASM)

Como ya se afirmó, la tesis central de las ASM gira en torno a que cualquier algoritmo puede ser modelado desde su nivel natural de abstracción por una ASM apropiada. Con base en este planteamiento, los miembros de la comunidad ASM han intentado desarrollar una metodología basada en conceptos matemáticos, que permita que los algoritmos sean modelados de manera natural, es decir, con sus niveles naturales de abstracción. El resultado de este proceso es una metodología para describir maquinas abstractas de estado simples que corresponden a los algoritmos (URL 1).

Concretando, la esencia de una ASM es conectar la brecha entre los modelos formales de computación y los métodos prácticos de especificación formal, ya que modela un sistema en el que un agente cambia el estado actual en pasos discretos. De esta manera, el comportamiento del sistema puede ser visto como una secuencia de estados, en los que cada estado no inicial es determinado por su predecesor en la secuencia. El método de especificación debería definir cual estado es y como un estado es obtenido de su predecesor. Cada estado es presentado por una estructura de primer orden, es decir, un conjunto con relaciones y funciones (lógica de primer orden no estructurada). En la lógica de primer orden la estructura es un conjunto no vacío con operadores y relaciones llamadas operaciones y relaciones básicas de estructura (Patnaik & Gore, 2002). Se podría afirmar entonces que definen un algoritmo mediante una abstracción del estado sobre el que se trabaja y una serie de reglas de transición entre elementos de dicho estado (Gayo et al., 2001).

Definición formal de una ASM

Una Máquina Abstracta de Estados para un lenguaje L, se define como una tupla (A_L, T_L), donde AL es un arreglo de algebra parcial que describen el estado inicial de una máquina abstracta, y TL es el conjunto de reglas de transacción en el cambio de estado de la máquina abstracta (Gottlob, Kappel, & Schrefl, 1991).

La semántica operacional de un lenguaje L se observa así:

El algebra A_L, es usada para definir un programa de código y datos. En A_L el programa es presentado en términos de sus árboles de análisis. El conjunto T_L de las reglas de transición son utilizadas para definir la ejecución de operaciones. Las reglas de transición son aplicadas repetidas veces al árbol de análisis empezando por el nodo raíz hasta que todos los nodos hayan sido evaluados y la ejecución de los programas termine. Cada estado de la máquina abstracta definida por una Evolving Algebra (Algebra Evolucionada) (A_L, T_L) se describe en un ordenamiento de Algebra parcial. El resultado de aplicar algunas reglas de transición a un estado descrito por un algebra AL es otra Algebra A_L', donde el estado inicial es descrito por A_L.

El algebra A_L comprime universos estáticos y dinámicos, y funciones estáticas y dinámicas en universos de productos cartesianos. Las funciones pueden ser totales o parciales. Los universos estáticos tienen un conjunto de elementos fijos, mientras que los

universos dinámicos pueden crecer y disminuirse a medida que las reglas de transformación se aplican.

De igual forma, las funciones estáticas no cambian en el tiempo, en cambio las funciones dinámicas sí. El dominio y los valores de las funciones dinámicas pueden cambiar. De manera general, A_L consiste de universos y funciones que son aplicados a diferentes programas escritos en L y aquellos que son específicos a un programa L en particular.

Las reglas de transición TL describen la ejecución de un programa para especificar las funciones dinámicas que evolucionan. El conjunto T_L es igual para todos los programas escritos en L.

Una regla de transición tiene la forma *if* b *then* $r_1,...,r_n$ con una de las siguientes dos clases:

I. Actualización de funciones: dado f como una función dinámica en A_L con dominio U_1 x...x U_k y rango U_0, donde U_i es el universo. Dado $e_0,...,e_k$ es cerrado (sin variables libres) expresiones sobre $U_0,...,U_k$. entonces $f(e_1,...e_k) := e_0$ es una función actualizada,

Ejemplo:

if
$e_0,...,e_k$ es evaluada a $a_0,...,a_k$

then

a_0 es asignada a $f(a_1,..,a_k)$

II. Extensión de universos: dado U un universo, y $f_1,...f_m$ un listado de funciones actualizadas.

Then let temp = New(U)
in $f_1,...,f_m$
endlet

Es una extensión universo, donde ***temp*** es una variable temporal usada en algunas $f_1,...f_m$. El significado de esta regla de transición es el siguiente. Primero, un nuevo elemento *U* es creado y ***temp*** es nombrada temporalmente.

Segundo, las funciones actualizadas $f_1,...f_m$ son retiradas por turnos. El alcance de ***temp*** es limitado por ***let*** y ***endlet.***

La semántica de transición de reglas es la siguiente, el cuerpo de una regla de transición es llevaba a cabo si y solo si su precondición es cierta. Una regla de transición es llevada a cabo mediante la ejecución de su función de actualización y el universo extendido en secuencia. Se requiere que las precondiciones de todas las reglas de transición seas mutuamente exclusivas.

Las reglas de transición pueden ser anidadas para una notación conveniente. Por ejemplo, la regla de transición

if b then
    if b_1 then r_1
    endif,
    if b_2 then r_2
    endif
endif

Abreviación de las dos reglas:

(1) if b Λ b_1 ***then*** r_1 ***endif***
(2) if b Λ b_2 ***then*** r_2 ***endif***

Las álgebras evolucionadas describen una semántica operacional de un lenguaje. Este enfoque define la semántica de un programa en términos de sus comportamientos de entrada y salida. Dos programas son llamados equivalentes si tienen la misma semántica, si exhiben los mismos comportamientos de entrada y salida (Gottlob et al., 1991).

Como se puede observar, una ASM se define como un pseudocódigo sobre datos abstractos sin ningún requisito teórico particular, donde se podría expresar una forma que garantiza la actualización de las funciones por unas reglas (Flake & Mueller, 2004):

if *Condición* ***then*** *<Actualizaciones>* ***else*** *<Actualizaciones>* ***endif***

Máquinas abstractas de estado y autómatas de pila

Considerando la definición formal de un Autómata de Pila (AP), que es una séxtupla

M = (Q, Σ, Δ, q_0, δ, F) donde,
Q : Es el conjunto finito de estados

Σ : Es el alfabeto de entrada
Δ : Es el alfabeto de pila
q_0 : Es el estado inicial Є Q
F : Es el conjunto de estados finales donde F≠ø y F⊆Q
δ : Es la función definida de la siguiente forma,

$$Q \times (\Sigma \cup \{\lambda\}) \times (\Delta \cup \{\lambda\}) \dashrightarrow_{\delta} P(Q \times \Delta^*)$$

La forma genérica de una terna será, $\delta(q,a,Z) = \{(r1, w1),\ldots,(rk,wk) \mid r1 \in Q , w1 \in \Delta^*\}$

En el caso en que un autómata se encontrara en un estado q, una transición haría por ejemplo que se leyera el símbolo a y la pila de la cinta estuviera en Z, se sustituiría ésta por la w_i y pasaría al estado r_i.

La relación principal entre un AP y un ASM se da cuando el AP, tiene la pila vacía, ya que el símbolo de entrada comienza a llenar la pila y podría afirmarse en términos prácticos que Z es sustituida por el símbolo que entra. Como se observó en la definición formal de un ASM, otra relación es que éste también tiene un conjunto de funciones que son actualizadas de acuerdo con unas condiciones. Si bien las ASM son más robustas en términos de expresividad se podría comparar esta acción en ese paso inicial de los AP.

Teniendo en cuenta que para comparar dos máquinas de estados, la expresividad en la gramática se convierte en un factor crucial, por esa razón, considerar simplemente la funcionalidad de estas máquinas resultaría un criterio muy escueto para proponer una relación estrecha. Como se anotó en el punto anterior del presente artículo, el hecho de que una ASM ofrezca reglas del tipo ***if-then-else-end***, genera una mayor expresividad que cualquier máquina de estados con la que se le compare; incluso con las Máquinas de Turing que fuese en su momento el detonante en los planteamientos de Guverich.

ASM aplicado a bases de datos

Existen diversas aplicaciones de las ASM relacionadas con las Bases de Datos (URL 1), en este documento se han querido enunciar tres importantes trabajos de investigación que ofrecen un panorama robusto en cuanto a su aplicación.

El primer trabajo Fordham, Abiteboul, & Yesha (1997) relacionado con un nuevo paradigma de bases de datos a partir del Algebra evolucionada planteada por Gurevich, la cual se convierte en un referente para maximizar la expresividad de los datos que reposan en una base de datos.

El segundo trabajo Gurevich, Soparkar, & Wallace (1997), ponen en el escenario el problema de recuperación del sistema, haciendo uso de las ASM y sus cualidades, donde se pueden generar mecanismos elegantes, intuitivos y accesibles de recuperación.

El tercero, Prinz & Thalheim (2003) muestra las ASM como una herramienta para la gestión de la semántica operacional de las operaciones, viéndolas como una máquina de estados.

Bases de datos evolucionadas: una aplicación para comercio electrónico

Muchas aplicaciones de las bases de datos complejas y dinámicas como la modelación del producto y el monitoreo de negociaciones requieren un conjunto de características que han sido adoptadas en los modelo semánticos y en las bases de datos como reglas activas, restricciones y herencia entre otras. Desafortunadamente, cada característica ha sido tratada ampliamente pero de forma aislada (Fordham, Abiteboul, & Yesha, 1997).

Además, en una transacción comercial los participantes replantean su estado financiero y nunca aceptarían un sistema del cual desconozcan su comportamiento. El problema se resuelve con un modelo de base de datos rico y extensible (base de datos evolucionada), es decir, equipado con una semántica clara y precisa basada en álgebras evolucionadas.

Este trabajo propone un modelo de base de datos rico y extensible, cuyo principal objetivo es la captura rápida en ambientes cambiantes. Se describe una negociación de comercio electrónico usando el modelo evolucionado de bases de datos, el cual captura el estado de los productos negociados, los negociadores y las leyes que gobiernan cada negociación en particular.

Una base de datos evolucionada se construye usando conjuntos extensibles de características de dominio. Algunas de estas características de domino son de captura genérica como pares atributo/valor, relaciones entre entidades, reglas activas (modificación del precio

cuando la descripción del producto cambia) y restricciones simples (garantizar un margen de beneficio). Otras características de dominio pueden ser más específicas como las relacionadas con autenticación, confidencialidad o descripción de compatibilidad, entre otros componentes. Todas estas características de dominio constituirán adicionalmente metada datos de las bases de datos evolucionadas para la aplicación en particular.

La principal conclusión de esta apuesta, es que las álgebras evolucionadas proveen un apropiado fundamento formal para el modelo de bases de datos propuesto.

Formalización de la recuperación en bases de datos

Este trabajo expone como la metodología de las ASM puede explicar la recuperación y servir de base formal para la definición de un algoritmo de recuperación. El algoritmo propuesto consta de varios niveles de abstracción (Gurevich, Soparkar, & Wallace, 1997).

Las investigaciones han producido algoritmos de alta eficacia pero también muy complejos. Las descripciones de dichos algoritmos son generalmente imprecisas y oscuras, favorecen la inclinación al error, son difíciles de entender y evaluar, y por consiguiente, solo familiares a expertos en el campo. La formalización de la recuperación es un aspecto relativamente nuevo en el estado del arte, pero el nivel de abstracción adoptado en los trabajos realizados, ha hecho a los modelos formales extensos y confusos. Los autores entonces, demuestran el uso de las ASM en el modelamiento y verificación de algoritmos de recuperación. Como es evidenciado en otros trabajos, las ASM pueden modelar un algoritmo en cualquier nivel de abstracción, lo cual es muy importante en este caso particular, pues se ve el problema de recuperación en bases de datos en diferentes modelos de abstracción, iniciando con modelo de alto nivel y sucesivamente refinándolo, haciendo implementación de decisiones en cada paso de refinamiento. Se prueba que el modelo inicial es correcto y que el modelo de cada paso es un refinamiento del modelo obtenido en el paso anterior. El resultado es un desarrollo ordenado y entendible de un algoritmo de recuperación validado.

En conclusión el trabajo muestra un enfoque de recuperación, que en su nivel más alto (nivel inicial) provee de manera clara una vista general del problema de recuperación, y en su siguiente nivel, introduce de forma amigable el detalle de la implementación particular. Los refinamientos metódicos de los subsiguientes modelos indican que un bajo nivel de optimizaciones puede ser agregado incrementalmente.

La semántica operacional de las transacciones

Las transacciones son conceptos de amplio uso en el área de bases de datos. Son comúnmente definidas como: dado un constructor sintáctico en una forma abstracta y declarado un número de propiedades, una máquina debe soportar lo que no es especificado e invisible. Este trabajo se enfoca en proveer una semántica para transacciones. Utilizan las ASM para definir la semántica operacional de las transacciones (Prinz & Thalheim, 2003).

Aunque existen numerosas definiciones de transacciones, aquí se entiende una transacción como una secuencia de operaciones de bases de datos que preserva las propiedades de atomicidad, consistencia, aislamiento y durabilidad.

En conclusión las transacciones son especificadas a nivel lógico como operaciones atómicas las cuales preservan la consistencia de una base de datos. Pueden ser potencialmente ejecutadas en paralelo, si no compiten por recursos o si dicha rivalidad puede ser resuelta. El nivel lógico no considera detalles específicos de opciones de implementación, las cuales dependen del soporte de la computación y del motor de memoria principal.

ASM en la actualidad y propuestas futuras

Desde mediados de la década de los años 90 han aparecido diversos avances en aplicaciones de ASM en el campo de la computación, por ejemplo, en bases de datos, comercio electrónico, arquitectura de *software*, teoría de modelos finitos, teoría de complejidad, sistemas distribuidos entre otros, que dan cuenta de su amplia potencialidad (Börger & Stärkt, 2003).

Sin embargo, se han hecho pocos trabajos de investigación relacionados con la gestión de las interfaces de usuario y usabilidad, en el sentido de formalizar ciertos procesos de interacción de usuarios

heterogéneos en diversas plataformas y el análisis de sus acciones (Razali & Garratt, 2010). De esta manera, se podría plantear la posibilidad de abordar proyectos de investigación aplicada en el campo de investigación en HCI, donde se tenga un conjunto de acciones que sean pensadas como estados que se puedan modelar en las ASM, eso con la finalidad de realizar evaluaciones heurísticas de plataformas desde los principios de la usabilidad, y validar la efectividad de las propuestas de interfaz con respecto a la interacción de un usuario con cualquier plataforma.

Conclusión

El planteamiento de las Máquinas de Estado Finito FSM (ej. las máquinas de Turing), fueron el motor que impulsaron el planteamiento y posterior desarrollo de las Maquinas Abstractas de Estado ASM, dado que permiten modelar procesos que no podían ser modelados por las FSM pues estos dependían del nivel de abstracción de un programa y el lenguaje MT. La tesis de Turing que planteaba que "todos los algoritmos pueden ser simulados por una máquina de Turing" de acuerdo con los planteamientos presentados en este artículo, quedó en cierta manera desvirtuada considerando que habían algoritmos que no podían ser modelados de manera fiel y acertada dada la diversidad de los algoritmos independiente de sus niveles de abstracción.

La ventaja que tienen las ASM frente a las MT y más específicamente las FSM es su riqueza pensada en sus estados y transiciones, esto hace que tenga mayores niveles de expresividad debido a su esquema formal.

Aunque aquí se enuncian ejemplos relacionados con las bases de datos, la aplicación de las ASM puede darse en diversas áreas de la computación (URL 1, 2012) donde estén involucrados una serie de procesos que puedan ser representados a través de estados y transiciones, en consecuencia, las ASM podrían servir como mecanismo de formalización de esos procesos, incluso para su verificación y validación, como es el caso de las herramientas de verificación formal para requerimientos de usabilidad.

Referencias

Blass, A., Guverich, Y., & Brussche, J. (2002). Abstract state machines and computationally complete query langages. *Information and Computation*, *174*(1), 20-36.

Börger, E. (2004). Modeling with abstract state machines: a support for accurate system desing and analysis. *Lecture Notes in Informatics,* 235–239.

Börger Egon, & Stärk Robert. (2003). *Abstract state machines*. *Springer-Verlag, Berlin Heidelberg.*

Dershowitz, N. (2010). Abstract state machines: a generic model of computation, (i), 1-12.

Flake, S., & Mueller, W. (2004). An ASM definition of the dynamic OCL 2 . 0 semantics. *Lecture Notes in Computer Science*, *3273*, 226-240.

Fordham, B., Abiteboul, S., & Yesha, Y. (1997). Evolving databases: an application to electronic commerce. *Proceedings of the 1997 International Database Engineering and Applications Symposium (Cat. No.97TB100166)*, IEEE Computer Society.

Gayo, J. E. L., Lovelle, J. M. C., Díez, M. C. L., Río, C. del, & Joyanes Aguilar, L. (2001). Comparación de técnicas de especificación semántica de lenguajes de programación. *Simposio Iberoamericano de Sistemas de Información e Ingeniería del software en la Sociedad del Conocimiento.*

Gottlob, G., Kappel, G., & Schrefl, M. (1991). Semantics of object-oriented data models - the evolving algebra approach. *Lecture Notes in Computer Science*, *504*, 144-160.

Gurevich, Y. (2000). Sequential abstract state machines capture sequential algorithms. *ACM Transactions on Computational Logic*, *1*(212), 1-32.

Gurevich, Y., & Börger, E. (1995). *Evolving algebras*. Denmark: Department of Computer Science, University of Aarhus.

Gurevich, Y., & Rosenzweig, D. (2000). Partially ordered runs : a case study. *ASM '00 Proceedings of the International Workshop on Abstract State Machines, Theory and Applications.*

Gurevich, Y., Soparkar, N., & Wallace, C. (1997). Formalizing database recovery. *Journal of Universal Computer Science*, *3*(4), 320-340.

Gurevich, Y., Veanes, M., & Wallace, C. (2006). Can abstract state machines be useful in languaje theory? *Theorical Computer Science*. 376(1-2), 17-29.

Lee, D., & Yannakakis, M. (1996). Principles and methods of testing finite state machines-a survey. *IEEE Transactions on Software Engineering*, *84*(8), 1090-1123.

Patnaik, G. K., & Gore, M. M. (2002). Design of compiler for mobile environment and its formalization using evolving algebra. *Proceedings Third International Conference on Mobile Data Management MDM 2002*, IEEE Computer Society.

Prinz, A., & Thalheim, B. (2003). Operational semantics of transactions. *Conferences in Research and Practice in Information Tecnology*, *17*, 169-179.

Razali, R., & Garratt, P. (2010). Usability requirements of formal verification tools: a survey. *Jorunal of Computer Science*, *6*(10), 1189-1198.

Sinha, R., & Patel, H. D. (2011). Abstract state machines as an intermediate representation for high-level synthesis. *Design, Automation & Test in Europe Conference & Exhibition (DATE).*

URL 1 : http://www.eecs.umich.edu/gasm/intro.html, *Recuperado el 25 de Mayo de 2012*. Página oficial de las ASM.

Sobre el autor

Javier Mauricio Reyes Vera
Grupo de Investigación CAMALEON. Magíster (c) en Ingeniería de Sistemas, Universidad del Valle. Profesor del Departamento de Diseño, Universidad del Valle. Cali, Colombia.
javier.reyes@correounivalle.edu.co