San Luis, 20 de diciembre de 2005

Sr. Decano de la Facultad de
Cs. Físico Matemáticas y Naturales
Dr. José Luis Riccardo

El que suscribe, Ing. Salvador Valerio Cavadini, DNI 24.399.352, se dirige a Ud. con motivo de solicitar su inscripción en el Doctorado en Ciencias de la Computación (plan ordenanza rectoral 54/91).

El tema del plan propuesto es "**Utilización del slicing de programas en la validación y modificación de las propiedades de un programa**". A tal fin propone como asesores científicos al Dr. Gilles Barthe del INRIA (Francia) y al Dr. Roberto Uzal de la Universidad Nacional de San Luis.

Se adjunta a la presente el plan de trabajo, CV de los asesores y sus respectivas notas de aceptación de la dirección del trabajo.

Sin otro particular saluda a Ud. muy atte.

Salvador V. Cavadini
Tesista

Justificación de Co- Director

Se considera que es importante que el Dr. Roberto Uzal actúe como co-director de doctorado, ya que es fundamental interactuar fluidamente con un investigador local al Departamento de Informática, teniendo en consideración que el director es externo a la UNSL.

Propuesta de Tesis Doctoral

UTILIZACIÓN DEL SLICING DE PROGRAMAS EN LA VALIDACIÓN Y MODIFICACIÓN DE LAS PROPIEDADES DE UN PROGRAMA

Resumen

La complejidad y tamaño del software implican una limitación para la *verificación formal* de sus propiedades, más aún cuando se trata de propiedades intrínsicamente difíciles de verificar como son, por ejemplo, las propiedades de seguridad y confidencialidad. La presente propuesta tiene por objetivo integrar la tecnología de *slicing de programas* al dominio de la verificación de programas. Creemos que el slicing puede ser utilizado para simplificar las pruebas de las propiedades de los programas a través de la división del mismo en subprogramas menos complejos y más fácilmente manejables en términos de la construcción de una demostración de sus propiedades. Para lograr esto es necesario, primero, estudiar en profundidad cómo utilizar las propiedades de los programas como criterios del proceso de slicing; también es preciso establecer qué propiedades del programa original son preservadas luego del proceso de slicing y estudiar la factibilidad de trasladar al programa original las verificaciones efectuadas sobre sus slices.

Objetivos

Esta propuesta de trabajo está dirigida a determinar cómo combinar las técnicas de verificación –que son potentes y precisas pero requieren gran interacción con el usuario– y las técnicas de *análisis estático de programas* –que tienen un alto grado de automaticidad pero que son menos potentes y precisas– para obtener métodos de verificación precisos y suficientemente automáticos. Más específicamente, se propone estudiar cómo integrar el slicing de programas con técnicas de verificación como la *demostración de teoremas* y la *interpretación abstracta* para verificar las propiedades de programas codificados en lenguaje C.

Esta investigación intentará responder las siguientes preguntas:

- ¿Qué tipo de propiedades pueden ser utilizadas como parámetros del proceso de slicing? ¿Cómo se utilizarán estas propiedades para extraer slices de programas?
- ¿Qué tipos de slices son los más útiles para la tarea de verificación de propiedades?
- ¿Qué propiedades del programa original son aún válidas luego del proceso de slicing?
- ¿Es posible transformar propiedades del programa original en propiedades de sus slices?

El slicing de programas, a menudo, es utilizado para la comprensión de programas aprovechando su capacidad para dividir el programa en partes funcionales menos complejas y manipulables de manera independiente. Es posible un aprovechamiento similar con el objeto de reducir la complejidad de las demostraciones de algunas propiedades. Es decir, ser capaces de obtener un slice de un programa P con respecto de una propiedad Q de tal manera que, si puede definirse una demostración de Q en el slice, entonces la propiedad Q es verdadera en P. Para lograr esto debe contestarse otra pregunta: ¿Las propiedades de un slice pueden trasladarse al programa original?

Dar respuesta a estas preguntas significa un primer paso hacia la integración del slicing de programas a un entorno de verificación completo.

Estado del Arte

Sin dudas, la propiedad más relevante de un programa es que **realiza la tarea para la que fue diseñado**. Desde los principios de la computación se ha intentado desarrollar métodos que permitan constatar que los programas hacen exactamente lo que sus diseñadores pretenden que hagan. Una prueba de esto es la siguiente cita de Alan Turing:

How can one check a routine in the sense of making sure that it is right?

> *In order that the man who checks may not have too difficult a task the programmer should make a number of definite assertions which can be checked individually, and from which correctness of the whole programme easily follows.*[1]

Actualmente, la manera más utilizada de obtener confianza sobre el comportamiento de los programas es someterlos a la realización de distintos tipos de pruebas, esta actividad conforma lo que se conoce como "software testing" [7]. Sin embargo, por más pruebas que se realicen, el testing del software no puede asegurar que un programa se comporta en concordancia con sus requerimientos de diseño. Tal como lo expresara Edsger Dijkstra:

> [...] *program testing can be quite effective for showing the presence of bugs, but is hopelessly inadequate for showing their absense.*[2]
>
> [...] *it does not suffice to design a mechanism of wich we* ***hope*** *that it will meet its requirements, but that we must design it in such a form that we can convince ourselves –and anyone else for that matter– that* ***it will****, indeed, meet its requirements.*[3]

En 1967, Robert Floyd propuso utilizar lo que se denominó el *método de aserciones intermedias* como una manera de estudiar las propiedades de los programas [4]. Floyd destacó la posibilidad de definir la semántica de cada operación de un lenguaje de programación como una *regla lógica* que especifica con exactitud qué aserciones son válidas luego de ejecutada la operación basándose en la información sobre las aserciones que son válidas antes de ejecutarse la operación[4]. Las ideas de Floyd fueron luego ampliadas y perfeccionadas por C.A.R. Hoare en un trabajo en el que presentó un *método axiomático* para demostrar que un programa es correcto e introdujo la idea de "invariante del algoritmo" [5].

Desde su publicación, el trabajo de Hoare ha recibido gran atención por parte de quienes buscan métodos que permitan demostrar la correctitud de los programas. Como

---

[1] A. M. Turing. "Checking a large routine". In Report of a Conference on High Speed Automatic Calculating Machines.

[2] Edsger W. Dijkstra. *A discipline of programming*. Prentice-Hall, New Jersey, 1976. página 20.

[3] ibid. página 216. Las **negritas** son nuestras.

[4] Al mismo tiempo, ideas similares fueron formuladas por Peter Naur quíen denominó "instantáneas generales" (*general snapshots*) a las aserciones [9]. Otro aspecto a acotar es el hecho de que posiblemente, el trabajo de Floyd haya estado basado en los grafos de flujo de control presentados por Herman Goldstine y

prueba del interés sobre el método propuesto por Hoare pueden citarse los innumerables esfuerzos por definir los axiomas que permiten operar sobre las distintas construcciones de los lenguajes de los diferentes paradigmas de programación y, al mismo tiempo el impacto que el método axiomático ha tenido sobre los métodos diseño y verificación de programas, e incluso sobre la manera de definir la semántica de los lenguajes de programación [2]. Incluso, algunos autores dividen temporalmente la investigación sobre la verificación de programas en dos grandes etapas: pre-Hoare y pos-Hoare. [6]

Más adelante, en 1976, Edsger Dijkstra presentó un nuevo método formal para establecer la correctitud de los programas a través de la *transformación de predicados* que permiten derivar las *poscondiciones* que deben cumplir los programas en las *precondiciones* más laxas tales que de iniciarse el programa en esas precondiciones, éste terminará su ejecución en un *estado* que cumple las poscondiciones [3]. Es decir, la idea detrás de este enfoque es, esencialmente, invertir la dirección del método de Floyd y Hoare. Dijkstra utilizó predicados para caracterizar conjuntos de estados del programa e introdujo la idea de que los programas deben interpretarse como *transformadores* de estos predicados de manera tal que estas transformaciones permitan relacionar, de la forma deseada, los estados finales e iniciales del programa. Este enfoque permite obtener algoritmos correctos si se comienza su diseño desde las especificaciones de las salidas deseadas y se trabaja "en reversa" sobre estas.

Sin embargo, después de más de tres décadas de presentadas estas ideas, como ya lo mencionamos al principio de este apartado, el testing de programas es el método más utilizado para intentar minimizar la posibilidad de que el comportamiento de los programas no sea el requerido. Esto –la falta de madurez práctica de los métodos formales para proveer de una prueba de correctitud– es causa de la imposibilidad de utilizar los métodos formales a nivel industrial tal como se usan otros métodos de la ingeniería de software. Podría afirmarse que la situación no ha mejorardo sustancialmente desde 1966 cuando Peter Naur decía:

> *It is a deplorable consequence of the lack of influennce of mathematical thinking on the way in which computer programming is currently being pursued, that the*

John von Neuman. Estos grafos incluyen "nodos de aserciones" que son análogos a las aserciones intermedias de Floyd.

regular use of systematic proof procedures, or even the realization that such proof procedures exist, is unknown to the large majority of programmers.[5]

El principal problema que presenta la incorporación de los métodos formales de verificación es la alta complejidad de las aserciones o pre/poscondiciones que se requieren para representar la semántica de los programas de la actualidad. Esta complejidad de la formulación de las verificaciones impone la necesidad de que la persona encargada de realizarla cuente con conocimientos lógico-matemáticos que escapan al común de los programadores y diseñadores. A esto debe sumársele el hecho de que el tamaño de los programas de hoy en día hace que su verificación formal sea impracticable en términos del tiempo requerido para dicha verificación. Si bien se han hecho avances significativos en el desarrollo de herramientas que tienden a automatizar los procesos de demostración, éstas no alcanzan a prescindir de la (alta) interacción de un usuario altamente capacitado.

Creemos que este problema es una oportunidad para estudiar si las técnicas de análisis estático –que permiten un alto grado de automatización– pueden ayudar a simplificar las demostraciones de las propiedades. En este sentido, es bien sabido que hay problemas que se prestan a ser resueltos a través de un enfoque "divide and conquer". Es aquí, precisamente, en donde estimamos que el slicing de programas [11] –posiblemente en combinación con otras técnicas de análisis estático como la propagación de constantes, eliminación de código muerto [1][8]– puede ser de gran ayuda.

El slicing podría utilizarse, por ejemplo, para: a) dividir el programa de manera tal que las demostraciones para cada una de las partes sean construidas más fácilmente que la demostración del programa en su conjunto; b) aislar aquellas sentencias que afectan el cómputo de una o más variables que sean de especial interés a los efectos de la verificación de correctitud; c) identificar las sentencias que cumplen con una o más propiedades.

Metodología de Trabajo

La realización de este trabajo se organizará en tres etapas. La primera comprenderá la exploración de bibliográfica sobre verificación formal de programas y tecnologías de

[5] Peter Naur. "Proof of Algorithms by General Snapshots".

análisis estático para luego desarrollar un método de slicing que combine el análisis estático de dependencias y la utilización de propiedades como guías del proceso de extracción de los slices. Con la ayuda de un prototipo de slicer que implemente el nuevo método de slicing –cuyo desarrollo formará parte de este trabajo– se buscará dar respuesta a los interrogantes que guían el presente proyecto.

Plan de Trabajo resumido

Año 1

- Completar y revisar bibliografía pertinente.
- Familiarizarse con los métodos de verificación de programas (lógica de Hoare, cálculo de precondiciones, etc.)
- Estudiar en profundidad de distintas técnicas de análisis estático de flujo de datos y de control.
- Estudiar las diferentes variantes de la técnica de slicing de programas –en especial aquellas que permiten condicionar el proceso de slicing– con vistas a la incorporación de propiedades como criterios de slicing.

Año 2

- Definir cómo utilizar propiedades del programa como criterios de slicing.
- Desarrollar un método que permita extraer slices utilizando como criterio las propiedades del programa.
- Colaborar en el desarrollo de una herramienta de slicing de programas C que servirá como plataforma para la realización de pruebas tendientes a establecer la viabilidad y utilidad de combinar los métodos de verificación de programas con el slicing de programas[6].

Año 3

- Estudiar la utilidad de la incorporación de otros análisis estáticos de código –como la propagación de constantes, eliminación de código muerto, etc.– en la herramienta de slicing.

[6] Esta herramienta se desarrollará en el marco de un proyecto de investigación que se lleva adelante en INRIA.

- Estudiar si las propiedades del programa original se mantienen en los slices resultantes del proceso de slicing guiado por propiedades.
- Determinar si es posible –y cómo– inferir propiedades del programa original desde las propiedades de sus slices.

Programa de Estudios Propuesto

Inicialmente se propone el siguiente conjunto de cursos que será ampliado y/o modificado a medida que avance el desarrollo de la tesis y acorde a los requerimientos del aspirante. Los cursos se tomarán de la oferta académica de las instituciones de enseñanza superior de Sophia-Antipolis, escuelas de verano y de otras Universidades con las que los Directores de esta tesis mantienen vínculos.

Primer año:

- *Programming language semantics*, dictado por Yves Bertot (2do año de Maestría)
- *Verification and security*, dictado por Gilles Barthe (2do año de Maestría)

Lugar de Trabajo

Laboratorios del equipo *Environments for Verification and Security of Software* (EVEREST) del INRIA en Sophia-Antipolis y laboratorios del *Centro de Investigación y Desarrollo de Software* (CIDESOFT-UCSE) y el *Instituto de Informática* de la Universidad Católica de Santiago del Estero (II-UCSE).

Recursos Requeridos

- Acceso a Internet.
- Acceso a las fuentes bibliográficas necesarias para la realización de la investigación.
- El resto de material para la presentación de la tesis será aportado por el la Facultad de Matemática Aplicada y el CIDESOFT de la UCSE.

Bibliografía

La bibliografía con la que se iniciará el trabajo de investigación está compuesta por los títulos que se enumeran más abajo. Naturalmente, esta lista se ampliará a medida que avance el desarrollo del trabajo aquí propuesto.

- Hiralal Agrawal. “On slicing programs with jump statements”. In *Proceedings of the ACM SIGPLAN '94 Conference on Programming Language Design and Implementation*, volume 29(6) of ACM SIGPLAN Notices, páginas 302-312, Orlando, FL, Junio 1994.

- Hiralal Agrawal, Richard A. DeMillo, y Eugene H. Spafford. “Dynamic slicing in the presence of unconstrained pointers”. Technical Report SERC-TR-93-P, Software Engineering Research Centre, Julio 1991.

- Alfred V. Aho, Ravi Sethi, Jefrey D. Ullman. *Compilers: Principles*, Techniques, and Tools. Addison-Wesley, 1986.

- Krzystof R Apt. “Ten Years of Hoare's Logic: A Survey—Part I”. *ACM Transactions on Programming Languages and Systems*, 3(4), pp. 431-483, October 1981.

- D. C. Atkinson y W. G. Griswold. “The design of whole-program analysis tools”. In *Proceedings of the 18th International Conference on Software Engineering*, pages 16-27, Berlin, Marzo 1996.

- Thomas Jaudon Ball. *The Use of Control-Flow and Control-Dependence in Software Tools*. PhD thesis, Computer Sciences Department, University of Wisconsin, Madison, 1993.

- Thomas Jaudon Ball y Susan B. Horwitz. “Slicing programs with arbitrary control flow”. In *Proceedings of the First International Workshop on Automated and Algorithmic Debugging*, volume 749 of Lecture Notes in Computer Science, pages 206-222, Heidelberg, Mayo 1993. Springer-Verlag.

- Nick Benton. “Simple relational correctness proofs for static analyses and program transformations”. *ACM SIGPLAN Notices, Proceedings of the 31st ACM SIGPLAN-SIGACT symposium on Principles of programming languages POPL '04*, Volume 39 Issue ,1 , 2004 .

- David Binkley. “Slicing in the presence of parameter aliasing”. In *proceedings of the 1993 Software Engineering Research Forum*, páginas 261-268, Orlando, FL, Noviembre 1993.

- David Binkley y Keith Brian Gallagher. “Program slicing”. In *Advances in Computers*, 1996.

- Andreas Blass; Gurevich Yuri. “The Underlying Logic of Hoare Logic”. *Bulletin of the EATCS*, Vol. 70, pp. 82-111, 2000.

- Rastislav Bodík y Rajiv Gupta. “Partial Dead Code Elimination using Slicing Transformations”. In *Proceedings of the ACM SIGPLAN ‘97: Conference on Programing Languaje Design and Implementation (PLDI)*, Las Vegas, Nevada, Junio 1997.

- D. Chase, M. Wegman y F. Zadeck. “Analysis of Pointers and Structures”. *Proceedings of the ACM SIGPLAN’90 Conference on Programing Languaje Design and Implementation*, páginas 296-309, White Plains, New York, Junio 1992.

- Jong-Deok Choi y Jeanne Ferrante. “Static slicing in the presence of goto statements”. *ACM Transactions on Programming Languages and Systems*, 16(4):1097-1113, Julio 1994.

- Joseph J. Comuzzi y Johnson M. Hart. “Program slicing using weakest preconditions”. In *FME '96: Industrial Benefit and Advances in Formal Methods*, volume 1051 of Lecture Notes in Computer Science, pages 557-575, Oxford, England, Marzo 1996. Springer.

- Edsger W. Dijkstra. *A discipline of programming*. Prentice-Hall, New Jersey, 1976.

- Michael D. Ernst. “Slicing pointers and procedures”. Technical Report MSR-TR-95-23, Microsoft Research, Redmond, WA, Abril 1994.

- Jeanne Ferrante, Karl J. Ottenstein, y Joe D. Warren. “The program dependence graph and its use in optimization”. *ACM Transactions on Programming Languages and Systems*, 9(3):319-349, Julio 1987.

- John Field, G. Ramalingam, y Frank Tip. “Parametric program slicing”. In *Conference Record of the 22nd ACM Symposium on Principles of Programming Languages*, pages 379-392, San Francisco, CA, Enero 1995. ACM Press.

- R. W. Floyd, “Assigning Meanings to Programs”. In *Proceedings of a Symposium in Applied Mathematics*, J. T. Schwartz, Ed., Vol. 19,”Mathematical Aspects of Computer Science”, American Mathematical Society, pp. 19-32, 1967.

- Istvan Forgacs y Tibor Gyimóthy. “An efficient interprocedural slicing method for large programs”. In *Proceedings of SEKE'97*, pages 279-287, Madrid, Spain, 1997.

- Mark Harman y Sebastian Danicic. “A new approach to program slicing”. In *7th International Software Quality Week*, San Francisco, Mayo 1994.

- Mark Harman y Sebastian Danicic. “Amorphous program slicing”. In *Fifth International Workshop on Program Comprehension*, Dearborn, Michigan, USA, Mayo 1997.

- C. A. R. Hoare.“An axiomatic basis for computer programming”. *Communications of the ACM*, v.12 n.10, p.576-580, Oct. 1969

- C. A. R. Hoare.“Programs are predicates”. Mathematical Logic and Programming Languages, Prentice-Hall, London, UK, 1985.

- Jingyue Jiang, Xiling Zhou, y David J. Robson. “Program slicing for C--the problems in implementation”. In *Proc. IEEE Int'l Conf. Software Maintenance*, pages 182-190, 1991.

- Ken Kennedy, John R. Allen. *Optimizing compilers for modern architectures: a dependence-based approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

- Panos E. Livadas y Adam Rosenstein. “Slicing in the presence of pointer variables”. Technical report, Computer and Information Sciences Department, University of Florida, Gainesville, FL 32611, 1994.

- James R. Lyle y David Binkley. “Program slicing in the presence of pointers”. In proceedings of the 1993 Software Engineering Research Forum, páginas 255-260, Orlando, FL, Noviembre 1993.

- Peter Naur. “Proof of Algorithms by General Snapshots”, BIT 6, 1966, pp. 310-316, 1966

- Thomas W. Reps y Wuu Yang. “The semantics of program slicing”. Technical Report TR 777, Computer Sciences Department, University of Wisconsin-Madison, Junio 1988.

- Anthony M. Sloane y Jason Holdsworth. “Beyond traditional program slicing”. Technical Report 95/8, Department of Computer Science, James Cook University, Townsville, QLD, 4811 AUSTRALIA, 1995.

- Gregor Snelting. “Combining slicing and constraint solving for validation of measurement software”. In *Static Analysis; Third International Symposium, SAS'96*, volume 1145 of Lecture Notes in Computer Science, Aachen, Septiembre 1996. Springer Verlag.

- Frank Tip. “A survey of program slicing techniques”. *Journal of Programming Languages*, 3(3):121-189, September 1995.

- G. A. Venkatesh. “The semantic approach to program slicing”. In *Proceedings of the ACM SIGPLAN '91 Conference on Programming Language Design and Implementation*, volume 26(6), pages 107-119, Junio 1991.

- Mark Weiser. *Program slices: formal, psychological, and practical investigations of an automatic program abstraction method*. PhD thesis, University of Michigan, Ann Arbor, 1979.

- Mark Weiser. “Programmers use slices when debugging”. *Communications of the ACM*, 25(7):446-452, 1982.

- Mark Weiser. “Program slicing”. *IEEE Transactions on Software Engineering*, 10(4):352-357, Julio 1984.

Referencias del Estado del Arte

1. Alfred V. Aho, Ravi Sethi, Jefrey D. Ullman. *Compilers: Principles*, Techniques, and Tools. Addison-Wesley, 1986.

2. Krzystof R Apt. “Ten Years of Hoare's Logic: A Survey—Part I”. *ACM Transactions on Programming Languages and Systems*, 3(4), pp. 431-483, October 1981.

3. Edsger W. Dijkstra. *A discipline of programming*. Prentice-Hall, New Jersey, 1976.

4. R. W. Floyd, “Assigning Meanings to Programs”. In *Proceedings of a Symposium in Applied Mathematics*, J. T. Schwartz, Ed., Vol. 19,”Mathematical Aspects of Computer Science”, American Mathematical Society, 1967, pp. 19-32.

5. C. A. R. Hoare."An axiomatic basis for computer programming". *Communications of the ACM*, v.12 n.10, p.576-580, Oct. 1969.

6. C. B. Jones. "The early search for tractable ways of reasoning about programs". *Annals of the History of Computing*, Vol 25, No 2, pp 139-143, 2003.

7. Cem Kaner, Jack Falk, Hung Quoc Nguyen: *Testing Computer Software.* Second Edition, John Wiley and Sons, 1993.

8. Ken Kennedy, John R. Allen. *Optimizing compilers for modern architectures: a dependence-based approach.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

9. Peter Naur. "Proof of Algorithms by General Snapshots", BIT 6, 1966, pp. 310-316, 1966.

10. A. M. Turing. "Checking a large routine". In Report of a Conference on High Speed Automatic Calculating Machines. Pp 67-69, 1949.

11. Mark Weiser. *Program slices: formal, psychological, and practical investigations of an automatic program abstraction method.* PhD thesis, University of Michigan, Ann Arbor, 1979.

Salvador V. Cavadini
Tesista

Dr. Roberto Uzal
Director

Dr. Gilles Barthe
Director