1. Título

Revisión y extensión de un model checker para la verificación automática de sistemas tolerantes a fallas

2. Introducción

Un aspecto fundamental para el buen funcionamiento de los distintos sistemas basados en computadoras (desde pequeños sistemas embebidos a grandes sistemas distribuidos) es el de poder tolerar fallas que puedan producirse ya sea como consecuencia de la ejecución misma del sistema (ej. una división por cero) o de agentes externos al sistema (ej. la pérdida de un mensaje de comunicación). La tolerancia a estas fallas implica que el sistema afectado pueda recuperarse del efecto que ésta produce y continuar con su normal ejecución.

Un sistema tolerante a fallas es un sistema que incrementa su confiabilidad respecto a uno que no lo es. O al menos debería. La incorporación de la tolerancia a falla en un sistema no es algo que se logra por mera redundancia de módulos. Un ejemplo paradigmático de un sistema cuyo módulo de tolerancia a falla no tuvo ningún efecto ocurrió en el primer lanzamiento del Ariane 5, el cual no tuvo un desenlace exitoso [8].

Como cualquier sistema, los sistemas tolerantes a fallas también deben ser verificados. En estos casos, no sólo se debe verificar que el sistema preserva las propiedades de *safety* y *liveness* deseadas, sino también que lo hace aún en presencia de fallas. Se han realizado muchos intentos para la verificación formal de sistemas tolerantes a fallas, pero la mayoría de los casos se basa en la incorporación explícita del modelado de la falla dentro del modelo del sistema de una manera ad-hoc. Esta técnica carece de metodología definida y como tal puede introducir errores en el modelo a verificar.

La incorporación de las fallas y su tratamiento dentro de un modelo formal requiere, entonces, un nuevo marco de especificación, ya sea para modelar el sistema como para especificar los requerimientos.

En este trabajo nos enfocaremos en la verificación automática de sistemas tolerantes a fallas siguiendo el enfoque presentado por el *model checking*. Model checking [5, 2] es una técnica de verificación que, dado el modelo del sistema bajo estudio y la propiedad requerida, permite decidir automáticamente si la propiedad es satisfecha o no.

Un primer enfoque a esta solución fue planteado en [7]. En este trabajo (1) se introdujo una extensión al lenguaje de NuSMV, denominado FALLUTO, de manera tal de permitir expresar la inyección de falla de manera modular y declarativa, y (2) se implementó un *front end* que compilaba esta extensión de NuSMV a NuSMV puro y, de producirse un contraejemplo, lo re interpretaba en el modelo orginal.

Este primer trabajo presenta resultados interesantes pero preliminares. Hay varios aspectos a revisar:

1. el sublenguaje de NuSMV que forma parte de FALLUTO es críptico y no permite identificar claramente las transiciones de cada módulo;

2. más aún, este mismo sublenguaje está bajo consideración para entrar en obsolescencia en cuyo caso dejaría de ser mantenido en el model checker NuSMV;

3. la extensión de FALLUTO correspondiente a la inyección de fallas contiene instrucciones redundantes y ambiguas;

4. si bien el lenguaje de modelado contempla las características necesarias para modelar fallas, la lógica de especificación es una lógica temporal tradicional (LTL).

3. Objetivos

Este proyecto persigue, entonces, los siguientes objetivos:

1. La definición de un lenguaje de modelado de sistemas e inyección de fallas que contemple las falencias de FALLUTO y los inconvenientes antes mencionado.

2. La definición de una lógica de especificación que permita hablar apropiadamente de tolerancia a fallas.

3. La implementación de un *front end* para NuSMV que permita la verificación de propiedades dadas en la nueva lógica en modelos dados en el nuevo lenguaje.

4. Metodología

Para la definición del lenguaje de modelado utilizaremos como punto de partida los conceptos definidos en [1] y [6], que dan un primer paso en la identificación del modelado de fallas separándolo del modelo normativo. El lenguaje debe describir claramente redes de autómatas con lo cual se deberá parecer más al lenguaje de PRISM [9, 10] que a NuSMV (aunque sin considerar los saltos probabilistas). Se busca, en este sentido, una articulación con el lenguaje definido en [3]. También será de gran importancia el trabajo [7] que ya diera un claro primer paso en la definición de un lenguaje para la especificación de fallas.

Para la definición de la lógica de especificación, se estudiará la lógica dCTL [4], definida especialmente para especificar propiedades que expresen tolerancia a fallas. Dado su expresividad, se analizará qué subconjunto es expresable en las lógicas que manipula NuSMV (estas son, CTL, LTL y PSL). En base a este estudio se concluirá cuál es la mejor manera de proceder con la lógica de especificación, teniendo en cuenta el balance entre expresividad y simplicidad del lenguaje respecto de su complejidad de análisis.

Finalmente, se construirá el *front end* para NuSMV que realice la verificación de propiedades en la lógica diseñada sobre modelos descriptos en el lenguaje también diseñado en este contexto. Para ello se utilizará el conocimiento adquirido en el desarrollo del model checker FALLUTO [7], en particular lo relacionado al estudio de fairness de fallas.

5. Plan de trabajo

Las tareas ha realizar a lo largo de este proyecto incluyen:

1. Revisión bibliográfica, en particular:
   a) Estudio del model checker NuSMV.
   b) Estudio del model checker Falluto.
   c) Estudio del lenguaje de modelado de Offbeat [3].
   d) Estudio de la lógica dCTL.
2. Definición del lenguaje de especificación de comportamiento de sistemas tolerantes a fallas
3. Definición de la lógica de especificación de propiedades de tolerancia a fallas.
4. Construcción del *front end* para NuSMV.
5. Documentación del software realizado.
6. Escritura del manuscrito del trabajo especial.

Referencias

[1] Anish Arora and Mohamed G. Gouda. Closure and convergence: A foundation of fault-tolerant computing. *IEEE Trans. Software Eng.*, 19(11):1015–1027, 1993.

[2] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. MIT press, 2008.

[3] Nicolás Bordenabe. Offbeat: Una extensin de PRISM para el análisis de sistemas temporizados tolerantes a fallas. Trabajo de grado, FaMAF, Universidad Nacional de Córdoba, 2011.

[4] Pablo F. Castro, Cecilia Kilmurray, Araceli Acosta, and Nazareno Aguirre. dCTL: A branching time temporal logic for fault-tolerant system verification. In *Procs. of SEFM 2011*, LNCS. Springer, 2011. To appear.

[5] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking*. MIT press, 2000.

[6] Felix C. Gärtner. Specifications for fault-tolerance: A comedy of failures. Technical Report TUD-BS-1998-03, Darmstat University of Technology, Germany, 1998.

[7] Edgardo E. Hames. Falluto: Un model checker para la verificación de sistemas tolerantes a falla. Trabajo de grado, FaMAF, Universidad Nacional de Córdoba, 2009.

[8] J.L. Lions et al. Ariane 501 inquiry board report, 1996. Disponible en la página `http://sunnyday.mit.edu/accidents/Ariane5accidentreport.html`.

[9] D. Parker. *Implementation of Symbolic Model Checking for Probabilistic Systems*. PhD thesis, University of Birmingham, 2002.

[10] Prism home page. http://www.prismmodelchecker.org/.