Herramienta para la experimentación sobre métricas de testeabilidad

Pedro Luis Martínez Moreno
Juan Manuel Nombela Gómez
Daniel Salinas Fernández

PROYECTO DE SISTEMAS INFORMÁTICOS
FACULTAD DE INFORMÁTICA
DEPARTAMENTO DE SIC
UNIVERSIDAD COMPLUTENSE DE MADRID

TRABAJO FIN DE CARRERA EN INGENIERÍA INFORMÁTICA

Curso Académico 2012-2013

Director: Pablo Manuel Rabanal Basalo

Codirector: Ismael Rodríguez Laguna

Autorización

Los alumnos abajo firmantes autorizan a la Universidad Complutense de Madrid a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la propia memoria, como el código, los contenidos audiovisuales incluso si incluyen imágenes de los autores, la documentación y/o el prototipo desarrollado.

En Madrid, a 1 de Septiembre de 2013.

Pedro Luis Martínez Moreno Juan Manuel Nombela Gómez Daniel Salinas Fernández

Dedicatoria

Quizá la dedicatoria más difícil sea aquella que incluye los nombres de las personas que faltan. Como aún cada uno de ellos nombra algo más que los recuerdos, quisiera dedicar la culminación de un arduo camino a cada persona que ha escuchado mi lamento, alegría o reflexión. En especial a las dos personas que han soportado el peso de mi crecimiento de manera inquebrantable. Sin ellos el desarrollo del conocimiento no sería más que una fugaz anécdota.

Pedro.

Quisiera dedicar el proyecto en especial a mis padres, por el esfuerzo continuo que han realizado para que yo pueda disfrutar de esta gran oportunidad de formarme ya no solo profesionalmente sino personalmente. También dedicárselo al resto de personas, ya sean amigos o familiares que han estado a mi lado en los momentos buenos y en los menos buenos, siendo un pilar muy importante en el que sostenerme. No quiero olvidarme de mi novia por su continuo apoyo y comprensión, y por hacer que todo este camino sea más fácil.

Juanma.

La dedicatoria va especialmente dedicada a mis padres, que gracias a ellos he tenido la posibilidad de estudiar esta carrera y siempre, y cuando digo siempre es en cada instante, han estado pendiente de mí. No ha sido fácil estudiar esta carrera, añadiendo el hecho de tener que trasladarme para ir a clase a otra ciudad, pero me han apoyado siempre y tanto ellos como yo hemos conseguido sacarlo. Esto es tanto suyo como mío, lo hemos hecho juntos. Gracias enormes de corazón. También me acuerdo de otras personas que me han echado una mano cuando se ha necesitado para obtener algo.

Daniel.

Agradecimientos

A todos aquellos profesores que, en el ejercicio de su profesión o vocación, han contribuido al desarrollo personal y cultural de nuestras personas. En especial a aquellos que, más allá de un temario o reglamento escrito, han fomentado el interés por el conocimiento, por encima de las ramas preestablecidas.

A los familiares o mecenas que han velado para que el coste de la cultura no fuese una piedra en el camino, sin su soporte nuestro aprendizaje sería una utopía.

Por último, a todos los que hicieron del conocimiento su vida, precursores de la ciencia y la tecnología. Sin su trabajo oscuro y poco recompensado la mayoría de las veces, no disfrutaríamos del nuestro.

Resumen

El proyecto consiste el desarrollo de una herramienta para la experimentación sobre métricas de testeabilidad. Para ello, los sistemas se simulan empleando un LTS (Sistema Etiquetado de Transiciones). Dichas medidas son controlabilidad, observabilidad, complejidad y transparencia. Aunque todas estas características afectan a la testeabilidad de los sistemas, no es fácil saber hasta dónde afecta cada una en la tarea de detectar posibles errores en el sistema. El objetivo es, por tanto, intentar demostrar cuales son las métricas son más susceptibles a la hora de detectar errores. Los LTS son un modelo para describir el comportamiento de cualquier sistema. Se han definido una serie de métricas que se utilizarán para clasificar las máquinas generadas. Posteriormente se tratarán de estudiar los errores de programación mediante LTS. Los posibles errores de programación se simulan mediante máquinas mutantes que se diferencian de la máquina original en un determinado punto, ya sea añadiendo un nuevo estado, eliminando una transición, etc. Dichos mutantes simulan típicos errores de programación como la asignación de un valor incorrecto a una variable, un fallo a la hora de poner una condición a una bifurcación, etc.

Palabras clave

LTS, métricas, testing, controlabilidad, observabilidad, complejidad, transparencia y mutante.

Abstract

The project consists of a laboratory experiment assessed on measures of testability to a labeled transition system, also known as LTS. These measures are controllability, observability, complexity and transparency. In the case of the transparency, it has to perform a union of two systems, the wrapper and the target system. Although these features affect the testability of the systems, it is not easy to know how far it affects everyone in the task of detecting errors in the system. The aim is to try to show which LTS are more susceptible to errors when they are based on the metric taken. The LTS, simulate any type of component, either software or hardware. For example, they could simulate a program, where each state would be a certain point in the program with a certain value for its variables, and the different transitions could simulate paths or bifurcations that can be taken when the program is running based on the entries received. In conclusion, it would attempt to demonstrate the testability that the program has and what are their values, and depending on them get a conclusion of how easy or difficult is to detect program errors in it. Possible programming errors are simulated by machines mutants which differ from the original machine at a given point, either by adding a new state, eliminating transition, etc. Such mutants simulate typical programming errors such as assigning an incorrect value to a variable, a failure of a condition on a fork, etc.

Keywords

LTS, wrapper system, target system, controllability, observability, complexity, transparency and mutant.

Índice

Capítulo 1: Introducción

Con los grandes avances realizados en las tecnologías de computación, los sistemas tienen que desarrollar tareas mucho más complicadas. Sin embargo, también se están convirtiendo en sistemas más seguros. Consecuentemente, el testeo es una parte indispensable dentro del desarrollo de modelos de diseño y también, por supuesto, en la implementación de cualquier modelo asociado. Incluso se ha probado que es una formidable tarea y necesaria para sistemas complejos. Todo esto motiva al estudio de las máquinas finitas de estados para conseguir el correcto funcionamiento de los sistemas y para descubrir aspectos de su comportamiento.

Una máquina finita de estados contiene un número finito de estados y produce una salida al transitar entre estados, al haber recibido unas entradas. Las máquinas finitas de estados son usadas para el modelado de sistemas en diversas áreas, incluyendo circuitos secuenciales, ciertos tipos de programas y, más recientemente, protocolos de comunicación. En un problema de testeo, se tiene una máquina sobre la cual falta algo de información. Sería recomendable que se pudiera deducir esa información proporcionando una secuencia de entradas a la máquina y que se observara una secuencia de salida producida.

Porque tiene una importancia práctica y un interés teórico, el problema de testear las máquinas finitas de estados ha sido estudiado en diferentes áreas y en varios tiempos. Los primeros estudios datan de los años 50. En los 60’s y 70’s se produjeron principalmente por la teoría de autómatas y por el testeo de los circuitos secuenciales. Esta área parece no haberse desarrollado mucho hasta hace unos años, cuando el problema del testeo resurgió y ahora es estudiado de nuevo debido a sus aplicaciones para la prueba de conformidad en protocolos de comunicación. Mientras algunos viejos problemas, que habían estado abiertos durante décadas, se han resuelto recientemente, nuevos conceptos y problemas más difíciles están emergiendo de la aparición de nuevas aplicaciones.

Se revisan los problemas fundamentales en el testeo de máquinas finitas de estados y las técnicas para resolver esos problemas, desarrollando así un progreso en el área desde su inicio hasta el presente y en el estado del arte. En adición se puede hablar de extensiones de una máquina finita de estados y otros tópicos relacionados con el testeo.

Las técnicas formales de testing nos proporcionan de métodos sistemáticos que comprueban la corrección de los sistemas. Estas técnicas permiten a los testadores semiautomatizar el trabajo en algunos de los siguientes pasos: extraer casos de test (desde el código de un programa o incluso desde su especificación, si ésta existe), aplicar casos de test al UIT (Implementación Bajo Test), recoger observaciones, y mostrar un diagnóstico de incorrecciones basado en dichas observaciones. En general,

el número de casos de test que tenemos que aplicar al UIT para garantizar su corrección es infinito, pero el tiempo dedicado al testeo en el desarrollo de un proyecto está limitado. Así, es apropiado seleccionar los casos de test que serán aplicados al UIT, de tal forma que la probabilidad de detectar un fallo sea alta siempre que dichos fallos existan, es un paso clave en cualquier proceso de desarrollo. Además, el programador posee también la capacidad de detección de fallos en los casos de test aplicados después sobre su código: si el programa satisface algunas propiedades deseadas, entonces la tarea de detectar fallos (siempre que éstos existan) debería ser más rápida. Algunas de las características que afectan a la testeabilidad de sistemas han identificadas:

(a) La controlabilidad es el grado en el que será posible controlar el estado del producto bajo test.
(b) La observabilidad es el grado en el que será posible observar las salidas deseadas.
(c) La transparencia mide la capacidad de obtener la salida deseada a través de una determinada entrada.
(d) La complejidad del código.

Aunque todas estas características afectan claramente a la testeabilidad de sistemas, no es fácil saber hasta dónde afecta cada una en la tarea de buscar errores en un sistema. Esto dificulta identificar la correlación entre estas características en el grado de testeabilidad del sistema.

Sobre el papel, desarrollamos un experimento de laboratorio que genera LTS para clasificarlos en función de las diferentes métricas para, posteriormente, generar mutantes de la LTS y decidir qué grado de facilidad o dificultad existe para matar el mutante (simula el descubrimiento de errores en el código). Presentamos un modelo de sistemas generalmente usado en el ámbito de las técnicas formales de testing y definimos formalmente los conceptos introducidos anteriormente (en particular (a), (b), (c), y (d)) para sistemas definidos mediante este modelo. A continuación, generamos automáticamente un conjunto de LTS y medimos las propiedades (a), (b), (c), y (d) para todos ellos. En cada modelo empleamos un método de mutation testing. Automáticamente generamos los mutantes (por ejemplo modelos definidos como el original, pero añadiendo los errores típicos que podría introducir un programador de manera aleatoria), y usamos varias estrategias de testing (camino aleatorio, testing adaptativo) para testear dichos mutantes. Calculamos cuántos de estos mutantes eran detectados al comportarse de manera diferente al sistema original, y usamos esta medida para cuantificar la testeabilidad del sistema original. Finalmente, se estudió la correlación entre la testeabilidad de sistemas y las pautas de testeabilidad presentadas anteriormente pudiendo asignar un determinado peso a cada pauta de testeabilidad,

es decir, qué medida puede ser más importante a la hora de crear sistemas fácilmente testeables.

1.1 Trabajo relacionado

A continuación se van a detallar una serie de herramientas similares a la que estamos desarrollando que resuelven una serie de problemas similares. No se van a desarrollar en su totalidad, sino que se va a comentar que es lo que hacen y a qué problemas se refieren en su desarrollo.

Brinksma01

En el artículo Testing Transition Systems: An Annotated Bibliography, Ed Brinksma and Jan Tretmans escriben que los sistemas de transiciones etiquetadas basados en testeo han conseguido un progreso considerable en los últimos 15 años y que se han desarrollado en un campo donde la teoría de testeo está (lentamente) reduciendo la brecha con la práctica. En particular, escriben que los nuevos algoritmos de generación de pruebas se diseñan para que puedan ser utilizados en situaciones reales.
En este artículo se presenta una bibliografía de sistemas de transiciones etiquetadas basadas en teoría de testeo y sus aplicaciones.

Tretmans96

En el artículo Conformance Testing with Labelled Transition Systems: Implementation Relations and Test Generation de Jan Tretmans se estudian los sistemas de transiciones etiquetadas basados en testeo presentando dos algoritmos de generación de test con su correspondiente implementación.
El primer algoritmo asume que la implementación se comunica con su entorno de manera simétrica, mediante interacciones síncronas. Está basado en la teoría de testeo de equivalencia y preorden, como la mayoría de teorías de testeo para sistemas de transiciones etiquetadas, y se encuentra en la literatura con algunas pequeñas variaciones.
El segundo algoritmo se basa en la suposición de que la implementación se comunica con su entorno a través de entradas y salidas. De esta manera, la implementación se formaliza restringiéndola a la clase de sistemas de transiciones etiquetadas, donde los sistemas pueden aceptar siempre entradas. Para esas implementaciones se desarrolla una teoría de testeo, análoga a la teoría de testeo de equivalencia y preorden. Consiste en relaciones de implementación formalizando la idea de conformidad con esas implementaciones, con respecto a las especificaciones del sistema de transiciones etiquetadas, casos de testeo, ejecuciones de testeo, la idea de estar pasando un test y la generación del algoritmo de testeo, el cual está validado para producir el test para una de sus relaciones de implementación.

Poston12

En el artículo A Software Testing Assessment To Manage Project Testability de Robin Poston, Jignya Patel y Jasbir Dhaliwal se escribe sobre la demanda de sistemas de servicio, que en gran parte proviene de “una demanda derivada” y que está influenciada directamente por la manera en que se desarrollaron actividades anteriores. Las primeras etapas del ciclo de vida de proyectos de desarrollo de software estructurado (SDLC) pueden, a menudo, ejecutarse después de lo previsto, reduciendo el tiempo disponible para la realización de las pruebas adecuadas, especialmente cuando los plazos de desarrollo de software tienen que ser cumplidos. Esta situación fomenta la necesidad de influir en actividades de pretest y dirigir el esfuerzo de testeo eficientemente. Las investigaciones realizadas examinan como monitorizar actividades en las primeras etapas de un proyecto en relación a su efecto en la eficiencia y eficacia de la próxima etapa de test del SDLC. Se basa en la perspectiva de "diseño de la testeabilidad" mediante la introducción de la perspectiva de "gestión de la testeabilidad". La testeabilidad del software se centra en si las actividades del SDLC están progresando de manera que permitan al equipo de pruebas encontrar defectos en los productos software si éstos existiesen. Para hacer frente a este reto, se desarrolla un sistema de medición utilizando un software de testeo de evaluación. Esta evaluación está diseñada para proporcionar a los administradores de testeo la información necesaria para: influir en las actividades pretest de forma que se aumente la eficiencia y eficacia del testeo, y utilizar los recursos del plan de testeo para optimizar la eficiencia y la efectividad del mismo.
Se desarrolla software específico para valorar medidas de test mediante entrevistas con informadores clave. Se presentan los datos recogidos de medidas en proyectos de software estructurado a gran escala para ilustrar la utilidad y aplicación de la evaluación.

Lee96

En el artículo Principles And Methods Of Testing Finite State Machines de David Lee y Mihalis Yannakakis se estudian una serie de problemas que surgen dentro del testeo de las máquinas de estados. El primer tipo de problemas aparece cuando se tiene el diagrama de transiciones de estados de una máquina finita pero no se sabe en qué estado se encuentra la ejecución. Se puede aplicar una secuencia a la máquina de manera que, desde el comportamiento de entradas/salidas, se pueda deducir en qué estado se encuentra. Específicamente, en el problema de identificación de estados habría que identificar primeramente el estado inicial de la máquina. Una secuencia de testeo que resuelve este problema es llamada “secuencia distinguidora”. En el problema de verificación de estados hay que verificar en qué estado de la máquina se encuentra en cada momento. La secuencia de testeo que resuelve este problema se llama “secuencia UIO”.
Otro tipo de problemas es el testeo de conformidad. Dada una especificación de una máquina de estados finita que tiene un diagrama de transiciones y una implementación asociada, siendo ésta última una caja negra en la que se pueden observar los comportamientos de las entradas y las salidas, se quiere testear si la implementación concuerda con la especificación. Esto es llamado testeo de

conformidad o problema de detección de falta y una secuencia de test que resuelve este problema se llama “secuencia de chequeo”.

Capítulo 2: Modelo formal

En esta sección introduciremos la idea del modelo de especificación e implementación bajo test (IUT) y nuestra estructura de trabajo. Asumimos que el funcionamiento de las especificaciones pueden ser definidas por medio de sistemas de transiciones etiquetadas (LTS). Un LTS es una máquina con varios estados que cambian en función de algunos eventos internos y externos. El estado puede cambiar porque la máquina recibe una entrada desde su entorno. Además, también asumimos que la máquina puede avanzar silenciosamente hacia otro estado, en respuesta a una acción que no produce ninguna salida, pudiendo ser detectado o no por un observador externo. Esta acción silenciosa será denotada por el símbolo τ. En nuestra aplicación y en sucesivos ejemplos, denotamos el símbolo τ como #.

2.1 Sistema de transiciones etiquetadas

Un sistema de transiciones etiquetadas M es una tupla (S, I, O, s^{in}, T) donde S es un conjunto de estados, I es un conjunto de entradas, O es un conjunto de salidas, $s^{in} \in S$ es el estado inicial, y T es un conjunto de transiciones. Cada transición es una tupla $(s, a, s') \in T$ donde $s, s' \in S$ y $a \in I \cup O \cup \{\tau\}$. Si $a \in I$ entonces la transición se denota como si una entrada a es recibida en el estado s, entonces M cambia su estado a s'. Si $a \in O$ se denota que, cuando M está en un estado s, puede producir una salida a y moverse al estado s’. Si $a = \tau$ entonces se dice que, cuando M está en el estado s, puede silenciosamente moverse al estado s'. Una $transición(s, a, s')$, puede también denotarse como $s \xrightarrow{a} s'$.

De aquí en adelante, la anterior LTS $M = (S, I, O, s^{in}, T)$ será asumida en las próximas definiciones. Algunas ideas relacionadas con la evolución de los LTSs serán presentados en la próxima definición. Un LTS puede evolucionar silenciosamente hacia una secuencia de acciones τ, por lo que es conveniente identificar esos estados que una LTS puede alcanzar silenciosamente desde un estado dado. Con este propósito, definimos el cierre de un estado como el conjunto de estados que una LTS puede alcanzar desde dicho estado, por medio de cierto número de transiciones τ (0 incluido). Una secuencia de entradas (observables) y salidas que pueden ser respuesta de una LTS se llama traza. Denotamos que, si se toman varias transiciones consecutivas, $s \xrightarrow{i} s_1, s_1 \xrightarrow{\tau} s_2, s_2 \xrightarrow{o} s_3, s_3 \xrightarrow{\tau} s_4, s_4 \xrightarrow{i} s'$, entonces la traza producida es $i \cdot o \cdot i$ (τ transiciones no producen ningún comportamiento observable). En la próxima definición, también definimos el conjunto de trazas que una LTS puede

tomar desde un estado dado. Denotamos que las LTS puede tener un comportamiento no determinista: si tenemos las transiciones $s \xrightarrow{o_1} s'$ y $s \xrightarrow{o_2} s''$entonces, desde el estado s, la LTS puede tomar la producción o_1 y moverse a s' o la producción o_2 y moverse a s''. Además las LTSs pueden ser no observables y no deterministas: si tomamos dos transiciones $s \xrightarrow{o} s'$ y $s \xrightarrow{o} s''$, entonces tanto s' como s'' pueden ser alcanzados desde el estado s después de que la LTS emita la salida o. Esto, además, se mantendría si la transición $s \xrightarrow{o} s''$ fuera remplazada por las transiciones $s \xrightarrow{\tau} u$ y $u \xrightarrow{o} s''$ en el ejemplo anterior, ya que la transición τ no es observada. Consecuentemente, una traza dada puede guiar a la LTS a diferentes estados. A continuación, identificamos el conjunto de pares (traza, conjunto de estados) donde la traza es emparejada con el conjunto de estados que puede alcanzar en la LTS: si un par (σ, U) pertenece a este conjunto de pares entonces, realizando la traza σ, la LTS puede (de manera no determinista) alcanzar todos los estados del conjunto U.

2.2 Cierre

Dado un $s \in S$. El cierre de s, denotado por $cierre(s)$ es definido por inducción de la siguiente manera:

- $s \in cierre(s)$;
- Si $s' \in cierre(s)$ y existe la transición $s \xrightarrow{\tau} s'' \in T$ entonces $s'' \in cierre(s)$.

Dados $a_1, \dots, a_n \in I \cup O$ y $s \in S$. Definimos por inducción el conjunto de trazas y destinos en M desde el estado s_0 denotado por $trazasydest(M, s_0)$, como sigue:

- $(\epsilon, cierre(s_0)) \in trazasydest(M, s_0)$;
- Si $(\sigma, U) \in trazasydest(M, s_0)$ y existe $u \xrightarrow{a} v \in T$ con $u \in U$,$a \in I \cup O$ y $v \in S$, entonces $(\sigma', U') \in trazasydest(M, s_0)$, donde $\sigma' = \sigma \cdot a$ y $U' = \cup \left\{ cierre(v') \mid \exists\, u', v' : u' \in U \wedge u' \xrightarrow{a} v' \in T \right\}$.

El conjunto de trazas de M, denotado por $trazas(M, s_0)$, es el conjunto de todos σ tal que existe un U con $(\sigma, U) \in trazasydest(M, s_0)$. Escribimos $s_0 \overset{\sigma}{\Rightarrow} s$ si existe un $(\sigma, U) \in trazasydest(M, s_0)$ con $s \in U$. Asumimos que $trazasydest(M)$ y $trazas(M)$ están en $trazasydest(M, s^{in})$ y $trazas\left(M, s^{in}\right)$, respectivamente.

Las trazas son secuencias de entradas y salidas. Dada una traza, lo próximo que identificamos es su secuencia de entradas y su secuencia de salidas. Por ejemplo, tenemos $i_1, i_4 \in I$, y $o_2, o_3, o_5 \in O$. Entonces dada la traza $\sigma = i_1 \cdot o_2 \cdot o_3 \cdot i_4 \cdot o_5$, la secuencia de entradas para σ es $i_1 \cdot i_4$, mientras que la secuencia de salidas es $o_2 \cdot o_3 \cdot o_5$. En las próximas definiciones, los naturales $f_1, \dots, f_k$ son usados para denotar los índices de las entradas consecutivas de una traza (1 y 4 en el ejemplo

anterior), mientras que $g_1, \dots, g_l$ denotan los índices de las salidas consecutivas (2, 3 y 5).

2.3 Trazas

Sea $\sigma = a_1 \cdot \dots \cdot o_n \in trazas\,(M,\ s_0)$. Se consideran $prefijos(\sigma) = \{\epsilon, a_1,\ a_1 \cdot a_2,\ \dots\ ,\ \sigma\}$. Sea $1 \leq f_1 < \ \dots\ < f_k \leq n$ y $1 \leq g_1 < \ \dots\ < g_k \leq n$ naturales y A = $\{a_{f_1},\ \dots\ ,\ a_{f_k}\}$ y B = $\{a_{g_1},\ \dots\ ,\ a_{g_l}\}$ en el que A ⊆ I, B ⊆ 0 y A ∩ B = ϕ . Se dice que $a_{f_1} \cdot \dots \cdot a_{f_k}$ y $a_{g_1} \cdot\ \dots\ \cdot\ a_{g_l}$ son la secuencia de entrada y la secuencia de salida de σ, denotado respectivamente como $secuencia\_entrada(\sigma)$ y $secuencia\_salida(\sigma)$. Se dice que la longitud de σ, α y β, denotada como $longitud$(α), $longitud(\sigma)$ $y\ longitud(\beta)$ es n, k y l, respectivamente. Se considera $secuencia\_entrada$(M, s_0) = $\{\alpha \mid \exists\ \sigma\ \in trazas(\ M,\ s_0) : entradas(\sigma) =\ \alpha\ \}$.

Típicamente, los testeadores aplican secuencias de entradas a la aplicación en orden para dirigirla (en nuestro modelo abstracto, para dirigirse a un estado concreto). Debido al potencial de las máquinas de estados etiquetadas (que reflejan el potencial no determinista de las especificaciones e implementaciones), una secuencia de entradas puede dirigir la máquina de estados etiquetada a diferentes estados destino. A continuación definimos el conjunto de estos estados.

2.4 Destinos

Dada una secuencia de entradas α y un estado $s_0 \in S$ se define el conjunto de destinos de α desde s_0, denotado como $destinos(M, \alpha, s_0)$ como sigue:

$$\left\{s \mid \exists\ \sigma :\ s_0 \overset{\sigma}{\Rightarrow} s\ \wedge entradas(\sigma) =\ \alpha\right\}$$

Asumimos que los $destinos(M, \alpha)$ son una abreviación de los $destinos(M, \alpha, s^{in})$.

Desafortunadamente, con frecuencia no es posible testear las implementaciones de manera aislada. Por el contrario, la mayoría de las veces la implementación está conectada con otros componentes o unidades del sistema, y estos componentes no podemos eliminarlos del sistema antes de analizar la implementación. Esto conlleva a escenarios en los que los testadores deben interactuar con el sistema en el que la implementación está encuadrada. A continuación, definimos de manera formal cómo las máquinas etiquetadas de estados pueden componerse para formar sistemas (que serán representados por máquinas también). Esencialmente, una máquina etiquetada de estados (LTS) denota la composición de varias LTS como el producto cartesiano de esas máquinas. Los componentes (LTS) dentro de un sistema se comunican con cada uno de los siguientes: las salidas de algunas máquinas (LTS) son entradas de otras LTS. Esto se denota usando la misma nomenclatura para las correspondientes transiciones

de entradas/salidas de ambos componentes. Asumimos que la comunicación interna entre los componentes del sistema, que no puede ser observada desde fuera del mismo, será denotada como transición τ. Las entradas de los componentes del sistema que no son salidas de otros componentes, asumimos que son entradas del sistema, aquellas que el usuario puede utilizar para comunicarse con el mismo. Por otro lado, las salidas de los componentes que no son salidas de otros componentes indican las salidas del sistema; son, por tanto, las salidas al exterior que pueden ser observadas.

Consideramos el caso particular del sistema en el que un componente manifiesta el comportamiento que esperamos conseguir (llamada máquina objetivo en la siguiente definición) y el otro componente simula la capa de comunicación que hay entre el testador y el componente que deseamos testear (llamada máquina envoltorio en la siguiente definición). Sin pérdida de generalidad, asumimos que el sistema sólo se comunica con el mundo exterior a través de las entradas y salidas de la máquina envoltorio.

2.5 Unión de sistemas de transiciones etiquetados

Sea $M_1, \dots, M_n$ sistemas de transiciones etiquetadas para todo $1 \leq i \leq n$, y M_i= $(S_i, I_i, O_i, {s_i}^{in}, T_i)$ para todo $1 \leq j < k \leq n$ en el que $I_j \cap I_k = \phi$ y $O_j \cap O_k = \phi$. El sistema resultante de $M_1, \dots, M_n$ denotado como $M_1 \times \dots \times M_n$ es un sistema de transiciones etiquetadas $M = (S, I, O, s^{in}, T)$ donde

$S = S_1 \times \dots \times S_n$,

$I = \cup_{1 \leq i \leq n} I_i \setminus \cup_{1 \leq i \leq n} O_i$,

$O = \cup_{1 \leq i \leq n} O_i \setminus \cup_{1 \leq i \leq n} I_i$,

$s^{in} = ({s_1}^{in}, \dots, {s_n}^{in})$, y

T se define como sigue:

$$T = \left\{ (s_1, \dots, s_j, \dots, s_k, \dots, s_n) \xrightarrow{\tau} (s_1, \dots, {s'}_j, \dots, {s'}_k, \dots, s_n) \;\middle|\; \exists\, s_j \xrightarrow{a} {s'}_j \in T_j, \right.$$
$$\left. s_j \xrightarrow{a} {s'}_k \in T_k : a \neq \tau \right\} U \left\{ (s_1, \dots, s_j, \dots, s_n) \xrightarrow{a} (s_1, \dots, {s'}_j, \dots, s_n) \;\middle|\; \exists\, s_j \right.$$
$$\left. \xrightarrow{a} {s'}_j \in T_j : a \in I \cup O \cup \{\tau\} \right\}$$

Sea M = $M_1 \times M_2 = (S, I, O, s^{in}, T)$ un sistema donde se tiene $M_i = (S_i, I_i, O_i, {s_i}^{in}, T_i)$ para todo $1 \leq i \leq 2$. Se dice que M es un entorno de pruebas de integración con un envoltorio M_1 y un objetivo M_2 si $I = I_1 \setminus O_2$ y $O = O_1 \setminus I_2$.

Asumimos que, tanto la especificación como la implementación, son definidas de manera correcta por las máquinas de estados etiquetados (LTS), además asumimos

también la existencia de un botón de reset en la implementación, ya que es algo común es las metodologías de testing. De manera que, siempre que el testador quiera devolver a su configuración inicial la implementación, pueda hacerlo. Si la implementación es la máquina objetivo es el marco de trabajo integrado de testing (ver la sección anterior), asumimos que el sistema completo puede ser reseteado.

2.6 Controlabilidad

Se va a definir la controlabilidad, que manifiesta la capacidad de los testeadores para dirigir la implementación hacia algún estado destino concreto. Teniendo una LTS, la controlabilidad de un estado s mide nuestra capacidad para alcanzar de manera unívoca y rápida el estado s desde el estado inicial de la máquina. Se define en dos etapas. Primero, identificamos todas las secuencias de entradas que tienen el mínimo número posible de estados destino desde el estado inicial – teniendo en cuenta que el estado s de la máquina objetivo está incluido entre esos destinos. A continuación, cogemos la secuencia más corta en este conjunto. De manera que la controlabilidad del estado s es inversamente proporcional al número mínimo de destinos, multiplicado por la longitud de la secuencia más corta que alcanza ese número de destinos. La controlabilidad de la máquina LTS es la media de la controlabilidad de todos sus estados.

Sea $M = (S, I, O, s^{in}, T)$. Dado un estado $s \in S$, se define el mínimo número de destinos no deterministas para alcanzar s, denotado como $dest\_nondet$(s), como $min\ \{|destinos(M, \propto)|\ \propto \in secuencias\_entrada(M) \wedge\ s\ \in destinos(M, \propto)\}$.

Se define m como $dest\_nondet$(s). Se define la controlabilidad de s, denotada como $controlabilidad(s)$

$$\frac{1}{m * min\ \{n\ |\propto \in secuencias\_entrada(M) \wedge\ s \in destinos(M,\propto) \wedge |destinos(M,\propto)| = m \wedge\ longitud(\propto) = n\ \}}$$

La controlabilidad de M viene dada por la expresión $\frac{\sum_{s\ \in\ S\setminus\{s^{in}\}}\ controlabilidad(s)}{|S|}$.

Finalmente se ha estimado oportuno definir la controlabilidad de otra manera, para conseguir asignar una mayor responsabilidad (de manera negativa) al no determinismo.

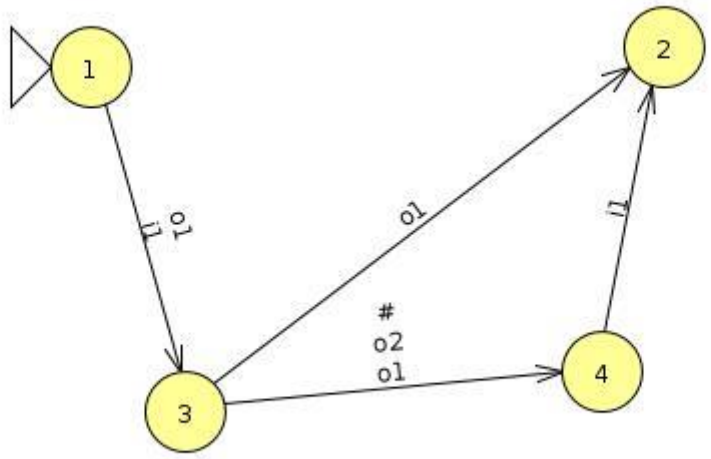
Se calculan todos los caminos que comienzan en el estado inicial de la máquina y terminan en el estado a evaluar. Al final, se escoge el camino con menor número de estados y menor número de indeterminaciones. El número de estados son aquellos por los que pasa el camino. El número de indeterminaciones es la suma del número de indeterminaciones en cada estado por el que pasa el camino. Si un estado sólo puede ir a otro con una entrada o salida, el número de indeterminaciones es cero. Si un estado va a otro con alguna entrada y una salida o transición vacía, entonces el

número de indeterminaciones es uno. Si un estado va a otro con un número n de salidas o transiciones vacías, entonces el número de indeterminaciones es n.

Al final, se tomó la siguiente definición alternativa, la controlabilidad de un estado se define como: controlabilidad(s) = $1/(N^{\propto} * L^{\beta})$

Donde N es el número de indeterminaciones del camino mínimo, L el número de estados del camino. $\propto$ y β son dos constantes que dan mayor peso a N que a L y toman los valores 5 y 2 respectivamente.

A continuación se muestra un ejemplo en el que la máquina tiene como entradas {i1} y como salidas {o1, o2}. Se calcula la controlabilidad para los estados 2, 3 y 4.



Controlabilidad para el estado 2:

Caminos posibles desde el estado inicial:

{1-i1-3-o1-2}

Número de estados: 3

Número de indeterminaciones: 5

{1-o1-3-o1-2}

Número de estados: 3

Número de indeterminaciones: 5

{1-i1-3-#-4-i1-2}

Número de estados: 4

Número de indeterminaciones: 5

{1-o1-3-#-4-i1-2}

Número de estados: 4

Número de indeterminaciones: 5

{1-i1-3-o1-4-i1-2}

Número de estados: 4

Número de indeterminaciones: 5

{1-o1-3-o1-4-i1-2}

Número de estados: 4

Número de indeterminaciones: 5

{1-i1-3-o2-4-i1-2}

Número de estados: 4

Número de indeterminaciones: 5

{1-o1-3-o2-4-i1-2}

Número de estados: 4

Número de indeterminaciones: 5

Todos los caminos tienen el mismo número de indeterminaciones (si hubiese alguno con menos de 5, ese sería el mejor), así que importa el que menor número de estados tenga. Hay dos caminos con el mismo número de estados {1-i1-3-o1-2} y {1-o1-3-o1-2}, ambos son idénticos, así que vale cualquiera de los dos. Lo importante es que con cualquiera de estos dos caminos tenemos N=5, L=3.

Por tanto, para el estado 2, la controlabilidad es controlabilidad(2) = $\frac{1}{5^5+3^2}$ = 0,00031

Controlabilidad para el estado 3:

Caminos posibles desde el estado inicial:

{1-i1-3}

Número de estados: 2

Número de indeterminaciones: 1

{1-o1-3}

Número de estados: 2

Número de indeterminaciones: 1

Todos los caminos tienen el mismo número de indeterminaciones, así que importa el que menor número de estados tenga. Pero dos caminos tienen el mismo número de estados 2, así que vale cualquiera de los dos. Lo importante es que con cualquiera de estos dos caminos tenemos N=1, L=2.

Por tanto, para el estado 3, la controlabilidad es controlabilidad(3) = $\frac{1}{1^5 + 2^2}$ = 0,2

Controlabilidad para el estado 4:

Caminos posibles desde el estado inicial:

{1-i1-3-#-4}

Número de estados: 3

Número de indeterminaciones: 5

{1-o1-3-#-4}

Número de estados: 3

Número de indeterminaciones: 5

{1-i1-3-o1-4}

Número de estados: 3

Número de indeterminaciones: 5

{1-o1-3-o1-4}

Número de estados: 3

Número de indeterminaciones: 5

{1-i1-3-o2-4}

Número de estados: 3

Número de indeterminaciones: 5

{1-o1-3-o2-4}

Número de estados: 3

Número de indeterminaciones: 5

Todos los caminos tienen el mismo número de indeterminaciones (si hubiese alguno con menos de 5, ese sería el mejor), así que importa el que menor número de estados tenga. Todos los caminos tienen el mismo número de estados, así que vale cualquiera de ellos. Lo importante es que con cualquiera de estos caminos tenemos N=5, L=3.

Por tanto, para el estado 4, controlabilidad(4) = $1/5^5+3^2$ = 0,00031

Controlabilidad total de la máquina: $\sum_{i=2}^{n} controlabilidad(M)$ = $\frac{0,00031+0,2+0,00031}{4}$ = 0,05015

2.7 Observabilidad

Se define la observabilidad de una máquina de transiciones etiquetadas $M = (S, I, O, s^{in}, T)$ como la capacidad de detectar el comportamiento observable asociado. La observabilidad debe ser medida en términos de la capacidad de inferir, desde las salidas observables, qué transiciones se están tomando dentro de la máquina, a condición de que se comporte como se había definido en la especificación. En este caso, y por definición, las transiciones τ no pueden ser observadas, ya que ellas producen un comportamiento no observable.

La observabilidad de cada estado de la máquina de transiciones etiquetadas es inversamente proporcional al número de estados que se pueden alcanzar desde el propio estado para cada salida observable. Para obtener este número, tenemos en cuenta que la máquina:

-Podría realizar acciones τ antes de producir la salida.
-Podría tomar varias transiciones que emitan la misma salida.
-Podría realizar otras τ transiciones después de producir la salida.

Por último, la observabilidad de una máquina se mide haciendo la media de la observabilidad de todos los estados.

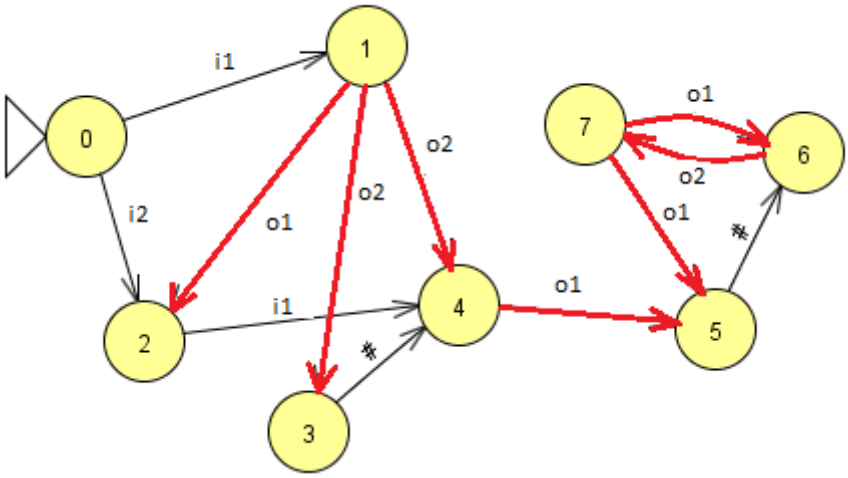
$$observabilidad(s) = \sum_{o \in O} \frac{1}{\left|\left\{s''' \mid \exists\, s', s'', s''' : s' \in cierre(s) \wedge s' \xrightarrow{o} s'' \in T \wedge s''' \in cierre(s)\right\}\right|}$$

Donde $s \in$ S\{s^inicial}

$$observabilidad(inicial) = 1$$

$$observabilidad(M) = \frac{\sum_{s \in S} observabilidad(s)}{|S|}$$

Se muestra un ejemplo explicativo de esta medida, donde las transiciones en rojo denotan las salidas y el resto las transiciones de entrada y τ transiciones, es decir, como entradas se tienen {i1, i2} y como salidas {o1, o2}.



Observabilidad del estado 0: el resultado es 1 del propio estado.

Observabilidad del estado 1: con la salida o1 se obtiene 1 (del estado 2) y con la salida o2 se obtiene 2 (del estado 3 y del estado 4). Como se tienen dos salidas, por tanto el resultado es $(\frac{1}{1}+\frac{1}{2})$ / 2 = 0,75.

Observabilidad del estado 2: el resultado es 1 del propio estado.

Observabilidad del estado 3: con la salida o1 se obtiene 2 (del estado 5 y del estado 6 por la transición vacía) y con la salida o2, 1. Como se tienen dos salidas, por tanto el resultado es $(\frac{1}{2}+\frac{1}{1})$ / 2 = 0,75.

Observabilidad del estado 4: con la salida o1 se obtiene 2 (del estado 5 y del estado 6 por la transición vacía) y con la salida o2, 1. Como se tienen dos salidas, por tanto el resultado es $(\frac{1}{2}+\frac{1}{1})$ / 2 = 0,75.

Observabilidad del estado 5: con la salida o1, 1 y con la salida o2 se obtiene 1 (del estado 7). Como se tienen dos salidas, por tanto el resultado es $(\frac{1}{1}+\frac{1}{1})$ / 2 = 1.

Observabilidad del estado 6: tiene para la salida o1, 1 y para la salida o2 tiene 1 (del estado 7). Como se tienen dos salidas, por tanto el resultado es $(\frac{1}{1}+\frac{1}{1})$ / 2 = 1.

Observabilidad del estado 7: con la salida o1 se obtiene 2 (del estado 5 y del estado 6) y con la salida o2, 1. Como se tienen dos salidas, por tanto el resultado es $(\frac{1}{2}+\frac{1}{1})$ / 2 = 0,75.

La observabilidad total es la media de la observabilidad de todos los estados. Por consiguiente:

Observabilidad = $\frac{1+0{,}75+1+0{,}75+0{,}75+1+1+0{,}75}{8}$ = 0,875.

2.8 Transparencia

Definimos la transparencia como la capacidad de la máquina envoltorio (esto es, el componente abstracto que encapsula la interfaz entre el testador y el componente testeado) de transmitir de manera clara al objeto testeado los estímulos del test y de devolver al testeador las respuestas de la máquina testeada. De este modo, la transparencia hace referencia a la interfaz del sistema que rodea. Nótese que si no existe una máquina envoltorio, obtenemos una transparencia perfecta, ya que podemos interactuar directamente con la máquina objetivo.
Un estado de la máquina envoltorio es controlado de manera transparente por un conjunto de entradas I de la máquina objetivo si el testador puede estimular a la máquina envoltorio de manera que las reacciones producen salidas específicas que son interpretadas (de manera unívoca) como entradas de la máquina objetivo dadas por I. Esto es, para todas las entradas de la máquina objetivo I, la máquina envoltorio es una especie de intérprete que, de manera unívoca, transforma algunas de sus entradas en entradas de la máquina objetivo. Además, la máquina envoltorio también es transparente de manera observable para el mismo conjunto de entradas I de la máquina objetivo si las salidas producidas por ésta última (como respuesta de las entradas recibidas) son unívocamente trasladadas de vuelta por la máquina envoltorio, produciendo algunas salidas al mundo exterior que permiten al testador conocer la salida actual producida por la máquina objetivo. Definiremos transparencia de una máquina LTS consiste en la media de la transparencia de cada estado de la máquina. Debe quedar constancia, acorde con la siguiente definición, que sólo los estados de la máquina envoltorio que reciben entradas del mundo exterior son elegibles para ser transparentes. No obstante, estos estados deben ser soportados por otros estados de la máquina envoltorio, los cuales envían/reciben señales de la máquina objetivo y trasladar las salidas de la máquina objetivo al mundo exterior. De esta manera, incluso en el mejor caso, sólo algunos estados de la máquina envoltorio pueden ser considerados como transparentes. La métrica nos permite comparar la transparencia de diferentes máquinas envoltorio en términos de los valores devueltos.

Sea $M = M_1 \times M_2 = (S, I, O, s^{in}, T)$ un entorno de prueba integrado donde M_1 es la máquina envoltorio y M_2 la máquina objetivo. Suponemos $M_i = (S_i, I_i, O_i, {s_i}^{in}, T_i)$ para todo $i \in \{1, 2\}$. Dado $s \in S_1$, la transparencia de s, denotada por transparencia(s), es igual a 0 si $x \in trazas(M_1, s_0)$ donde $x \notin I$. La transparencia de s se define como sigue:

$$\frac{max\{|I| \mid I \text{ es transparente}\}}{|I_2|}$$

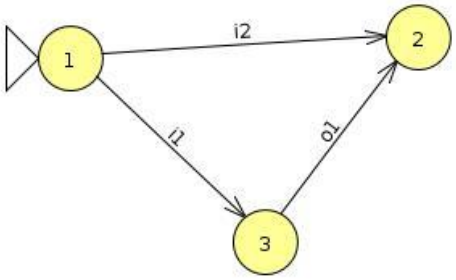
Donde $\mathcal{I} \subseteq I_2$ es transparente si se cumplen las siguientes condiciones:

(a) (Transparencia de control) Para todo $i \in \mathcal{I}$ existe $a \in I$ que $a \cdot i \in trazas(M_1, s)$ y no existe x con $x \neq a$ que $a \cdot i \in trazas(M_1, s)$.
(b) (Transparencia de observación) Para todo $i \in \mathcal{I}$ y $o \in O_2$ existe $a \in I$ y $b \in O$ que $a \cdot i \cdot o \cdot b \in trazas(M_1, s)$ para todo $a \cdot i \cdot x \in trazas(M_1, s)$ y donde $x \in O_2$ y no existe $o' \in O_2$ con $o' \neq o$ que $a \cdot i \cdot o' \cdot b \in trazas(M_1, s)$.

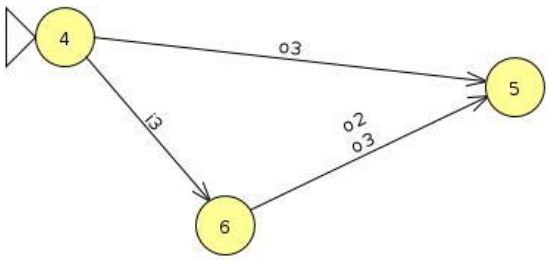
Sea $M = (S, I, O, s^{in}, T)$. La transparencia de M, denotada por $transparencia(M)$, esta dada por la expresión $\frac{\sum_{s \in S} transparencia(s)}{|S_1|}$.

A modo de ejemplo, definimos la transparencia de las máquinas que se muestran a continuación. La primera máquina actuará como máquina envoltorio, mientras que la segunda hará las veces de máquina objetivo. El primer paso será combinarlas para formar la máquina sobre la que realizar los cálculos.

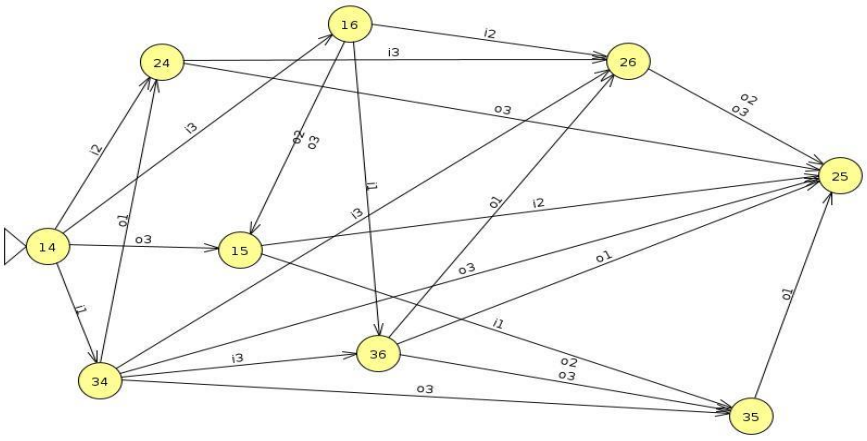
Máquina 1, que tiene como entradas {i1, i2} y como salidas {o1}.



Máquina 2, con entradas {i3} y salidas {o2, o3}.



Máquina resultado de la combinación de las máquinas 1 y 2.



Como se ha visto en las definiciones, se realizarán los cálculos sobre transparencia de control y observación, pero por separado. Calcularemos primero la transparencia de control y, después, la transparencia de observación.

A continuación se usará I1 para referirse a las entradas de la máquina envoltorio, I2 para las entradas de la máquina objetivo. Por otro lado, O1 para las salidas de la máquina envoltorio, así como O2 para las salidas de la máquina objetivo.

Transparencia de control: Consiste en generar caminos I1 -> I2 para cada estado de la máquina combinación. Al final, comprobamos el número de entradas de I2 que somos capaces de generar introduciendo entradas de I1. El total para cada estado se calcula sumando 1 para cada I2 que podemos generar y 0 para aquellas que no hemos conseguido generar, dividiendo el total entre el número de entradas de I2.

Fórmula: TC=k/n. Donde TC es usado como abreviación de Transparencia de Control, k es el número de entradas pertenecientes a I2 que podemos alcanzar con alguna entrada perteneciente a I1 y n es el número de entradas pertenecientes a I2.

La transparencia de control de la máquina combinada será la suma del total de cada uno de los estados dividida entre el número de estados de la máquina combinada.

Fórmula: $\sum_{i=1}^{n} TC_i$ / n

Transparencia de observación: En la definición formal explicada anteriormente, la transparencia se definía como la conjunción de la transparencia de control y observación. Finalmente se decidió abordar ambas por separado, por lo que se definió la transparencia de observación como sigue:

Para cada estado de la máquina producto de la combinación de las máquinas envoltorio y objetivo, se obtiene el conjunto de caminos que incluyan la secuencia {I1 -

> I2 -> O2 -> O1}, entendiendo I1/O1 como entradas/salidas de la máquina envoltorio; I2/O2 entradas/salidas de la máquina objetivo. Descartando los caminos idénticos (si los hubiera) se mira, para cada entrada I1 el subconjunto de caminos de la lista anterior que comienzan con dicha I1, si existen dos o más caminos que comienzan con la misma I1 y terminan con la misma 01. Como se sabe que son distintos (porque las réplicas han sido descartadas, escogiendo el original), se sabe que van a tener distinta O2. Un indicativo de que hay falta de transparencia es si, introduciendo la misma I1, alcanzamos la misma O1 a través de distintas O2. Por tanto, se cuenta el número de caminos distintos tales que I1 -> I2 -> 02 ->O1 y I1-> I2 -> O2' -> O1.

Fórmula: TO=$\sum_{i=1}^{n} TO_i$. Donde $TO_i = 1/n$ (n es el número de secuencias distintas I1 -> I2 -> O2 -> O1 para la entrada i_i perteneciente a I1) es la transparencia para cada entrada i_i que pertenece a I1.

Para cada entrada I1 se realiza la inversa 1/n, siendo n el número de caminos distintos para la entrada I1 en el estado que se están mirando. Como se ha explicado anteriormente, un camino es considerado distinto a otro si tienen distinta O2 y la misma O1 o si tienen la misma O2 y distinta O1.

Si no hay caminos para una entrada I1, la transparencia se considera 1.

Al final, el total se calcula sumando los totales de todos los estados de la máquina combinación entre el número de estados de la misma.

Fórmula: $\sum_{i=1}^{n} TC_i$ / n

Cálculos para la transparencia de control en el ejemplo:

Estado 14: Sólo hay una entrada perteneciente a I2 (i3) así que, con encontrar al menos un camino I1 -> I2, se habrá conseguido el objetivo. Para este estado hay al menos uno {14 i1 34 i3 36}, por lo que la transparencia de control para el estado 14 es $\frac{1}{1}$ =1.

Estado 34: No tiene caminos que incluyan la secuencia I1 -> I2. Por tanto, transparencia de control 0.

Estado 24: No tiene caminos que incluyan la secuencia I1 -> I2. Por tanto, transparencia de control 0.

Estado 16: No tiene caminos que incluyan la secuencia I1 -> I2. Por tanto, transparencia de control 0.

Estado 15: No tiene caminos que incluyan la secuencia I1 -> I2. Por tanto, transparencia de control 0.

Estado 36: No tiene caminos que incluyan la secuencia I1 -> I2. Por tanto, transparencia de control 0.

Estado 35: No tiene caminos que incluyan la secuencia I1 -> I2. Por tanto, transparencia de control 0.

Estado 26: No tiene caminos que incluyan la secuencia I1 -> I2. Por tanto, transparencia de control 0.

Estado 25: No tiene caminos que incluyan la secuencia I1 -> I2. Por tanto, transparencia de control 0.

Transparencia de control total: $\frac{1+0+0+0+0+0+0+0+0}{9}$ = 0,111

Donde 9 es el número de estados de la máquina combinada.

Cálculos para la transparencia de observación en el ejemplo:

Estado 14: Tiene dos caminos: {14 i1 34 i3 36 o2 35 o1 25} y {14 i1 34 i3 36 o3 35 o1 25}. Por tanto, introduciendo la entrada i1, obtenemos la salida o1. A priori parece bueno, pero en medio se observa que hay dos salidas de la máquina 2 que producen la misma salida de la máquina 1, eso reduce la transparencia. Por tanto, para la entrada i1 en el estado 14, se obtiene como transparencia de observación $\frac{1}{2}$. Para la entrada i2 no se obtiene ningún camino I1 -> I2 -> O2 -> O1, por lo que la transparencia para el estado 14 para la entrada i2 es 1. Así, la transparencia total del estado 14 es $\frac{0,5+1}{2}$ =0,75.

Estado 34: No tiene caminos que incluyan la secuencia I1 -> I2 -> 02 -> O1 sin pasar por un ciclo. Por tanto, transparencia de observación 1 ($\frac{1+1}{2}$ =1).

Estado 24: No tiene caminos que incluyan la secuencia I1 -> I2 -> 02 -> O1 sin pasar por un ciclo. Por tanto, transparencia de observación 1 ($\frac{1+1}{2}$ =1).

Estado 16: No tiene caminos que incluyan la secuencia I1 -> I2 -> 02 -> O1 sin pasar por un ciclo. Por tanto, transparencia de observación 1 ($\frac{1+1}{2}$ =1).

Estado 15: No tiene caminos que incluyan la secuencia I1 -> I2 -> 02 -> O1 sin pasar por un ciclo. Por tanto, transparencia de observación 1 ($\frac{1+1}{2}$ =1).

Estado 36: No tiene caminos que incluyan la secuencia I1 -> I2 -> 02 -> O1 sin pasar por un ciclo. Por tanto, transparencia de observación 1 ($\frac{1+1}{2}$ =1).

Estado 35: No tiene caminos que incluyan la secuencia I1 -> I2 -> 02 -> O1 sin pasar por un ciclo. Por tanto, transparencia de observación 1 ($\frac{1+1}{2}$ =1).

Estado 26: No tiene caminos que incluyan la secuencia I1 -> I2 -> 02 -> O1 sin pasar por un ciclo. Por tanto, transparencia de observación 1 ($\frac{1+1}{2}$ =1).

Estado 25: No tiene caminos que incluyan la secuencia I1 -> I2 -> 02 -> O1 sin pasar por un ciclo. Por tanto, transparencia de observación 1 ($\frac{1+1}{2}$ =1).

Transparencia de observación total: $\frac{0'75+1+1+1+1+1+1+1+1}{9}$ = 0,972

Donde 9 es el número de estados de la máquina combinada.

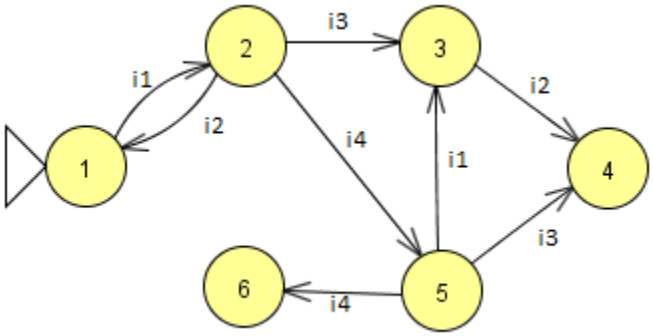
2.9 Complejidad

Otra medida para una máquina es la complejidad. Hace falta definir dos constantes α y β tales $\alpha + \beta = 1$. La complejidad de M para los parámetros alfa y beta, denotada como $complejidad\ (M, \alpha, \beta)$ es igual a $\alpha * |S| + \beta * |T|$ donde $|S|$ es el cardinal de S, es decir, el número de estados de la máquina asociada y $|T|$ es el cardinal de T, es decir, el número de transiciones de la máquina asociada.

Se demuestra este cálculo con un ejemplo explicativo:

Para obtener la complejidad de una máquina, primero se obtiene el número de estados de la máquina así como el número de transiciones. Dados estos dos valores, se obtienen otros dos valores α y β con valores entre 0 y 1 de tal forma que entre los dos sumen 1. Se ha dado igual importancia tanto a la cantidad de estados como a la cantidad de transiciones, ya que los valores de α y β se han fijado a 0,5 cada uno. Lo que resta es multiplicar el valor α por el número de estados y sumarle la multiplicación de β por el número de transiciones.

Siendo este el sistema de transiciones etiquetadas:



Se obtienen el número de estados y de transiciones, así como los valores de α y β que son 0,5 cada uno. Los valores de $|S|$ y $|T|$ son 6 y 7 respectivamente:

$\alpha * |S| + \beta * |T| = 0.5 * 6 + 0.5 * 8 = 7.0.$

Capítulo 3: Metodología

Nuestro experimento para medir el peso que deben tener la controlabilidad, observabilidad, transparencia y complejidad en la testeabilidad de sistemas consiste en los siguientes pasos:

1. Se generan aleatoriamente varias LTS para ser usadas como objetivos y envoltorios en nuestro caso de estudio. Cada LTS es generado aplicando reglas probabilísticas descritas con detalle en la sección 3.1. Para cada LTS que va a ser usada como objetivo medimos su controlabilidad, observabilidad y complejidad. Para cada LTS usada como envoltorio, junto con la objetivo, se mide su transparencia.

2. Para cada máquina objetivo, se generan aleatoriamente varios mutantes. Un mutante es una máquina igual a la original salvo en un punto donde hemos introducido una mutación. Las mutaciones intentan simular los errores conceptuales que puede cometer un desarrollador mientras implementa un sistema. Una mutación puede ser la eliminación de un estado, reemplazar entradas y salidas, eliminar una transición, añadir un estado, etc. Las operaciones usadas para generar mutantes están descritas en la sección 3.2.

3. Para cada mutante de la máquina objetivo original se aplican dos estrategias de testing:

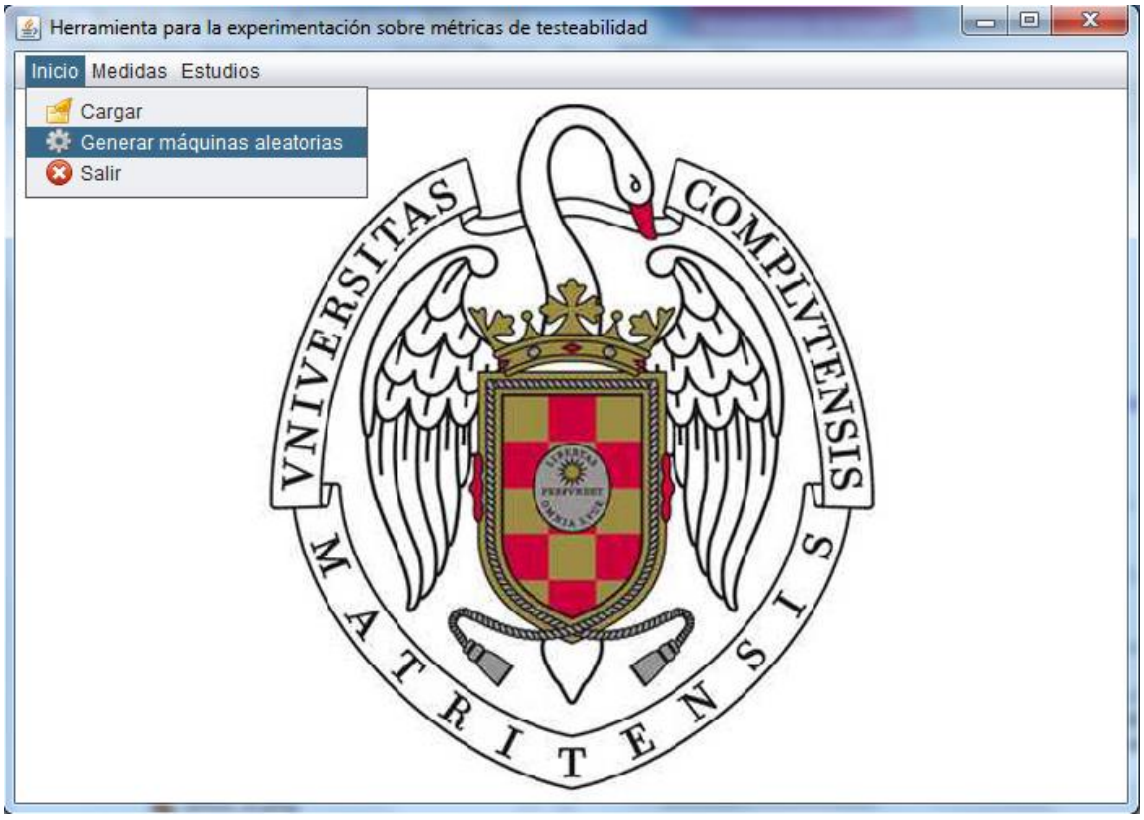
    a. Camino más largo: se almacena el conjunto de entradas y salidas que dan lugar a un camino más largo (visita el mayor número de estados).
    b. Testing adaptativo: unión de caminos que cubren la totalidad de las transiciones de la máquina.

Ambas estrategias de testing son descritas en detalle en la sección 3.3. Se aplica la estrategia a todos los mutantes. Para cada mutante analizamos la secuencia de respuestas (salidas) que genera. Se quiere observar si dicha secuencia es posible en el sistema original. Si la secuencia de salidas puede darse también en el sistema original, el comportamiento del mutante no es considerado como defectuoso. De otra manera, diremos que hemos detectado una falta en el mutante y, siguiendo la notación que hemos impuesto, lo expresaremos como la muerte del mutante. Para esta estrategia de testing, mediremos el número de mutantes muertos. Entonces, diremos que el grado de testeabilidad de la tripleta (estrategia, envoltorio y objetivo) es igual a la proporción de mutantes que han sido matados.

4. Para cada estrategia, máquina envoltorio y objetivo, estudiamos la comparación entre el resultado de los experimentos con mutantes y las medidas.

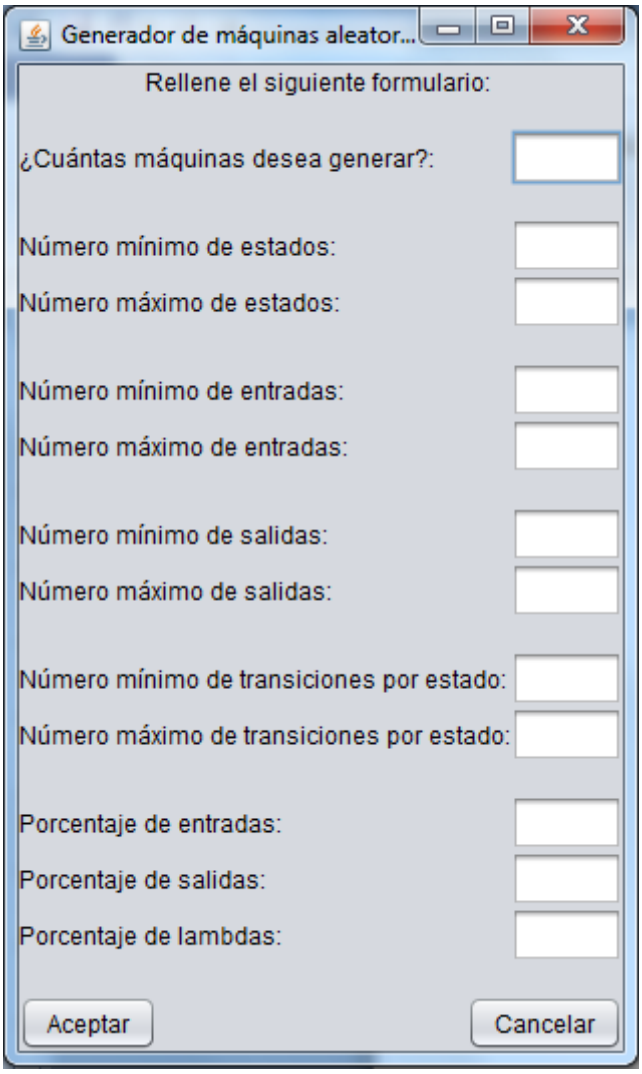
3.1 Generación de máquinas aleatorias

En este apartado, se va a explicar cómo se generan máquinas aleatorias en la herramienta desarrollada y cómo se clasifican. Inicialmente, el usuario seleccionará en el menú Inicio la opción "Generar máquinas aleatorias"

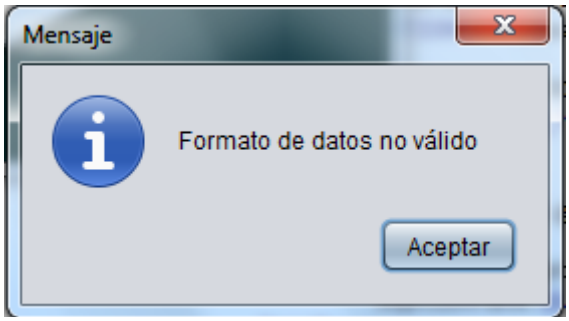


A continuación, se mostrará un formulario donde se pedirán los datos necesarios para generar las máquinas aleatorias.

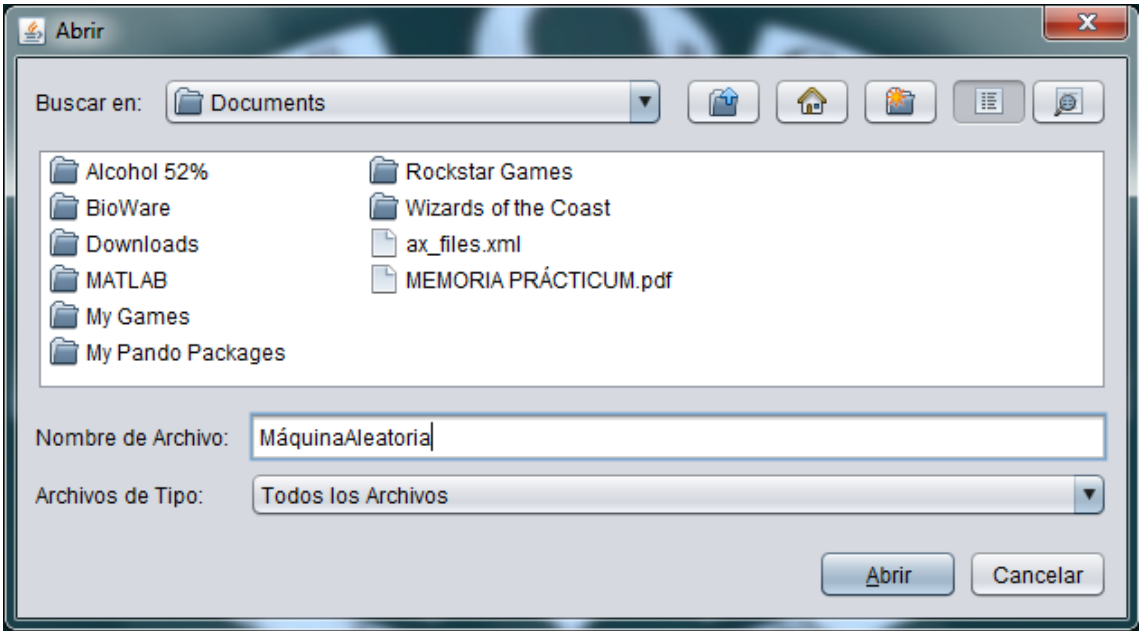
1. Número de máquinas aleatorias que desea generar.
2. Número mínimo y máximo de estados.
3. Número mínimo y máximo de entradas.
4. Número mínimo y máximo de salidas.
5. Número mínimo y máximo de transiciones salientes por estado.
6. Porcentaje de entradas, salidas y τ. Este porcentaje nos permitirá tener cierto control a la hora de generar máquinas más o menos indeterministas.



También debemos señalar que el proceso no continuará hasta que no se hayan introducido valores válidos (coherentes), mostrando el oportuno mensaje de error.
El siguiente mensaje aparecerá si no se introducen valores numéricos en el formulario o si se deja algún campo sin rellenar.



Por último, el sistema le pedirá al usuario que introduzca la ubicación donde desea almacenar las máquinas generadas, así como el nombre que desea dar a los ficheros en los que se almacenará. Deberá introducir un nombre genérico y el sistema asignará a cada fichero, el nombre introducido seguido del número de la máquina. Por ejemplo, si decidimos generar 3 máquinas con el nombre "MáquinaAleatoria", en la ubicación elegida se generarán 3 ficheros con los siguientes nombres: "MáquinaAleatoria1.txt", "MáquinaAleatoria2.txt" y "MáquinaAleatoria3.txt".



Con esta información recogida el sistema generará cada máquina aleatoria de la siguiente manera:

1. Estados: se generará un número aleatorio entre el mínimo número de estados y el máximo que introdujo el usuario que determina el número de estados de la máquina. Dichos estados comenzarán en el 1 y finalizarán con el número aleatorio obtenido.

2. Entradas: se generan n entradas aleatorias (siendo n el número de entradas introducido por el usuario). Se nombrarán i_n, donde n es el número de la entrada. Para evitar que las máquinas generadas tengan las mismas entradas, para máquinas correlativas, se utilizarán entradas correlativas también. Por

ejemplo, si la primera máquina tiene entradas i_1 … i_n, la máquina 2 tendrá entradas a partir de i_{n+1} .

3. Salidas: se determinarán de manera análoga a las entradas, exceptuando el nombre que será el de o_n.

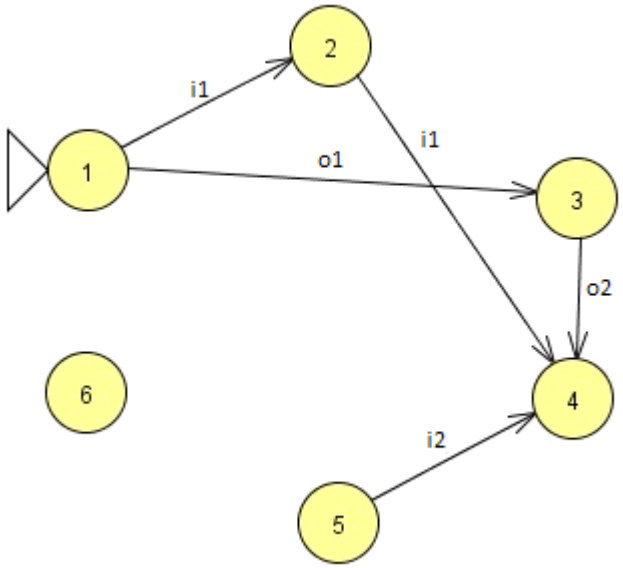
4. Estado inicial: por convenio, se denotará como estado inicial el 1.

5. Transiciones: por último, se procederá a obtener las transiciones de la máquina. Para ello, en cada estado de la máquina se obtiene un número aleatorio, entre el mínimo número de transiciones salientes por estado y el máximo que introdujo el usuario, que determinará el número de transiciones salientes desde cada estado. Llegado este punto, para cada transición se debe decidir si transitará mediante una entrada, una salida o una transición τ. Para decidirlo, se procede a obtener un número aleatorio entre 1 y 100.

    a. Si se obtiene un número menor o igual al porcentaje de entradas introducido por el usuario, la transición se realizará mediante una entrada.
    b. Si no, si se obtiene un número menor o igual al porcentaje de entradas más el porcentaje de salidas introducidos por el usuario, la transición se realizará mediante una salida.
    c. En otro caso, la transición se realizará mediante τ.

De cualquier forma, el estado destino de la transición se obtendrá de manera pseudoaleatoria entre el conjunto de estados de la máquina. También señalar que, en los casos a y b, la entrada o salida utilizada en la transición será obtenida de manera pseudoaleatoria, entre el conjunto de entradas o salidas de la máquina.

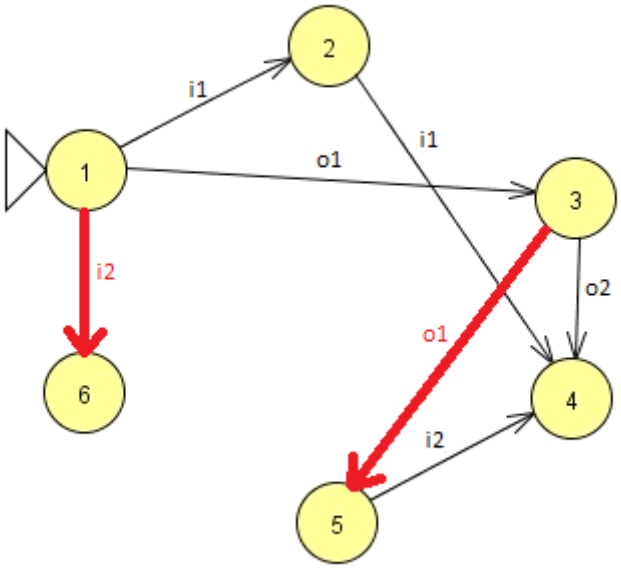
De esta manera la máquina queda generada, sin embargo es posible que no de manera válida. Es posible que, debido a la aleatoriedad del método de generación de máquinas, algunos estados queden no alcanzables, es decir, que partiendo desde el estado inicial no se pudiera llegar a ellos por medio de las transiciones disponibles, lo que supondría un problema a la hora de realizar el estudio de la máquina, además no se ajustaría a un modelo más o menos real.



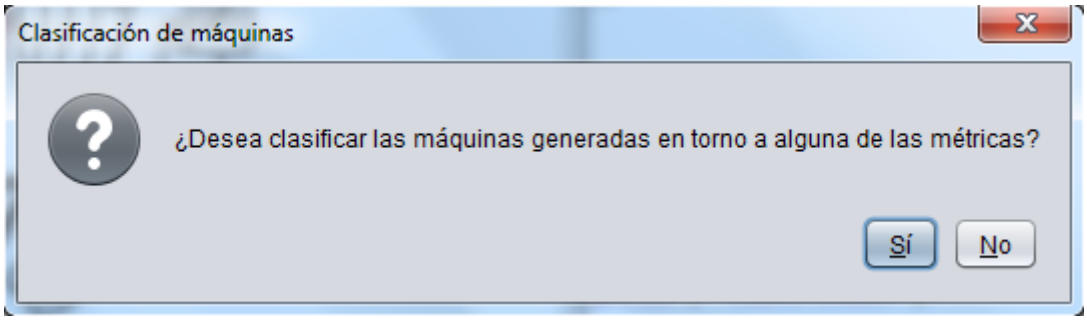
Para solucionar este problema, al finalizar la construcción de la máquina se procederá a comprobar que es válida. En caso de que no serlo, se generarán transiciones pseudoaleatoriamente de forma que los estados que no eran alcanzables lo sean y así se obtenga una máquina válida. Se procederá, por tanto, del siguiente modo:

1. Se generarán dos listas. La primera contendrá los estados que son alcanzables y la segunda los estados que no lo son.

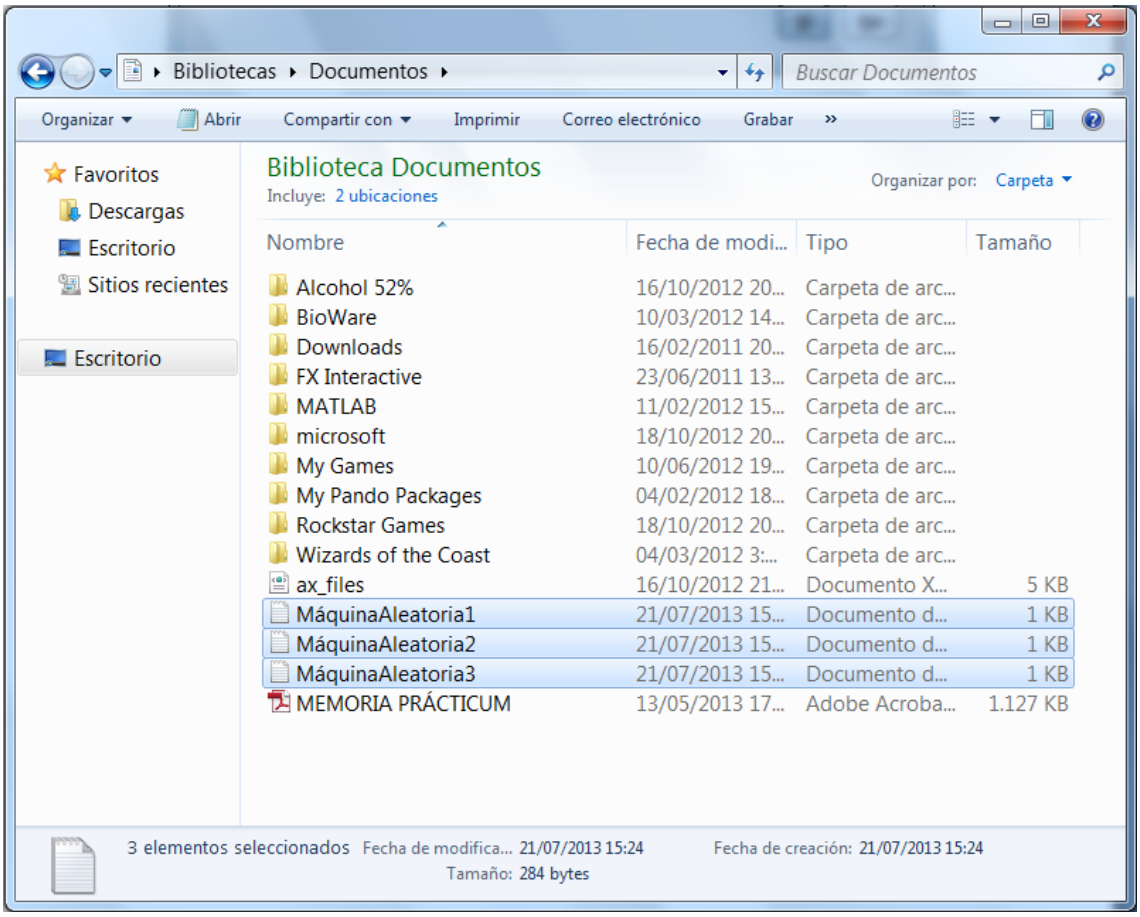
2. Lo que se pretende hacer es generar transiciones de manera que los estados no alcanzables dejen de serlo. Para ello, se generará una transición por cada estado no alcanzable almacenado en la lista obtenida anteriormente. El estado no alcanzable será el destino de la transición, que partirá desde uno de los estados alcanzables de la otra lista, con lo que se consigue que los estados no alcanzables dejen de serlo. El procedimiento será similar al utilizado a la hora de generar transiciones en la máquina. Es decir, mediante un número aleatorio decidiremos si la transición se realiza mediante una entrada, una salida o una transición. Las probabilidades serán las mismas que las descritas anteriormente. Una vez obtenido el tipo de transición, basta con obtener el estado de partida de manera aleatoria entre el conjunto de estados alcanzables y la entrada o salida (en caso de que haya sido así seleccionado) entre en conjunto de entradas o salidas de la máquina.



Una vez realizada la comprobación de la validez de la máquina y subsanada en el caso necesario, se mostrará un mensaje al usuario para darle la opción de clasificar las máquinas generadas en grupos según alguna de las métricas con las que trabajamos.



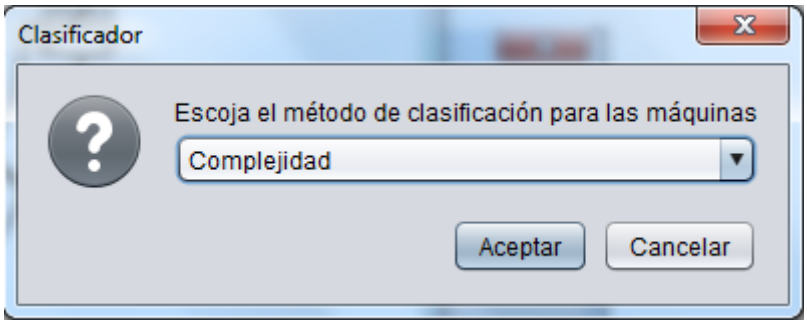
Si el usuario decide no clasificarlas, las máquinas quedan almacenadas en el directorio que seleccionó, y finalizará el proceso de generación de máquinas aleatorias.



Si decide clasificarlas, aparecerá un nuevo mensaje con los distintos métodos de clasificación posibles.

1. Complejidad.
2. Controlabilidad.
3. Observabilidad.

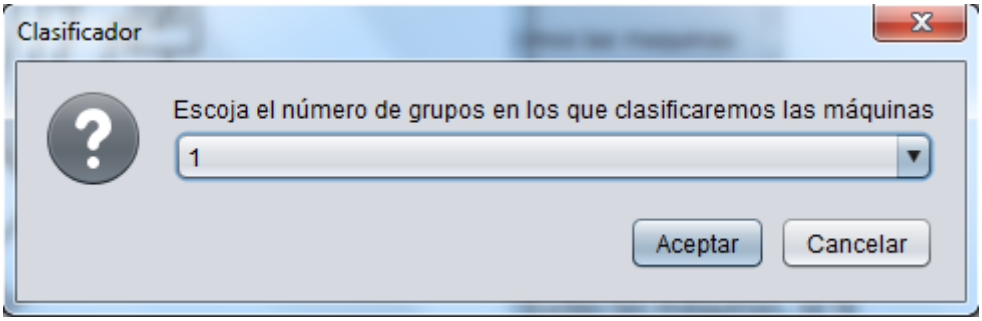
Nótese que no es posible utilizar como método de clasificación la transparencia ya que ésta requiere de la composición de dos máquinas para ser obtenida.



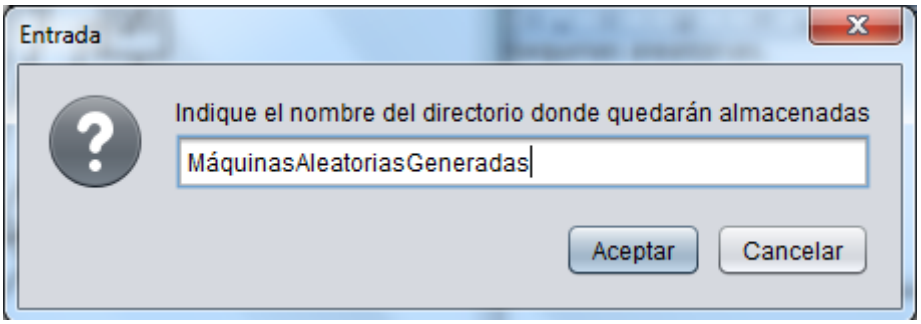
Una vez decidido el método que utilizaremos para clasificarlas, resta indicar el número de grupos en los que queremos clasificar las máquinas. Se calculará la métrica escogida para cada máquina, de manera que el valor mínimo y máximo sean los límites del intervalo. Se generarán tantas carpetas como grupos se han escogido, de manera que

se incluya el mismo número de máquinas en cada carpeta. Las distintas opciones que se muestran son:

A. 1: disponible en todos los casos.
B. 2: disponible si se ha generado más de una máquina aleatoria.
C. 3: disponible si se han generado más de dos máquinas aleatorias.
D. 4: disponible si se han generado más de tres máquinas aleatorias.
E. 5: disponible si se han generado más de cuatro máquinas aleatorias.
F. 6: disponible si se han generado más de cinco máquinas aleatorias.
G. 7: disponible si se han generado más de seis máquinas aleatorias.
H. 8: disponible si se han generado más de siete máquinas aleatorias.
I. 9: disponible si se han generado más de ocho máquinas aleatorias.
J. 10: disponible si se han generado más de nueve máquinas aleatorias.



Tras seleccionar el número de grupos en los que serán clasificadas las máquinas, se le pedirá al usuario que indique el nombre del directorio donde desea que se almacenen las máquinas clasificadas. Este directorio se ubicará en el directorio del proyecto.



Con todo esto, el clasificador tiene toda la información necesaria para la clasificación de las máquinas en grupos. Dicha clasificación se realizará como se indica a continuación:

1. Inicialmente insertaremos en una lista los pares número de máquina y valor de dicha máquina para la métrica seleccionada.

2. El siguiente paso consiste en ordenar dicha lista en función de la métrica seleccionada por el usuario, para ello utilizaremos el método del ordenación *quicksort*.

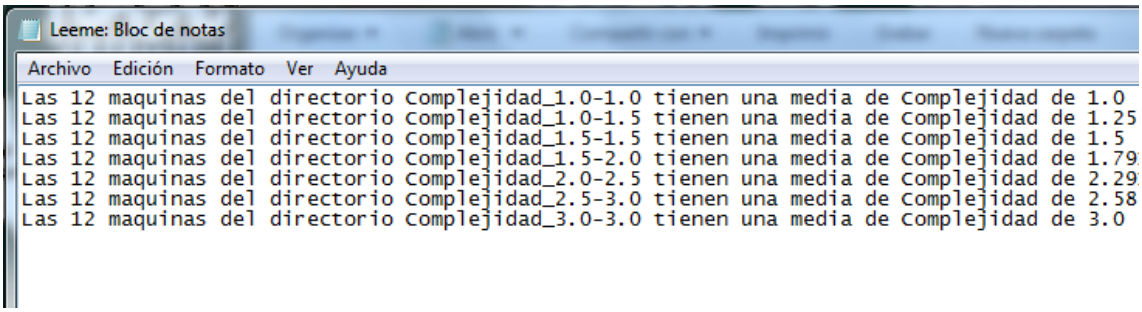
3. Una vez ordenadas, lo que resta es ir copiándolas en los directorios oportunos. Los nombres que iremos dando a los directorios, vienen dados por el siguiente formato:

   MétricaSeleccionada_ValorMaquinaInicial-ValorMaquinaFinal donde:

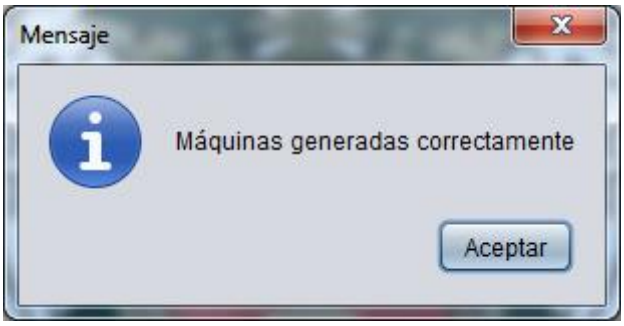
   A. MétricaSeleccionada: puede ser Complejidad, Controlabilidad u Observabilidad.
   B. ValorMaquinaInicial: es el valor de la métrica seleccionada para la primera máquina que ubicaremos en dicho directorio.
   C. ValorMaquinaFinal: es el valor de la métrica seleccionada para la última máquina que ubicaremos en dicho directorio.

   Según las vamos copiando en el directorio oportuno, los ficheros generados inicialmente los vamos borrando quedando así las máquinas clasificadas.

4. Por último señalar, que también será generado un fichero con el nombre "Leeme.txt" que contendrá información sobre el valor medio de la métrica utilizada para clasificar las máquinas. Este valor medio se calcula para cada grupo de máquinas almacenadas en los directorios descritos anteriormente.

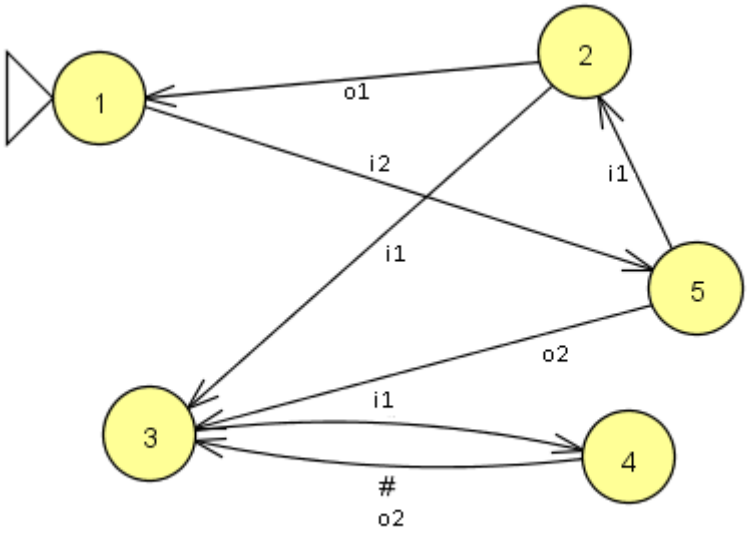


Llegado este punto, finaliza la generación y clasificación de máquinas aleatorias mostrando al usuario un mensaje confirmatorio de este hecho.



3.2 Generación de máquinas mutantes

Se define la mutación de una máquina como una variación que la distingue de la original. Para ello se distinguen seis tipos de mutación: añadir y eliminar un estado, añadir y eliminar una transición, cambiar una transición de tal modo que haya una entrada y cambiar una transición de tal modo que haya una salida.
A continuación se muestra una máquina ejemplo a partir de la cual se procederán a realizar las distintas mutaciones sobre ella.

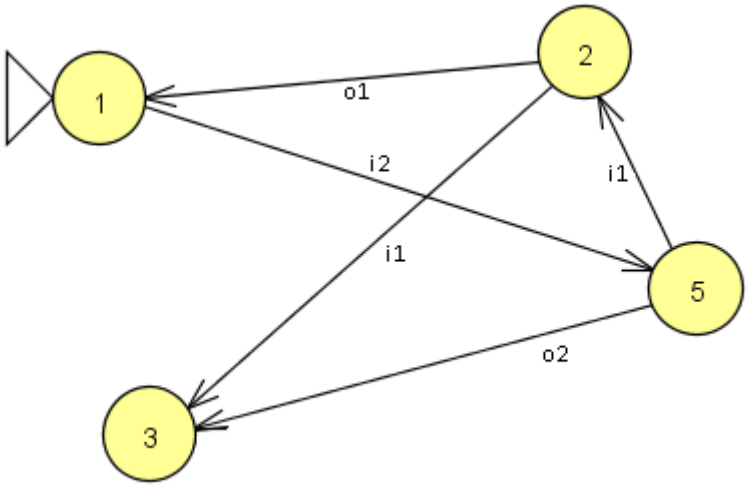


1. Eliminación de un estado
   La máquina original se ve alterada por la eliminación de uno de sus estados. Como cabe esperar, el estado escogido no puede ser el inicial de la máquina. En la definición de la máquina LTS, se asume que toda máquina es una conjunción de estados unidos mediante transiciones, que cuelgan de un estado inicial. Por tanto, el estado inicial de la máquina no puede eliminarse, en caso contrario quedaría un LTS inconexo e incoherente.

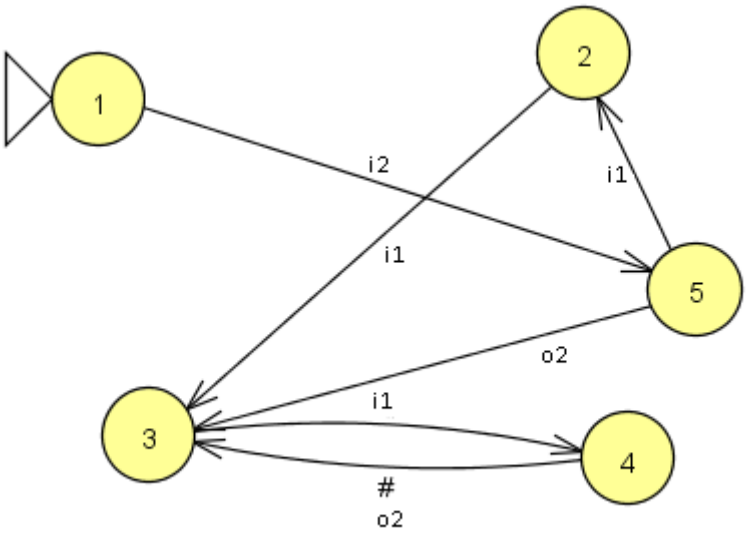
   Una vez que está claro lo que no se puede eliminar, se pasa a calcular aleatoriamente el estado a eliminar. En principio puede ser cualquier estado de la máquina, así que la elección es libre, sin condicionamientos externos, más allá de no poder ser el estado inicial. Cuando tenemos el estado aleatorio, el planteamiento siguiente es pensar si, al eliminarlo, la máquina se queda incoherente. Realmente sólo pueden inquietar dos opciones. La primera es que otros estados no puedan llegar hasta aquél que se pretende eliminar, pero esto no supone un problema puesto que, si se elimina, se deben eliminar también todas las transiciones de otros estados que lleguen hasta él. Sin embargo, la siguiente opción sí hay que tenerla muy en cuenta, se trata de los estados a los que se puede llegar con el estado a eliminar. En este punto puede quedar la máquina con estados no conexos, es decir, puede quedar una máquina incoherente. Por tanto, se debe comprobar, uno a uno, si los estados a los que se puede llegar desde el estado candidato para ser eliminado, pueden ser visitados mediante transiciones que vienen de otros estados.

Una vez que se ha comprobado que ningún estado quedaría *huérfano*, será el momento de eliminar el estado. A la vez de eliminar el estado, tenemos que eliminar aquellas transiciones que llegaban hasta él desde otro estado o, aquellas que salían del propio estado hacia otros.

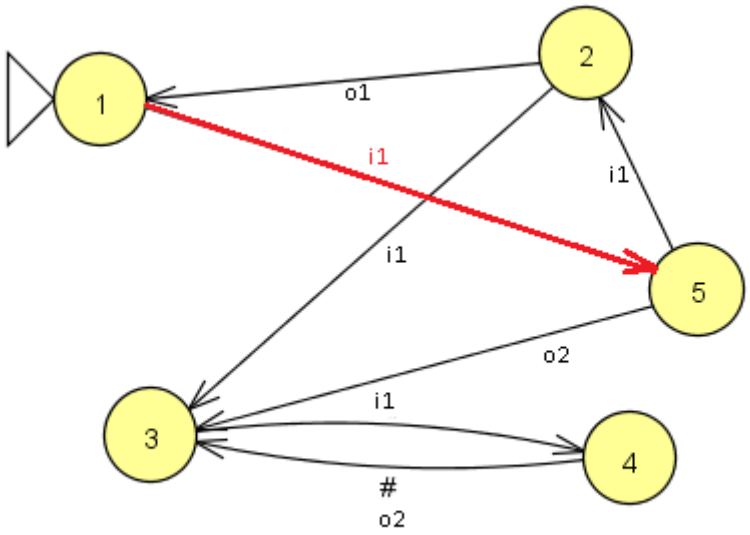
A continuación se muestra un ejemplo respecto a la maquina original en la que se ha eliminado un estado, en este caso el estado número 4, junto a sus respectivas transiciones:



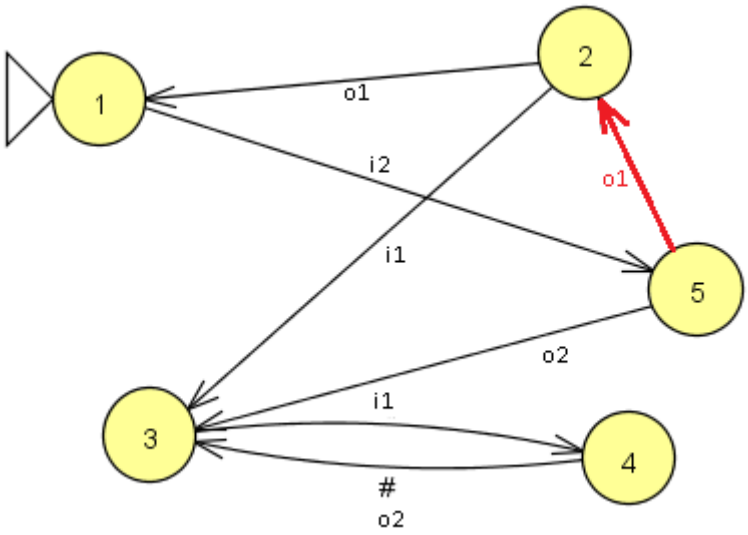
2. Eliminación de una transición
   El primer paso es comprender que el concepto de transición está ligado al de estado. Una transición es el camino que surge de un estado (al que podemos llamar origen de la transición) y conduce a otro estado. Por tanto, para eliminar una transición al azar, se debe buscar un estado al azar. En este caso, a diferencia del punto anterior, se puede elegir el estado inicial. De nuevo, se escoge un estado aleatoriamente. Una vez que se sabe el estado, debe elegirse una transición al azar. Ahora surge de nuevo la opción de que, al eliminar esa transición escogida aleatoriamente, el estado al que va quede inalcanzable. Se debe comprobar que ese estado está unido al resto de estados, esto es, que desde el estado inicial se puede llegar a ese estado por otro camino que no sea el que acabamos de escoger para eliminar. Si el estado siguiente (al que va la transición escogida al azar) no queda inconexo, se puede eliminar. Evidentemente, si se escoge al azar un estado sin transiciones hacia otros estados, deberá escogerse de nuevo otro estado aleatorio.
   A continuación se puede observar un ejemplo de una máquina, en la que se ha eliminado una transición que había en la original. La transición eliminada ha sido la que iba desde el estado 2 al estado 1 con la salida o1.



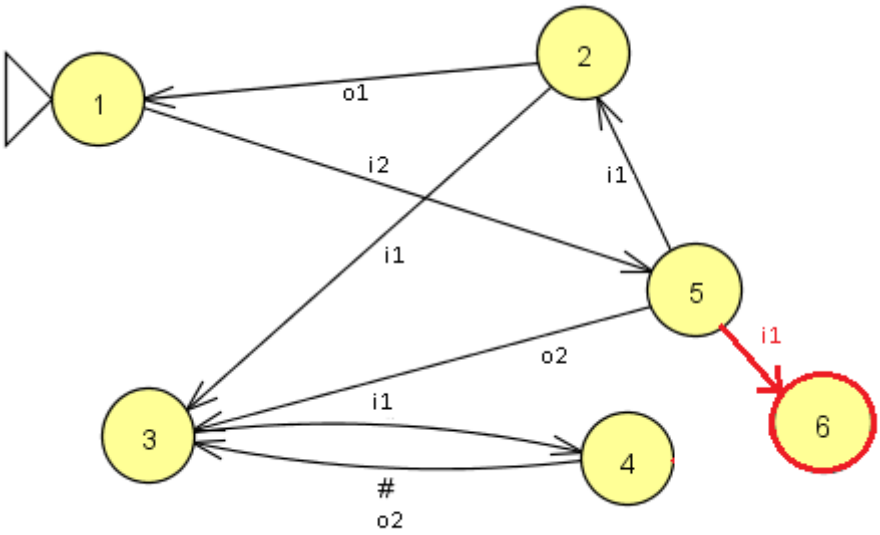
3. Sustitución de una transición por una entrada
   Lo primero es escoger una entrada aleatoria del conjunto de entradas de la máquina. Posteriormente se escoge un estado al azar y, en función de éste, una transición del mismo. El interés de este cambio reside en la naturaleza distinta entre entradas y salidas. Ninguna otra limitación especial, puesto que no es tan delicado como eliminar una transición o un estado, que puede dar paso a una incoherencia en la máquina.
   En el siguiente ejemplo de máquina, se puede observar como se ha cambiado una transición que tenía entrada i2 y se le ha puesto la entrada i1.



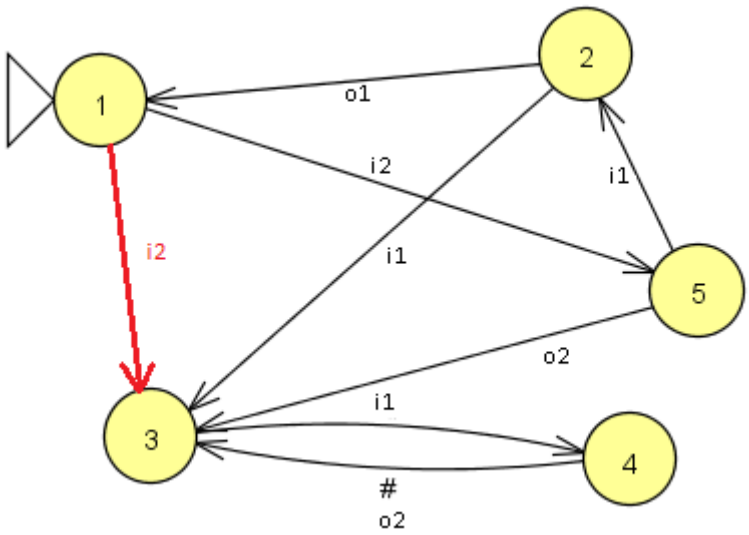
4. Sustitución de una transición por una salida
   Sigue la inercia del punto anterior. Se escoge una salida al azar, un estado al azar y una transición del mismo. En el caso de que el estado no tuviese transiciones habría que escoger otro estado.
   A continuación se muestra un ejemplo de máquina a la que se le ha cambiado una transición con entrada que tenía i1 y se le ha puesto salido o2.



5. Añadir un estado
   Consiste en añadir un nuevo estado a la máquina. Para ello se le da un nombre correcto que no esté ya en la máquina, en este caso el siguiente del último estado de la máquina. Al añadirle se queda inconexo con el resto de estados, por eso se tiene que añadir una transición que vaya al nuevo estado creado, ya sea mediante transición de entrada o de salida.
   En el ejemplo se puede observar cómo se ha añadido el estado 6, que es el siguiente a los que había y además una transición con entrada i1 desde el estado 5 al estado 6.



6. Añadir una transición
   Esta mutación añade una transición, ya sea con entrada o salida de las disponibles en la máquina, a cualquiera de los estados que tiene la máquina en cuestión.
   En el ejemplo se observa que se ha añadido la transición con entrada i2 que va desde el estado 1 al estado 3.



3.3 Estrategias de testing

Existen tres técnicas de testing distintas que se van a definir en los puntos siguientes. Tienen como finalidad proporcionar un *camino* a seguir. De esta forma, se obtienen una serie de entradas y salidas. La finalidad de este método de testing es comprobar si la máquina mutante se comporta como la original. Para ello se toman las entradas que han proporcionado los siguientes métodos de testing, y se comprueba que la máquina original es capaz de generar las mismas salidas que las máquinas mutadas:

1. Random Walk
   Consiste en hacer transitar a la máquina mutante, guardando las entradas y salidas generadas por la misma. La cuestión que plantea este método es muy simple, se trata de transitar sin pensar en lo que habrá después. Simplemente es otorgar una probabilidad a cada transición del estado que estamos evaluando, evidentemente a todos la misma. Por tanto, se va transitando de un estado a otro en función de probabilidades. Con cada movimiento se guarda la entrada/salida propia de la transición.

   Hay un aspecto que hay que tener en cuenta y son los ciclos. Una máquina con ciclos podría generar un bucle infinito a la hora de transitar. Una posible solución es limitar el número de pasos, de esta forma conseguimos un camino que, termina en un número de pasos máximo.

2. Camino más largo
   Durante el proceso de generación de caminos correspondiente al estudio de la controlabilidad, se almacena el conjunto de entradas y salidas que dan lugar al camino más largo (visita mayor número de estados). El proceso es el siguiente: para cada uno de los estados se generan todos los caminos que, comenzando en el estado inicial, terminan en el estado evaluado. Se almacenan a la vez las entradas/salidas que van guiando a través de los estados. Al finalizar la generación de los caminos hacia el estado, se guarda el mayor de todos (el que visita mayor número de estados). Sucesivamente, tras generar todos los

caminos que van del primer estado al resto de estados, se va actualizando el camino almacenado como el "mejor", esto es, si el camino más largo del nuevo estado evaluado es mayor que el que se tenía, se guarda en lugar del anterior.

3. Testing Adaptativo
   Se ha escogido una de las posibles variantes para este modelo. En concreto, la cobertura de aristas del grafo. Dado un LTS (se puede interpretar el conjunto de estados y transiciones de la máquina como tal), nuestro testing adaptativo consistirá en escoger caminos de entradas y salidas que cubran todas las aristas del LTS.

   Para conseguir el recubrimiento de la totalidad de las aristas se ha aprovechado el cálculo de la controlabilidad. Ya que en la controlabilidad se conseguían todos los caminos desde el estado inicial al conjunto de estados, se puede cubrir la totalidad de las aristas cogiendo uno a uno dichos caminos.

   El primer paso, evidentemente, es ordenar los caminos de mayor a menor número de aristas cubiertas. Parece lógico pensar que es mejor escoger primero el camino de mayor longitud, para así tener que coger un menor número de caminos a la hora de la cobertura. El primer camino siempre se añade. A continuación se pasa a considerar el resto de caminos uno a uno. Si el camino nuevo alberga alguna arista no incluida en los anteriores, se incluye.

   De esta forma se ha conseguido una lista de caminos que desde el estado inicial. Con nuestro *reset* se simula el comportamiento de un sistema real, ya que parece intuitivo que en una aplicación se pueda volver al punto inicial toda vez que se han completado una serie de operaciones.

Capítulo 4: Diseño de las clases principales

4.1 Clase Estado

La clase estado tiene las siguientes propiedades que se van detallar a continuación:

1. Nombre
   Entero que almacena el nombre del estado. Su finalidad es evidente, identificar el estado.

2. Transiciones
   HashMap cuya clave es un *String* que denota la entrada/salida de la transición, el valor es una lista de enteros que indican el conjunto de estados a los que se transita desde el estado que contiene las transiciones (*nombre*) con la entrada/salida que denota la clave.

Esta propiedad es fundamental, ya que mantiene toda la coherencia de la máquina y permite transitar a través de ella, así como el correcto desarrollo de los cálculos que se efectúan sobre la máquina.

3. Entradas
   Lista de tipo *String* con todas las entradas y salidas que admite el estado. De esta manera no hay que recorrer el conjunto de entradas/salidas de la máquina, sino sólo aquellas que están presentes en las transiciones del estado.

4. Número de transiciones posibles
   Entero que denota el número de transiciones totales que hay en el estado. De esta forma se puede saber el total de las mismas y asignar probabilidades a cada una de ella de forma que, transitar por las mismas, sea completamente aleatorio.

5. Etiquetas
   Es una variable de tipo *EstadoProbabilidad*, definido en la sección 4.4, que indica (en las máquinas combinadas) a qué máquina pertenece el estado.

4.2 Clase Máquina

La clase máquina tiene las siguientes propiedades:

1. Estados
   Lista compuesta por el conjunto de estados de la máquina. Cada estado es una instancia de la clase *Estado*, definida en la sección 4.1.

2. Entradas
   Lista de tipo *String* con todas las entradas que admite la máquina. Permite conocer en cada momento qué tipo de entradas pueden producir transiciones en los estados de la máquina. Como las transiciones de un estado están incluidas dentro del HashMap, entonces permite buscar para cada entrada, la lista asociada a dicha letra en el HashMap (siendo una lista vacía en caso de no tener transiciones para esa entrada).

3. Salidas
   Lista de tipo *String* con todas las entradas que admite la máquina. Tiene la misma funcionalidad que la lista de las salidas explicada en el punto anterior.

4. Estado actual
   Es un entero que tiene como finalidad indicar el estado en el que se encuentra la máquina. Por defecto, ese valor se corresponderá con el estado inicial.

5. Número de entradas de M2
   Entero que indica el número de entradas de la máquina 2. Por defecto será cero, salvo en los casos de una máquina compuesta. Su finalidad es almacenar el número de entradas de la máquina objetivo cuando se produce una

composición de máquinas. Al componer dos máquinas, la máquina resultante no sabía qué entradas pertenecían a cada máquina, con esta variable se tiene ese número para aplicarlo, por ejemplo, en el cálculo de la *transparencia*.

4.3 Clase Transición

Esta clase sirve para identificar una transición de la máquina de transiciones etiquetadas. Se detallan sus propiedades a continuación:

1. EstadoInicial
   Es un atributo que identifica al estado inicial de una transición.

2. Entrada
   Éste indica la entrada mediante la cual se pasa de un estado a otro en la transición.

3. EstadoSiguiente
   Es un atributo de tipo entero que se corresponde con el estado al que se mueve una transición.

4.4 Clase EstadoProbabilidad

Esta clase es una auxiliar utilizada en varios sitios de manera diferente, según la necesidad, que tiene las siguientes propiedades:

1. Nombre
   Es un atributo de tipo entero.

2. Probabilidad
   Es otro atributo que tiene tipo entero.

Capítulo 5: Usos de la aplicación

5.1 Carga de máquinas desde fichero

Las máquinas se cargan desde fichero bajo una estructura determinada. Dicha estructura se basa en la distinción por líneas. Cada línea del fichero representa una parte fundamental de la máquina.
A continuación se analiza cada una de las líneas del fichero que contiene la máquina asociada:

1. Línea 1
   Con esta línea se recogen los nombres de los estados de la máquina separados por un espacio en blanco. Por simplificación, se asume que el nombre de los estados es de tipo entero.

2. Línea 2
   Alberga las entradas de la máquina. Se produce de la misma forma que en la línea anterior, separadas por espacios en blanco.

3. Línea 3
   Si la línea 2 acogía a las entradas de la máquina, ésta tendrá las salidas de la misma.

4. Línea 4
   Utilizada para delimitar cuál será el estado inicial de la máquina. Evidentemente debe ser un entero y, pertenecer al conjunto de estados de la máquina descritos en la línea 1. En este caso va a ser el 1 por simplicidad.

5. Línea 5 – Línea n
   Comenzando en la línea 5 y hasta el final del fichero, lo que se encuentra son las transiciones entre estados de la máquina. La estructura de las líneas es la siguiente:

   EstadoInicial Entrada/Salida/# EstadoFinal.

   Evidentemente, las entradas/salidas, deben pertenecer a los conjuntos de entradas y salidas descritas en las líneas 2 y 3 y ya recogidas. Así mismo, los estados inicial y final deben pertenecer al conjunto de estados de la línea 1. Parece obvio, pero las transiciones deben permitir que el grafo de los estados quede conexo para que la máquina funcione de manera normal. Por tanto, cualquier estado puede no tener transiciones de salida, menos el estado que se haya definido como estado inicial (línea 4).

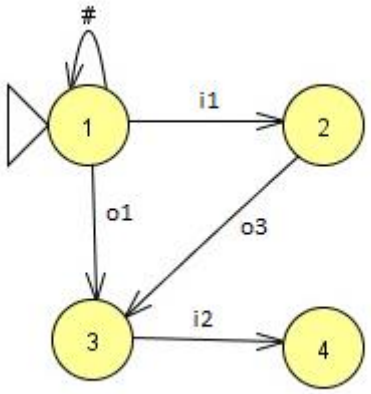
   Como se puede comprobar, existe la opción de transitar desde un estado inicial a otro final con el carácter #, es el símbolo escogido para transitar de un estado a otro sin consumir entrada o salida (transición vacía). Es conocida como transición τ.

Ejemplo de un fichero de carga:

```
1 2 3 4
i1 i2
o1 o2 o3
1
1 i1 2
1 o1 3
1 # 1
```

2 o3 3
3 i2 4

Ésta es la máquina resultante:



5.2 Realización de un estudio completo

Ya han sido analizados los métodos para realizar estudios, por separado, con respecto a una máquina concreta (complejidad, transparencia, observabilidad y controlabilidad). Ahora vamos a realizar un análisis profundo sobre dos máquinas cualesquiera. Como ha sido tratado anteriormente, el estudio consiste en tratar de comprender cómo influyen las métricas descritas en la detección de errores típicos de programación. Para ello, se necesita modificar la máquina objetivo, generando *mutantes* que se comporten de manera distinta al original.

Existen tres tipos de estudios en la aplicación, en función de las técnicas usadas para la generación del camino en la máquina original y, según el estudio del mutante. Estos estudios corresponden con los pasos realizados en nuestro estudio. Se permite la realización de los estudios 1, 2 y 3 sobre una máquina envoltorio y otra objetivo, pero también sobre una carpeta repleta de ficheros máquina, o una carpeta organizada según la generación de máquinas explicada en puntos anteriores.

1. Estudio 1
   Utiliza el método del *Random Walk* para la generación del camino en la máquina mutante. Para la comprobación sobre la original se hace uso del *Método de* existencia, descrito en la sección 5.3 en el apartado 2.

2. Estudio 2
   Genera el camino en la máquina mutante mediante el *Camino más largo*, descrito en la sección 3.3 en el apartado 2. La comprobación en la máquina objetivo original se consigue mediante el *Método de la existencia*.

3. Estudio 3
   El último tipo de estudios es el más sofisticado o, al menos, el que más terreno cubre de la máquina. Para conseguir el camino en la máquina mutante se utiliza

Testing Adaptativo, descrito en la sección 3.3 en el apartado 3. Una vez conseguido el camino, la comprobación en la máquina objetivo original se realiza mediante *Método de la existencia.*

Antes de pasar a ver en detalle qué hace de manera genérica un estudio, se procede a diseccionar cada uno de ellos, analizando qué casos cubre y cuáles no. Esta reflexión ayudará a comprender los resultados, interpretarlos de manera oportuna y plantear posibles avances en el futuro.

El estudio 1, introduce dos elementos que dan un salto de calidad a nuestro estudio, la longitud del camino y el *método de la existencia*. Vamos a quedarnos con el camino más largo de los generados en el estudio de la controlabilidad, pero hemos incluido un sistema de poda para evitar generar caminos innecesarios. Por tanto, mejoramos sustancialmente, pero no tanto como quisiéramos. El *método de la existencia* consiste en comprobar el camino producido de manera que exploramos los estados uno a uno según vamos avanzando. De esta forma sabemos con seguridad si se puede dar el camino encontrado o no en la máquina original. Este cambio es definitivo en nuestra manera de comprobar los caminos, va a resultar una ayuda vital para comprender qué está pasando en el estudio.

El estudio 2 es una ligera variación del estudio anterior. Una vez que es eliminada la poda en el proceso de generación de caminos, se puede escoger el camino más largo de todos los generados. Este proceso no resuelve el problema de la ramificación en el grafo, pero es un ligero avance, ya que permite abordar un amplio número de estados y aristas. La comprobación del camino generado se realiza mediante el *método de existencia*.

Por último, el estudio 3 aborda el problema del recubrimiento de aristas. Sobre el grafo que genera la máquina, se procede al cubrimiento de las aristas/transiciones del mismo. Para ello, se aprovechan los caminos generados para el cálculo de la controlabilidad, se escogen de mayor a menor número de transiciones siempre que el nuevo camino añada al menos una transición nueva. Se consigue, por tanto, el total cubrimiento de las transiciones de la máquina. Este método simula perfectamente la totalidad de posibilidades de la máquina, tanto entradas como salidas y transiciones vacías. Descubrir los fallos es sencillo puesto que el camino (conjunción de varios caminos desde el estado inicial a uno final, en simulación del *reset*), cubre la totalidad de los casos. La única forma de no detectar las mutaciones es generar un camino más corto, válido en la máquina original debido a que es un subcamino de un camino total. Estos caminos válidos pero reducidos pueden producirse tras la eliminación de un estado (ya que la misma implica la eliminación de las transiciones que llevan hasta él) o eliminación de una arista.

Podemos llevar a cabo estudios sobre un conjunto de máquinas, la aplicación leerá todos los ficheros máquina, los combinará si es posible, generará los mutantes y mostrará los resultados. Puede seleccionarse una carpeta que contenga sólo ficheros .txt o, por el contrario, una carpeta que tenga las máquinas organizadas por carpetas (tal y como se realiza en la clasificación que se puede realizar al generar máquinas

aleatorias). Para realizar el estudio sobre una carpeta que contenga un conjunto de ficheros máquina, se realizan los siguientes pasos:

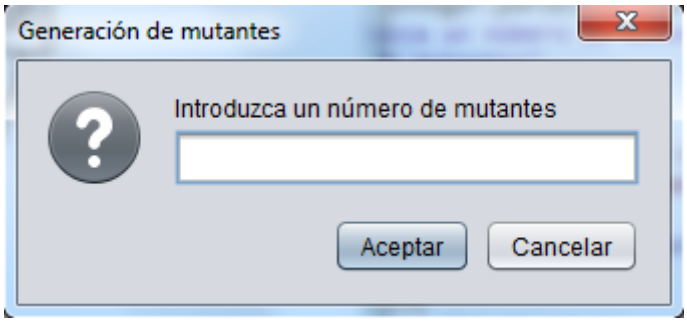
1. Escoger la opción *Estudio (1, 2 o 3)* dentro de *"Estudios->Estudios sobre varias máquinas*. Para el caso de estudio sobre máquinas transparentes se escoge la opción *Estudio (1, 2 o 3)* dentro de *"Estudios->Estudios sobre varias máquinas transparentes*.

   La diferencia entre una y otra es el método de selección de los ficheros de las carpetas seleccionadas.

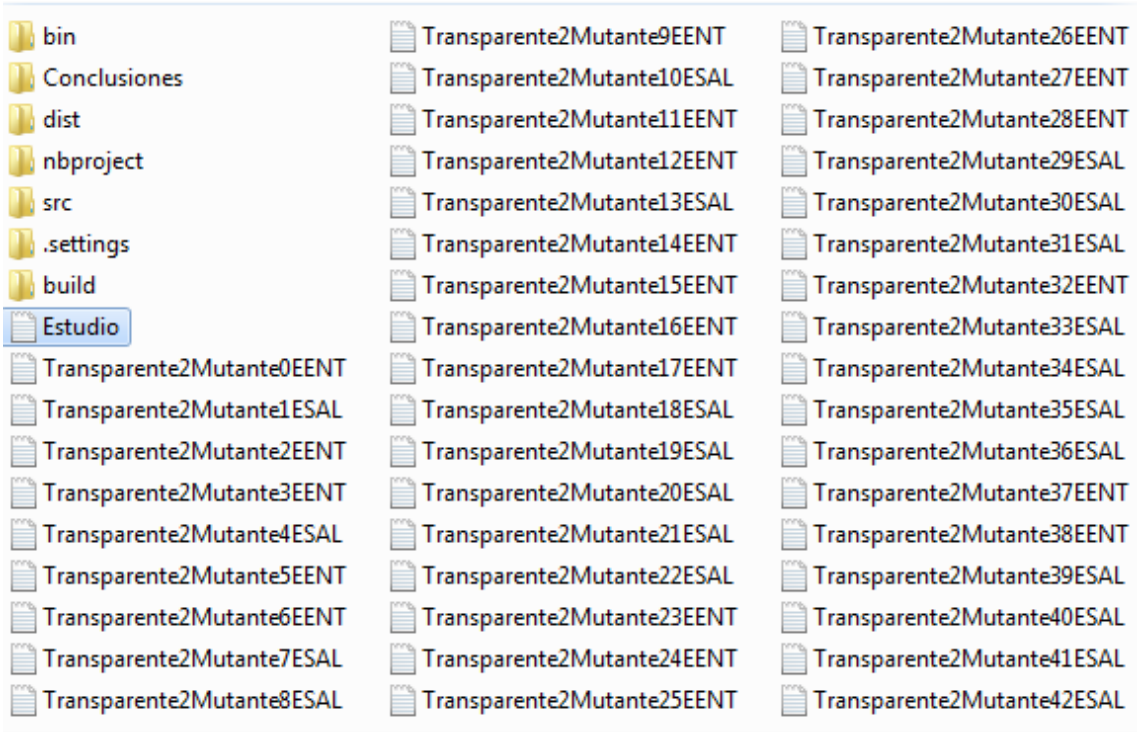
2. Escoger la carpeta que contiene los ficheros máquina

3. Estudio
   Una vez que se ha completado el punto anterior, la máquina compuesta está preparada para un estudio completo sobre la misma. Para ello, se procede a la generación de mutantes sobre las máquinas objetivo (o máquinas 2).

4. Pedir el número de mutantes
   La aplicación solicita al usuario que introduzca el número de mutantes que desea generar. Cabe destacar que dichos mutantes se generan a partir de las máquinas objetivo, ya que el posterior estudio se realiza mediante la composición de la máquinas envoltorio (máquina 1) procesadas a partir de la carpeta escogida y los mutantes generados a partir de la máquinas objetivo (máquina 2) de la misma carpeta.



5. Generar mutantes
   Se genera el número de mutantes solicitados por el usuario según lo explicado en la sección 3.2. De manera totalmente aleatoria, se realizan pequeñas modificaciones en la máquina objetivo. Dichas modificaciones serán: eliminación de un estado, eliminación de una transición, cambio de una entrada, cambio de una salida, añadir un estado o añadir una transición.

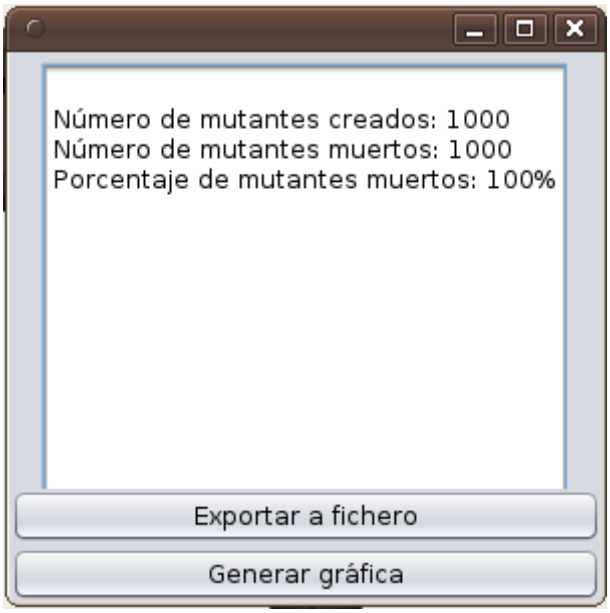


6. Generar camino
   Para discriminar si una máquina mutante debe ser considerada muerta o no, vamos a generar un camino (dependiendo de las características del estudio 2, 3 o 4; tal y como está descrito en la sección 5.2) sobre la máquina mutante. Una

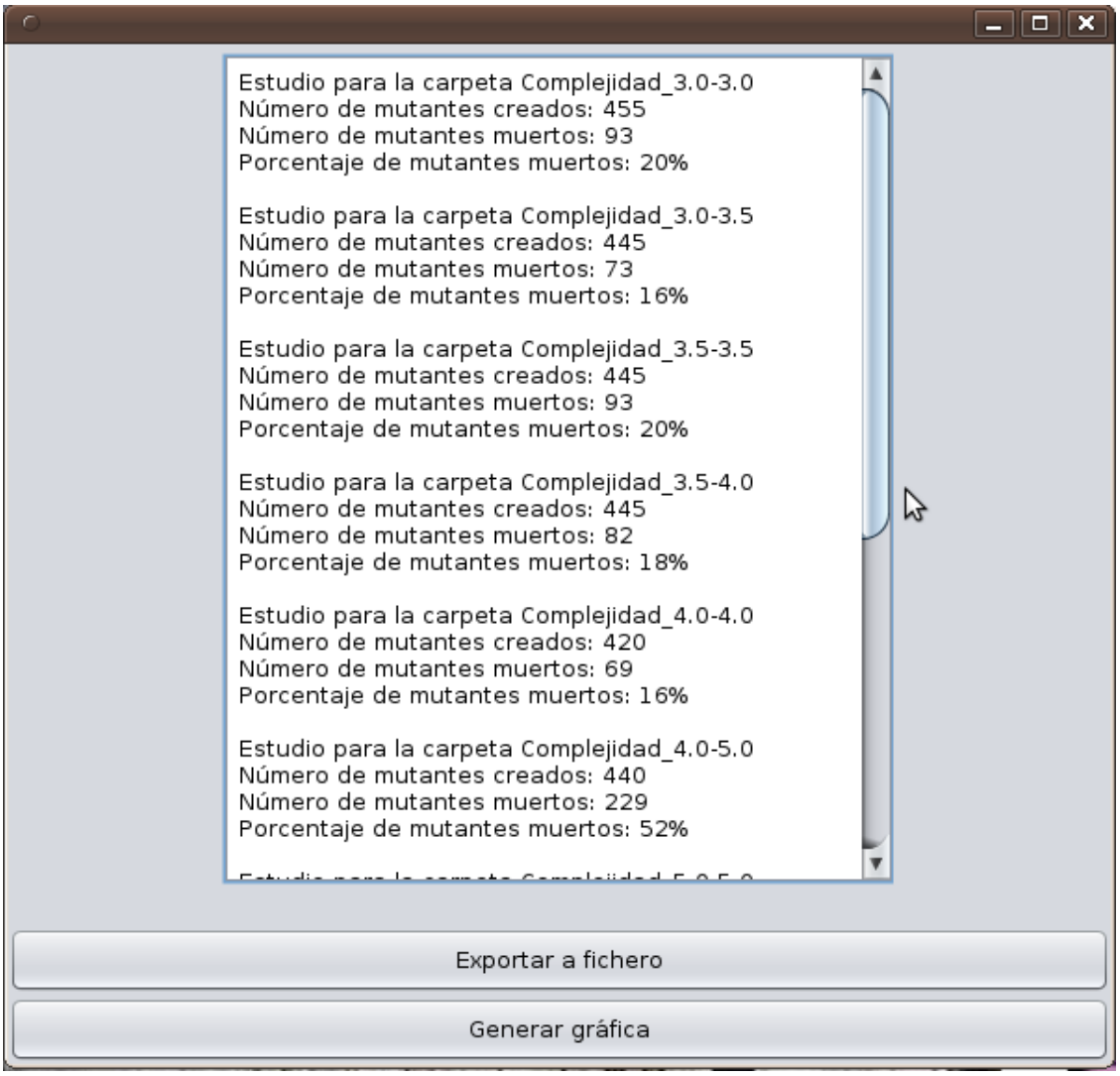
vez que tengamos el camino (conjunto de entradas y salidas de la máquina envoltorio), trataremos de comprobar si la máquina original puede conseguir el mismo conjunto de salidas para la máquina envoltorio usando el mismo conjunto de entradas para dicha máquina.

7. Comprobación mutante
   Conjunto de acciones encaminadas a decidir si matamos o no un mutante. Esta decisión se realiza conforme a dos métodos distintos (*método de tránsito* y *método de existencia*) que se explican en la siguiente sección (5.3).

8. Mostrar porcentajes mutantes muertos
   El siguiente punto del estudio completo consiste en mostrar una breve información sobre el número de mutantes creados, fallecidos y el porcentaje de mutantes muertos. Si la carpeta escogida no tiene carpetas con ficheros máquina dentro, será de la siguiente manera:

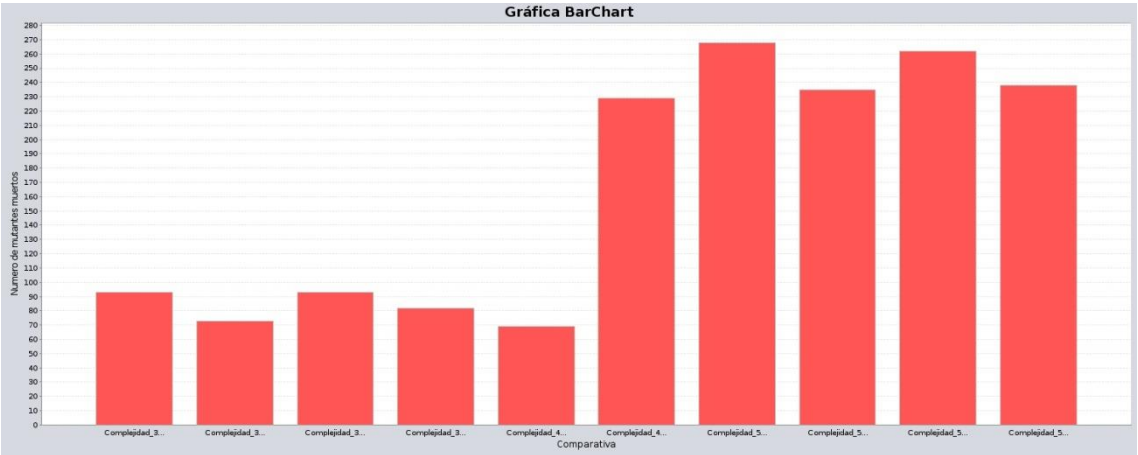


En caso contrario, se realizará una distinción de los resultados para cada carpeta analizada:



9. Exportar a fichero
   Se permite la exportación de los resultados a fichero .txt

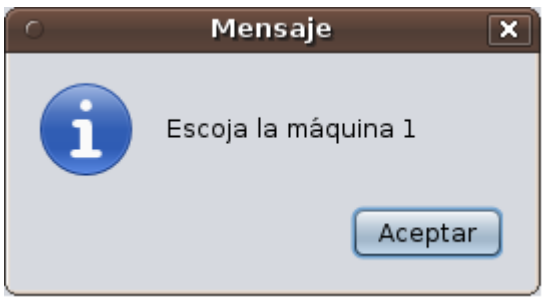
10. Generar gráfica
    Se permite la creación de una gráfica con los resultados alcanzados.



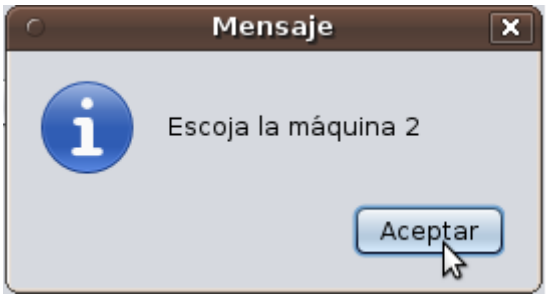
Para realizar este estudio sobre dos máquinas, la aplicación sigue los siguientes pasos:

1. Escoger la opción *Estudio (1, 2 o 3)* dentro de *"Estudios->Estudios sobre una máquina"*.

2. Escoger la máquina 1
   La aplicación nos ofrece la opción de escoger el fichero .txt que deseemos para cargar la máquina que hará las veces de envoltorio en el estudio posterior.

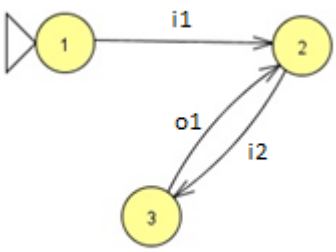


3. Escoger la máquina 2
   La aplicación nos ofrece la opción de escoger el fichero .txt que deseemos para cargar la máquina que hará las veces de objetivo en el estudio posterior.

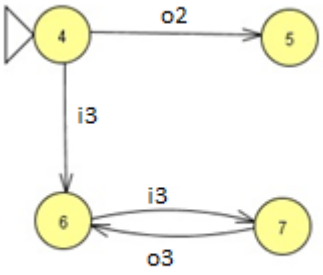


4. Generar máquina

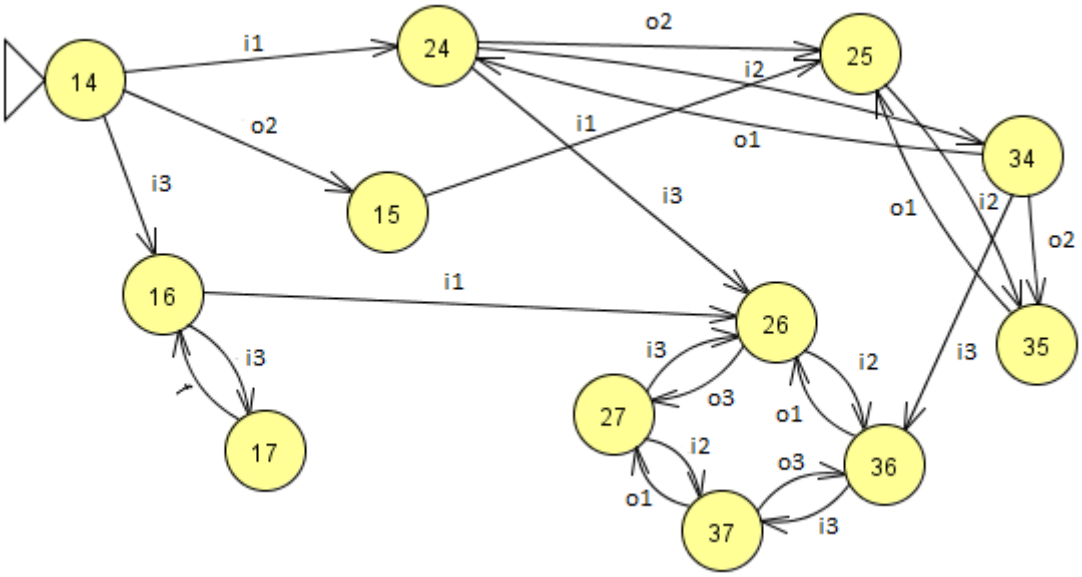
   La aplicación comprueba si puede generarse una máquina compuesta por las máquinas 1 y 2 que han sido escogidas por el usuario. En caso positivo, compone las máquinas de manera que, cada estado resultante, es la composición de un estado de la máquina 1 y otro de la máquina 2. La máquina compuesta está definida en la sección 2.5.



Máquina1

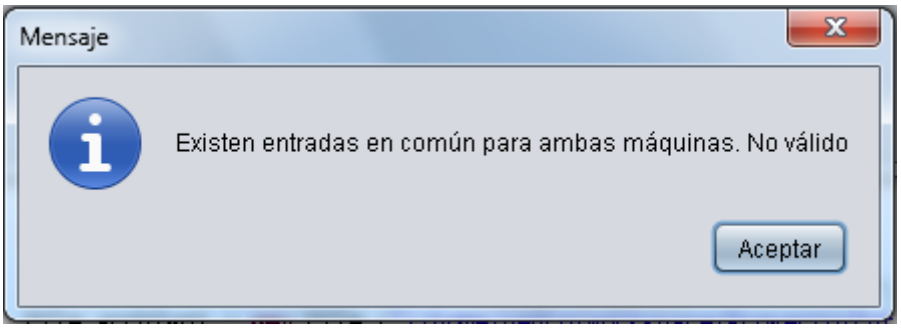


Máquina2



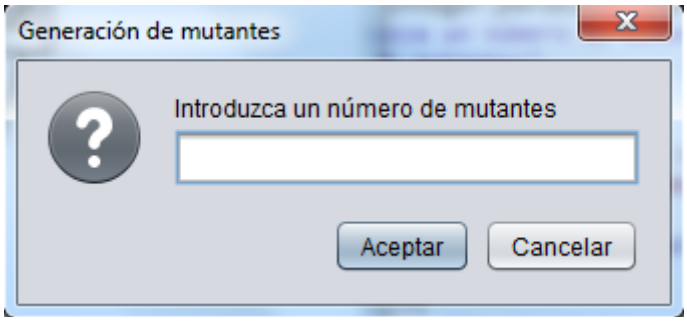
Máquina compuesta

En el otro caso, se muestra un mensaje detallando que no se ha podido hacer la composición de máquinas debido a que comparten entradas o salidas ambas.

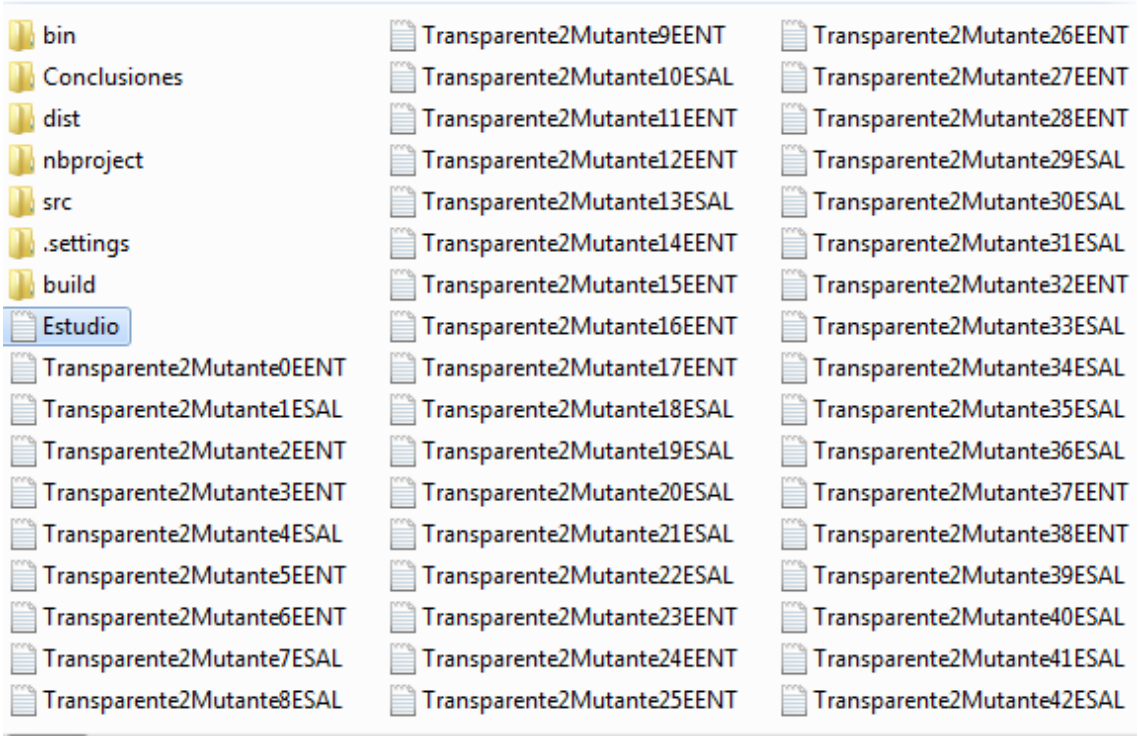


5. Estudio
   Una vez que se ha completado el punto anterior, la máquina compuesta está preparada para un estudio completo sobre la misma. Para ello, se procede a la generación de mutantes sobre la máquina objetivo (o máquina 2).

6. Pedir el número de mutantes
   La aplicación solicita al usuario que introduzca el número de mutantes que desea generar. Cabe destacar que dichos mutantes se generan a partir de la máquina objetivo, ya que el posterior estudio se realiza mediante la composición de la máquina envoltorio (máquina1) escogida por el usuario y los mutantes generados a partir de la máquina objetivo (máquina2).



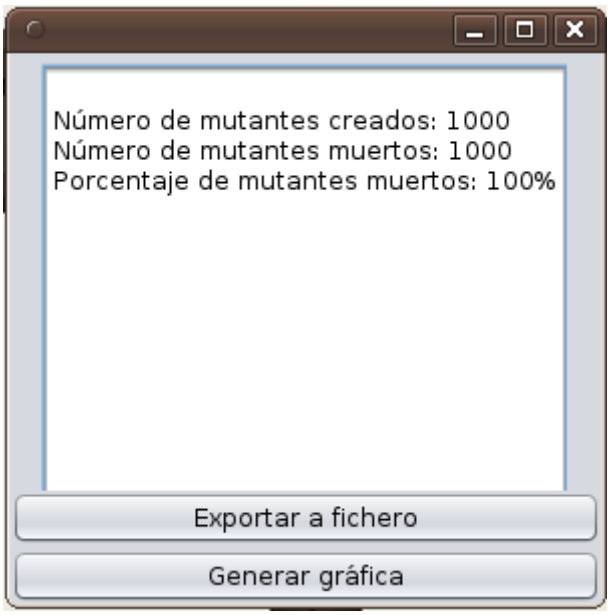
7. Generar mutantes
   Se genera el número de mutantes solicitados por el usuario según lo explicado en la sección 3.2. De manera totalmente aleatoria, se realizan pequeñas modificaciones en la máquina objetivo. Dichas modificaciones serán: eliminación de un estado, eliminación de una transición, cambio de una entrada, cambio de una salida, añadir un estado o añadir una transición.



8. Generar camino
   Para discriminar si una máquina mutante debe ser considerada muerta o no, vamos a generar un camino (dependiendo de las características del estudio 2, 3 o 4; tal y como está descrito en la sección 5.2) sobre la máquina mutante. Una vez que tengamos el camino (conjunto de entradas y salidas de la máquina envoltorio), trataremos de comprobar si la máquina original puede conseguir el mismo conjunto de salidas para la máquina envoltorio usando el mismo conjunto de entradas para dicha máquina.

9. Comprobación mutante
   Conjunto de acciones encaminadas a decidir si matamos o no un mutante. Esta decisión se realiza conforme a dos métodos distintos (*método de tránsito* y *método de existencia*) que se explican en la siguiente sección (5.3).

10. Mostrar porcentajes mutantes muertos
    El siguiente punto del estudio completo consiste en mostrar una breve información sobre el número de mutantes creados, fallecidos y el porcentaje de mutantes muertos.



11. Exportar a fichero
    Se permite la exportación de los resultados a fichero .txt.

12. Generar gráfica
    Se permite la creación de una gráfica con los resultados alcanzados.

5.3 Métodos de comprobación de mutantes

Una vez que se ha generado un camino (lista de entradas y salidas sobre la máquina mutante), tenemos que comprobarlo sobre la máquina original para decidir si matarlo o no. Al igual que existen dos métodos de generación de un camino, *random walk* que hace transitar a la máquina mutante quedándose con las entradas y salidas de la máquina envoltorio, y el *testing adaptativo* que genera una lista de entradas y salidas de la máquina mutante; para la comprobación existen dos formas de evaluación:

1. Método de tránsito
   Consiste en hacer la máquina original a partir de una lista de entradas generadas por el random walk. La máquina original intenta avanzar de un

estado a otro, si existe una transición que consuma el primer elemento de la lista de entradas, lo consume y transita, en caso contrario transita aleatoriamente con salida o transición vacía. Cuando transita con una salida, ésta se guarda (si es una salida de la máquina envoltorio).
Tras terminar, la máquina habrá llegado a un estado y terminará si no puede transitar o se han terminado las entradas de la lista inicial. Por tanto, habremos obtenido una lista con las salidas que han sido producidas.
Como habíamos obtenido una lista de salidas de la máquina mutante tras su tránsito aleatorio, comparamos esta lista con la que acabamos de obtener del mutante. Si no es igual, contamos una muerte. Pero sería injusto decir que el mutante ha muerto, ya que podemos haber transitado por un camino que no era el de la máquina original, así que decidimos repetir este proceso un número fijo de veces, en este caso cien. Tras repetir el proceso cien veces para el mismo mutante, si ha muerto cincuenta y una o más veces, entonces contamos al mutante como muerto y lo añadimos a la lista de mutantes muertos.

2. Método de existencia
   Este método se aplica al testing adaptativo. Consiste en comprobar si una secuencia de entradas y salidas de la máquina mutante, generada mediante testing adaptativo, es posible en la máquina original.
   Para esta forma de evaluación, hacemos transitar a la máquina de forma inteligente de todas las formas posibles, consumiendo elementos de la lista. Si existe una forma de encontrar ese camino, lo encontrará. En caso de no encontrar el mismo camino en el mutante, muere.

Capítulo 6: Conclusiones de los resultados experimentales

Las pruebas que dan paso a la conclusión de la investigación han sido realizadas sobre un número determinado de máquinas distintas generadas en función de la métrica correspondiente (observabilidad, controlabilidad, complejidad o transparencia). Sobre la base de las máquinas generadas, ha sido generado un número determinado de mutantes. Cabe destacar que, previamente a desarrollar los estudios que se muestran a continuación, se dedicó bastante tiempo a la comparación de los diferentes modelos de estudio que ofrece la aplicación. Una vez conseguida una batería de ejemplos considerable, se compararon entre si los modelos de estudio. Como conclusión se decidió optar por el estudio 3 como método para la realización de los experimentos finales. La decisión estuvo fundamentada en la versatilidad y mayor cercanía a la realidad que ofrecía dicho estudio. Al final, los datos recogidos corresponden al porcentaje de mutantes muertos. Estos cálculos serán de ayuda para completar el análisis de la investigación.

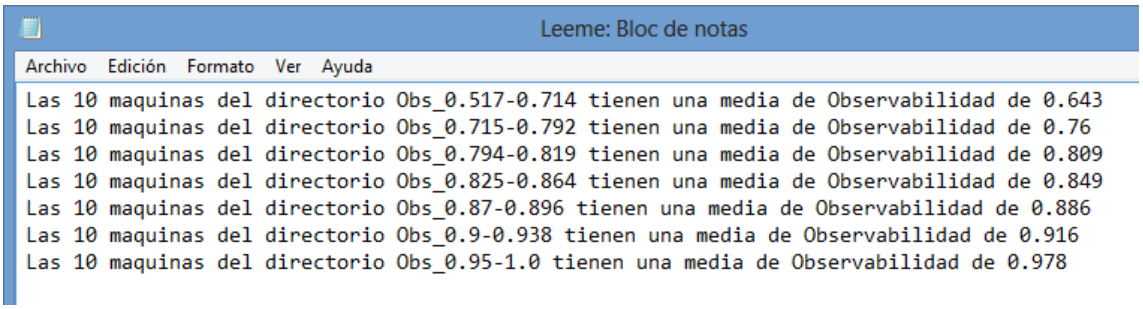
Observabilidad

La métrica a estudiar es la observabilidad. A continuación se va a proceder a explicar un ejemplo representativo para dicho estudio. Para ello se han creado 700 máquinas aleatorias con las siguientes características:

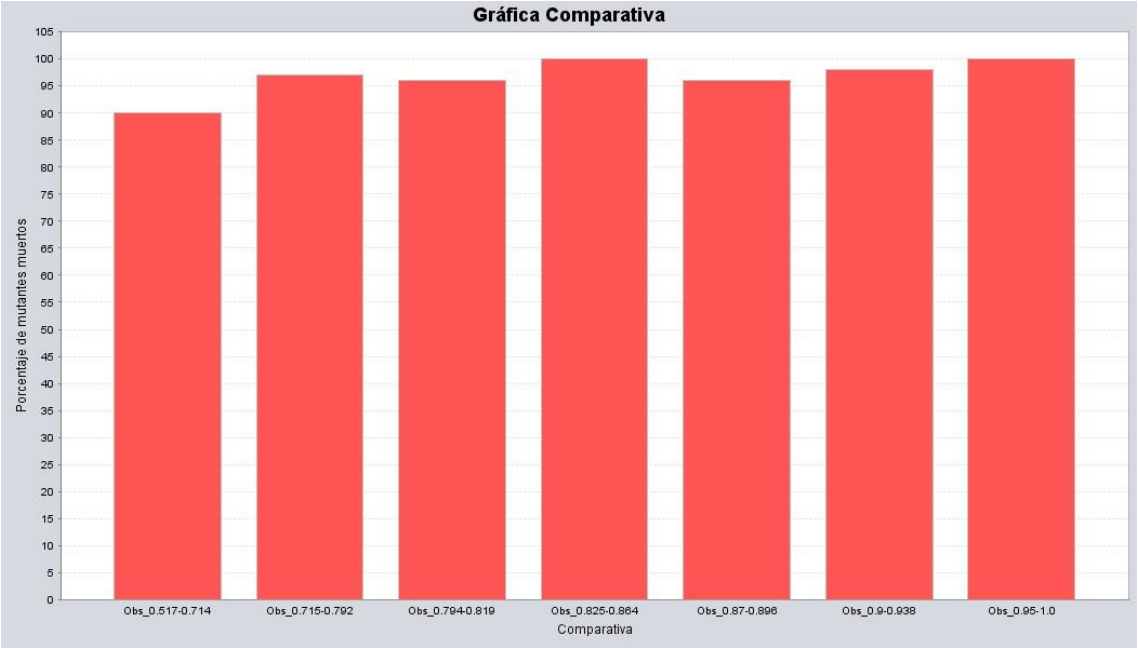
1. Número mínimo de estados: 4.
2. Número máximo de estados: 10.
3. Número mínimo de entradas: 1.
4. Número máximo de entradas: 1.
5. Número mínimo de salidas: 1.
6. Número máximo de salidas: 3.
7. Número mínimo de transiciones por estado: 5.
8. Número máximo de transiciones por estado: 6.
9. Porcentaje de entradas: 5.
10. Porcentaje de salidas: 80.
11. Porcentaje de lambdas: 15.

Las características realmente relevantes para este tipo de estudio son las concernientes al número de salidas, número de transiciones y los distintos porcentajes como vimos en el punto 2.7, el número de estados y de entradas no nos va a modificar a priori el valor de observabilidad de las máquinas.

Una vez generadas, se procedió a clasificarlas en torno a la métrica objeto de estudio y en 7 grupos. Se obtuvieron 7 carpetas, cada una con 100 máquinas donde los valores medios de observabilidad se muestran a continuación.

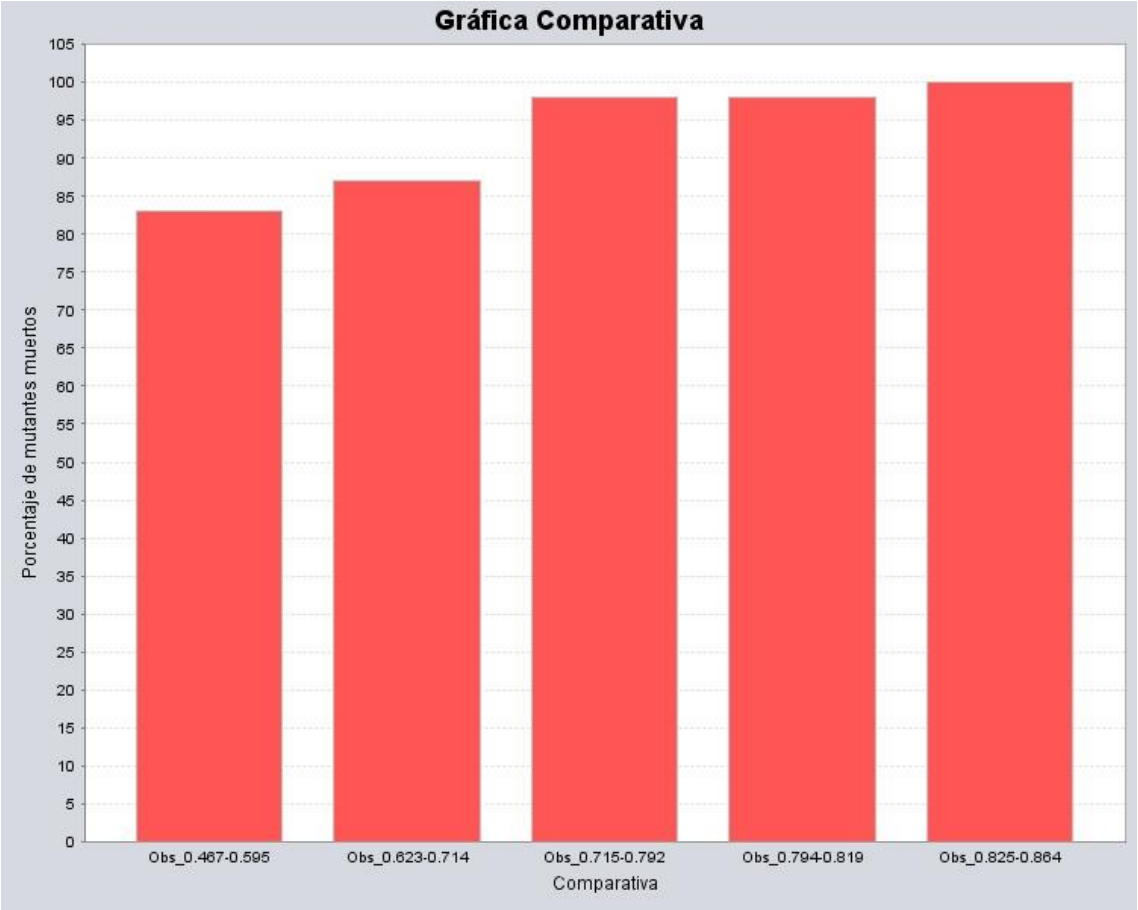


Se generaron 10.000 mutantes por máquina y se procedió a analizarlos. A continuación se muestra la gráfica resultante de la ejecución.



A simple vista se puede apreciar que la gráfica tiene un elevado valor de mutantes muertos. Lo que cabía esperar a priori es que según se incrementa la observabilidad de las máquinas, se incrementaría el porcentaje de mutantes muertos. Hay que tener en cuenta que el generador de máquinas aleatorias no es capaz de generar máquinas con valores de observabilidad bajos, debido a que este tipo de máquinas tienen una estructura muy determinada en sus transiciones. Es este ejemplo se puede observar que las máquinas con valores más bajos (las del primer grupo) tienen una media de observabilidad de 0,643.

Se procedió a crear máquinas manualmente con valores de observabilidad más bajos y a modificar los grupos del estudio anterior, integrando este tipo de máquinas o sustituyéndolas por algunas de las otras según procedía. Una vez clasificadas, volvíamos a ejecutar el estudio obteniendo el siguiente resultado.



Como se puede apreciar la gráfica tiene una tendencia ascendente, es decir, para valores de observabilidad menores se obtiene un porcentaje menor de mutantes muertos y este va creciendo conforme crece la observabilidad en las máquinas. La gráfica resultante del estudio aplicado sobre máquinas generadas aleatoriamente también muestra esta tendencia, pero al obtener valores de observabilidad mayores, es más difícil de apreciar.

En todo caso, se puede ver que los porcentajes de mutantes muertos son elevados para todos los grupos, analizando las máquinas que componen el grupo de menor valor de observabilidad, se puede observar que se pueden aplicar los siguientes tipos de mutación:

1. Eliminar un estado.
2. Eliminar una transición.
3. Añadir un estado.
4. Añadir una transición.
5. Sustituir un elemento de transición por una entrada.
6. Sustituir un elemento de transición por una salida.

Estudiando la estructura de este tipo de máquinas se observa que el mutante resultante de eliminar un estado, es posible que se produzca, aunque no en la

totalidad de las máquinas. El número de estados de la máquina no influye en los niveles de observabilidad de las máquinas, ya que al final el valor de observabilidad se calcula como la media de observabilidad de cada estado y este valor no está influido por el número de estados, sino por el número de transiciones y su forma. En conclusión, este tipo de mutantes se va a aplicar a este grupo de máquinas, pero únicamente si se selecciona un estado no necesario (su eliminación no provoca que algún otro estado quede inconexo). Analizando las máquinas observamos que en cada una, la aparición o no de este tipo de mutantes depende de su tamaño, pero si se puede afirmar que los mutantes de este tipo vivirán en este grupo, ya que los caminos resultantes en el mutante con un estado eliminado se encontrarán en la máquina original, por lo que no se detectará el error y el mutante vivirá.

Eliminar una transición en un mutante es posible y junto a los mutantes anteriores constituyen la principal causa de supervivencia de mutantes en este grupo de máquinas. Este grupo de máquinas tiene (por lo general) más transiciones que las máquinas de grupos más observables por lo que aquí es más probable seleccionar una transición que no se recorra a la hora de realizar el recubrimiento, y por tanto que el mutante sobreviva.

Los mutantes donde se añade un estado son siempre posibles y van a acabar muertos, ya que el nuevo estado será detectado a la hora de realizar el recubrimiento y dicho estado no se encontrará en la máquina original. Sin embargo, en este caso, los mutantes donde se añade una transición tienen alguna opción de sobrevivir, aunque no deja de ser reducida. Esta opción se basa en la probabilidad de seleccionar una transición que vuelva al estado de inicio y por tanto no sea detectada en el recubrimiento, pero esta opción es bastante remota dadas las características de las máquinas. En otro caso, este tipo de mutantes también acabará muerto.

Los mutantes de sustitución de un elemento de transición por una entrada o una salida, van a provocar en la mayoría de los casos muerte del mutante pero al tener este tipo de máquinas más transiciones que las máquinas más observables, es más posible que la sustitución se realice sobre una transición no tomada a la hora de realizar el recubrimiento y, por tanto, que el mutante sobreviva.

En conclusión, estudiando las máquinas tenemos un número moderado de muertes de mutantes pero hay más probabilidad de supervivencia que en grupos donde las máquinas sean más observables.

En grupos de observabilidad más elevada, los mutantes que ven modificado su porcentaje de muertes son:

1. Eliminar una transición.
2. Sustituir un elemento de transición por una entrada.
3. Sustituir un elemento de transición por una salida.

Cuando eliminamos una transición en un mutante con mayor observabilidad, al tener este tipo de máquinas un número más reducido de transiciones, es más probable que la transición eliminada forme parte del grupo de transiciones necesarias a la hora de realizar el recubrimiento y, por tanto, de ser detectada y su consecuente muerte.

Los mutantes resultado de sustituir un elemento de entrada por una entrada o una salida tienen mayor posibilidad de morir, por el mismo motivo que el grupo de mutantes descrito anteriormente. Al tener menos transiciones es más fácil que el cambio sea detectado.

En conclusión, las máquinas menos observables, donde se ha producido un error existe una menor facilidad para detectarlo, y según aumenta la observabilidad de las máquinas esta facilidad va aumentando. En una máquina observable, al tener un número menor de transiciones, es más sencillo conocer en qué punto de depuración nos encontramos y más difícil en máquinas menos observables. En todo caso, este tipo de máquinas, por sus características, tienen un alto porcentaje de detección de mutantes. Relacionando este tipo de máquinas con código, observamos:

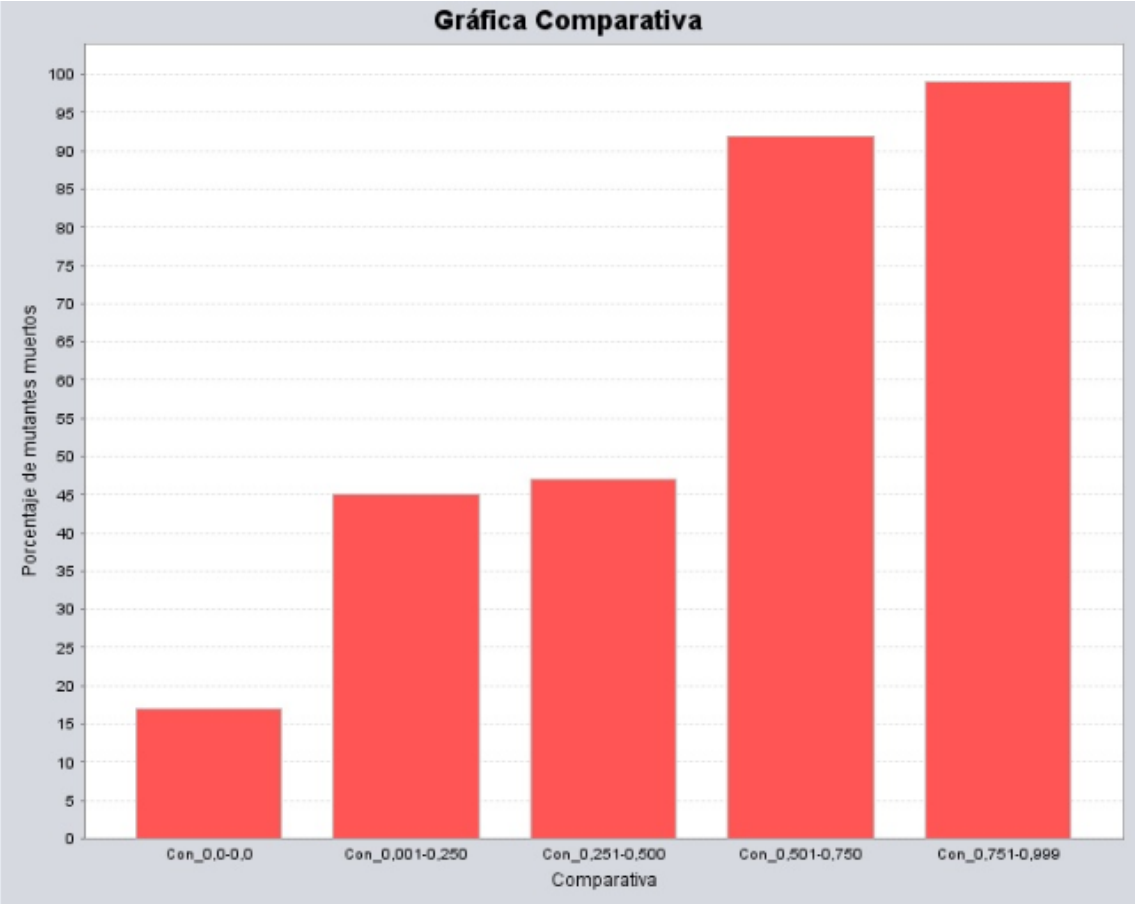
El código menos observable, tiene más transiciones y ellas se realizan mediante más salidas las cuales se suelen repetir. Es decir, en un determinado punto del programa existen más bifurcaciones posibles que producen salidas que se repiten en distintas bifurcaciones, por lo que es más difícil saber en qué punto del programa se encuentra y, por tanto, detectar un posible fallo de programación.

El código más observable, tiene menos transiciones y ellas se realizan mediante más entradas que salidas, y cuando son salidas no suelen estar repetidas por lo que es más fácil identificar el punto actual y, por tanto, detectar un posible fallo en el código.

Controlabilidad

Se han creado 5 intervalos de máquinas que están en el rango de valor de controlabilidad de 0-1. Cada intervalo contiene 100 máquinas. Un intervalo tiene como nombre Con_0,751-0,999, que quiere decir que las máquinas incluidas en él tienen un valor de controlabilidad entre 0,751 y 0,999. Lo mismo sucede para el resto de intervalos, con otros valores.

Para la realización del estudio se han creado 1000 mutantes para cada máquina. Esta es la gráfica resultante de la ejecución del ejemplo:



Se puede observar en la gráfica que, a mayor valor de controlabilidad, mayor número de mutantes muertos. Desde un punto de vista de programación es totalmente lógico ya que un código controlable permite saber en cada instante en qué punto se está, algo que facilita enormemente el testeo. Ahora bien, ¿tiene sentido desde el punto de vista teórico? La realidad es que si.

Una máquina con controlabilidad baja es aquella que tiene un elevado número de transiciones de salida o lambda para cada estado. Evidentemente, no sólo este tipo de máquinas provocan una controlabilidad baja. También aquellas máquinas que no tienen transiciones de entrada para ningún estado. Este último tipo de máquinas han sido las creadas. Ahora bien, ¿cómo explicar un porcentaje tan bajo para el primer intervalo? La razón es que las máquinas tienen pocos estados, del orden de dos o tres estados, con un elevado número de transiciones, salidas o lambda. Por tanto, pueden producirse todos los tipos de transiciones, dándose el siguiente número de opciones:

1. Añadir un estado: Evidentemente genera la muerte del mutante siempre.
2. Añadir una transición: En teoría siempre produce la muerte del mutante. Existen casos de excepción, como cuando se genera un ciclo o se añade una transición nueva que es idéntica a una existente (la cobertura de aristas no genera caminos con la misma transición, si esta se repite). Como las

máquinas generadas tienen bastantes transiciones por estado, es bastante probable que al añadir se repita, evitando la muerte del mutante.

3. Modificar transición (entrada/salida): Ocurre lo mismo que en el punto anterior, si al modificar se cambia por una transición que ya existía para el mismo estado, entonces la cobertura no cogerá esa transición (que ya ha sido escogida con anterioridad), evitando la muerte del mutante.
4. Eliminar una transición: Nunca mata al mutante
5. Eliminar un estado: Nunca mata al mutante.

Por tanto, la probabilidad de muerte del mutante es bastante reducida, ya que tres de las cuatro mutaciones que matan a los mutantes muchas veces no van a hacerlo por la configuración de las máquinas.

A medida que aumenta la controlabilidad, se reducen el número de transiciones con salidas o lambdas en los estados. De esta manera, como se puede imaginar, comienza a aumentar la probabilidad de éxito (matando mutantes) de las mutaciones que añaden una transición o la modifican. La explicación es clara, a medida que hay menos transiciones de salidas o lambdas, existen menos opciones de que al añadir una transición o modificarla, se repita la transición. Así se puede observar que el porcentaje de mutantes se incrementa notablemente debido a este razonamiento.

Las máquinas que componen el último intervalo analizado son aquellas que tienen un reducido número de transiciones, generalmente una por estado, siendo éstas entradas de la máquina. Evidentemente, cuando las máquinas tienen una transición por estado, las mutaciones que eliminan estados o transiciones tienen menor probabilidad de aplicarse (dejarían estados huérfanos o inalcanzables). Además, como hay una o dos transiciones por estado, si se añade una transición, la probabilidad de que sea distinta de las que había es bastante elevada, por lo que se escogerá en la cobertura de aristas (matando al mutante).

Evidentemente, no son el único tipo de máquinas que podrían haberse generado. Quizá, desde el punto de vista real, habría tenido más sentido haber creado máquinas con un mayor número de estados y transiciones para el intervalo de menor controlabilidad. Por esto, hacer un análisis desde el punto de vista teórico, aplicado a la aplicación realizada, se antoja importante. ¿Qué habría cambiado si se hubiesen creado este tipo de máquinas? Simplemente habría subido el porcentaje de muertes en los mutantes. Aún así, la tendencia de la gráfica habría sido la misma. La explicación es sencilla, con un mayor número de estados y transiciones por estado, se permiten todos los tipos de mutaciones. Como hay más estados, añadir transiciones debe matar más, la explicación es que existe un mayor número de destinos posibles para la transición nueva, pudiendo así surgir una relación entre dos estados que antes no estaban conectados de manera directa. Como se puede suponer, la probabilidad de muerte aumenta. De igual manera se entiende con la modificación de transiciones

mataría más mutantes. De nuevo, la explicación radica en que no todos los estados tienen igual número de transiciones, si se modifican aquellas que parten de un estado con un reducido número, la probabilidad de que se repita una transición será reducida.

Como conclusión, se puede observar que, en las máquinas estudiadas para la controlabilidad, el grado de afectación que supone la estructura interna de la máquina, no tiene tanta relevancia como en las otras métricas.

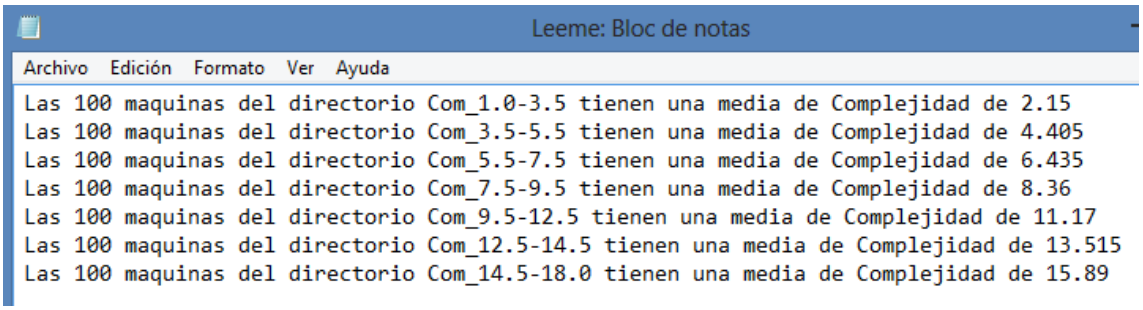
Complejidad

La siguiente métrica a estudiar es la complejidad. A continuación se va a proceder a explicar un ejemplo representativo para dicho estudio. Para ello se han creado 700 máquinas aleatorias con las siguientes características:

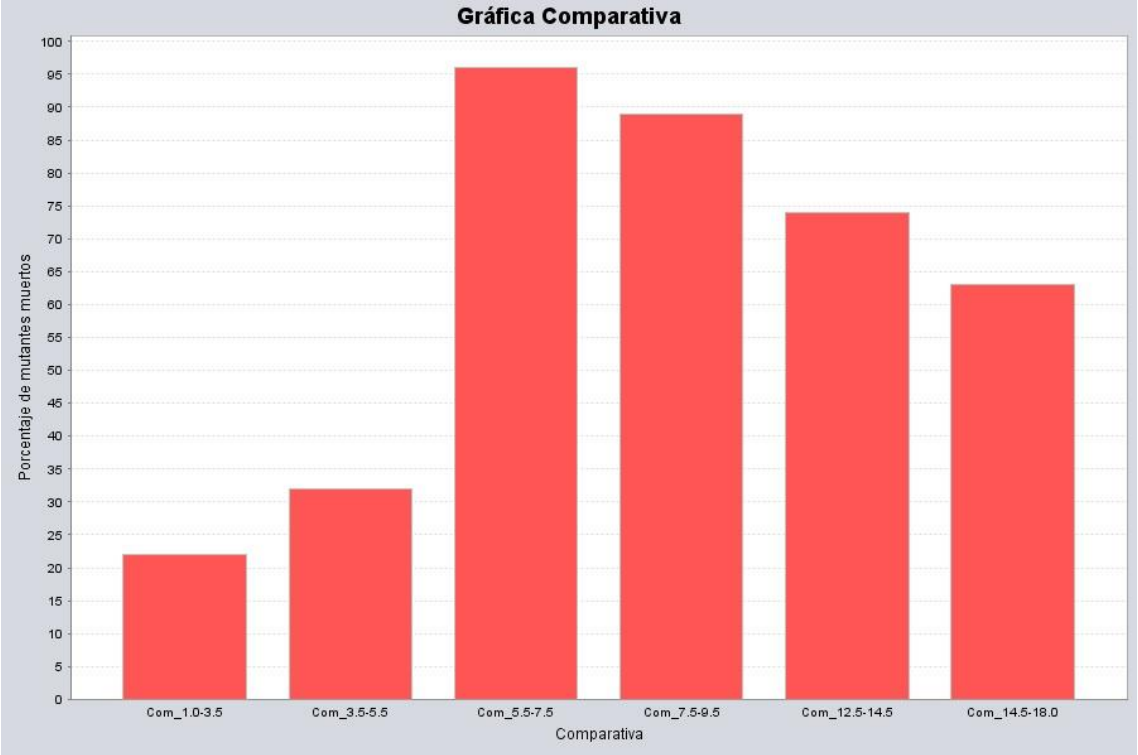
1. Número mínimo de estados: 1.
2. Número máximo de estados: 10.
3. Número mínimo de entradas: 3.
4. Número máximo de entradas: 5.
5. Número mínimo de salidas: 3.
6. Número máximo de salidas: 5.
7. Número mínimo de transiciones por estado: 1.
8. Número máximo de transiciones por estado: 3.
9. Porcentaje de entradas: 40.
10. Porcentaje de salidas: 40.
11. Porcentaje de lambdas: 20.

Las características realmente relevantes para este tipo de estudio son las concernientes al número de estados de las máquinas y al número de transiciones de las mismas, como vimos en el punto 2.9, el resto de valores a priori no nos van a condicionar nuestro estudio.

Una vez generadas, se procedió a clasificarlas en torno a la métrica objeto de estudio y en 7 grupos. Se obtuvieron 7 carpetas, cada una con 100 máquinas donde los valores medios de complejidad se muestran a continuación.



Se generaron 10.000 mutantes por máquina y se procedió a analizarlos. A continuación se muestra la gráfica resultante de la ejecución.

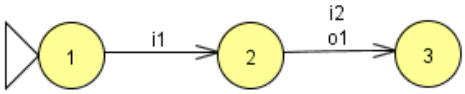


A simple vista se puede apreciar que la gráfica tiene una anomalía en las dos primeras carpetas. Este resultado, impactó bastante, pues lo esperado era que la gráfica tuviera una tendencia descendente desde un inicio, es decir, que las máquinas menos complejas fueran las que nos proporcionaran un porcentaje mayor de mutantes muertos. Este hecho impulsó el estudio de la estructura interna de las máquinas.

Se procede, por tanto, a analizar las máquinas de estos dos primeros grupos. Se apreció que se trataba de máquinas muy pequeñas, formadas por 1 o 4 estados y no más de 5 transiciones (recordamos que la complejidad se obtenía sumando los productos del número de estados por 0.5 y del número de transiciones totales por 0.5). Estas máquinas no se asemejaban a modelos reales (dimensión y forma) y debido a sus características se pudo observar que la mayoría tenía un patrón común: formaban un ciclo. Este hecho unido a las pequeñas dimensiones de las máquinas, permite obtener mutantes de tipos:

1. Eliminar un estado.
2. Eliminar una transición.
3. Añadir un estado.
4. Añadir una transición.
5. Sustituir un elemento de transición por una entrada.
6. Sustituir un elemento de transición por una salida.

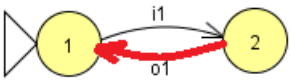
Se puede comenzar a analizar el primer tipo. Eliminar un estado, produce un mutante que tras el estudio no va a morir, ya que los caminos restantes tras eliminar el estado se van a poder hallar en el original. Sin embargo, este tipo de mutación va a ser muy poco frecuente por dos motivos: el estado inicial nunca puede ser eliminado y un estado intermedio necesario para alcanzar otro tampoco ya que dejaría estados inconexos. Estos dos motivos, unidos a que las máquinas están compuestas por un número tan reducido de estados dan como conclusión que esta mutación va a parecer en muy contadas ocasiones. Hay que tener en cuenta que, cuando no se puede eliminar un estado elegido al azar, se procede a elegir de nuevo al azar una nueva mutación a crear. Por tanto, en el siguiente ejemplo sólo habría un 33% de posibilidades de éxito para eliminar estado, ya que los estados 1 y 2 no pueden ser eliminados.



En cuanto a la eliminación de transición sabemos que dicha mutación no produce la muerte del mutante al poder encontrar los caminos restantes en la máquina original. Sin embargo se puede concluir, que se trata de una mutación que aparecerá muy poco frecuentemente debido a que la mayoría de las transiciones serán críticas y su eliminación provocaría la aparición de estados inconexos. En el ejemplo anterior se podrían eliminar cualquiera de las 2 transiciones que parten del estado 2. El proceso que se sigue una vez se selecciona este tipo de mutación es el siguiente: se escoge un estado y después una transición de ese estado, si no se puede eliminar, se busca otra transición de ese mismo estado, si en un número limitado de iteraciones no se ha conseguido eliminar, se procede a escoger otra mutación de nuevo al azar.

Añadir un estado se puede dar en toda máquina y supondrá la principal causa de muertes de mutantes dentro de esta carpeta, ya que al intentar alcanzarlo en la máquina original será imposible al no contar con él.

Añadir una transición también se puede dar en toda máquina sin embargo producirá un número reducido de muertes al contrario de lo que cabría esperar. Esto es debido al tamaño de las máquinas ya que al ser tan pequeñas, el recubrimiento no exigirá cubrir muchas de las transiciones añadidas, veámoslo con un ejemplo. A continuación, se puede apreciar que la transición 2 o1 1, no será recogida por el recubrimiento ya que partiendo desde el 1 y transitando mediante i1 alcanzaremos todos los estados.

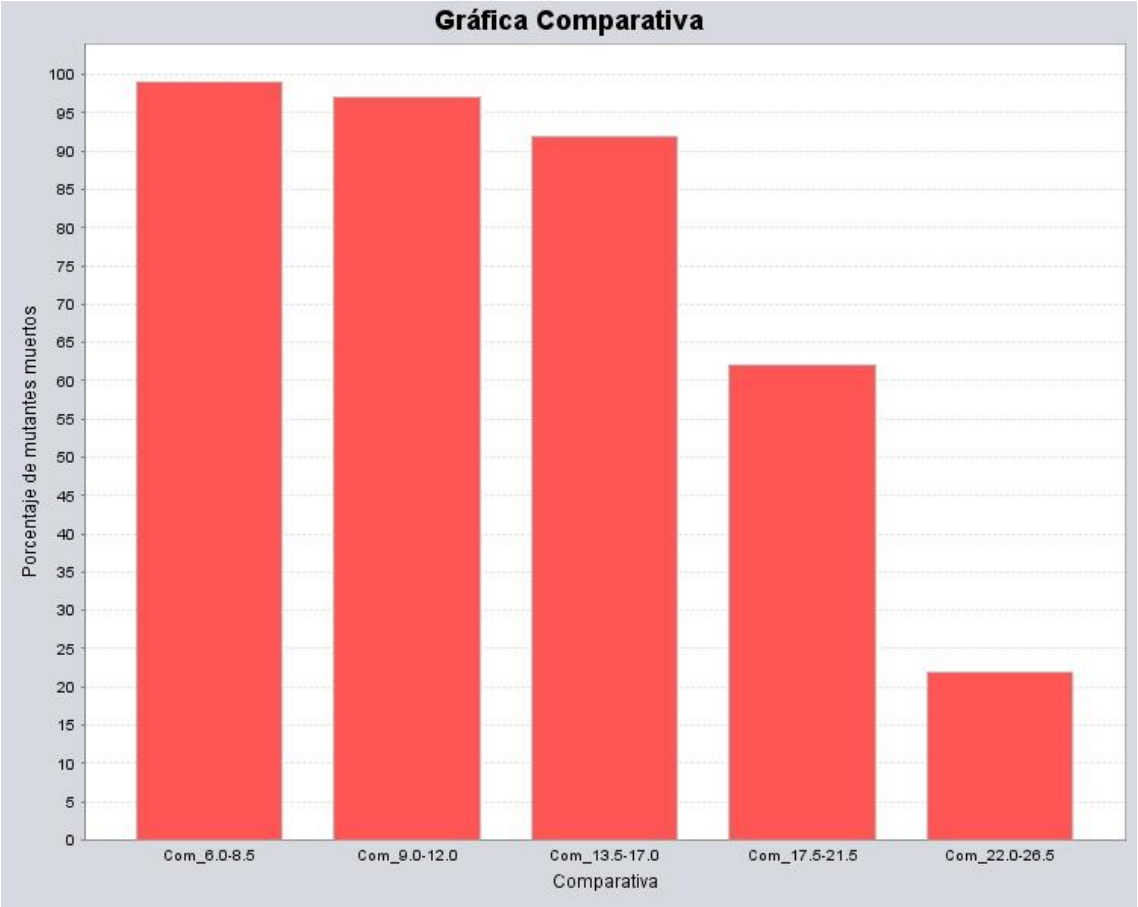


En cuanto a la sustitución de un elemento de transición por una entrada o una salida, producirá muertes en ocasiones limitadas (transiciones que impliquen el recubrimiento, como es el caso de 1 i1 2 en el ejemplo anterior). Estos tipos de mutación se pueden realizar en toda máquina y suponen la principal causa junto a la del punto anterior de concluir con mutantes vivos.

La conclusión que se puede obtener es que se trata de máquinas demasiado pequeñas, que no se ajustan a modelos reales y con la peculiaridad de los ciclos lo que hace obtener un porcentaje de mutantes muertos tan bajo.

Una vez detectado el error, se procedió a crear las máquinas implicadas de manera manual. En total se crearon 125 máquinas, clasificadas en torno a la métrica objetivo del estudio en 5 grupos de 25 máquinas cada uno.

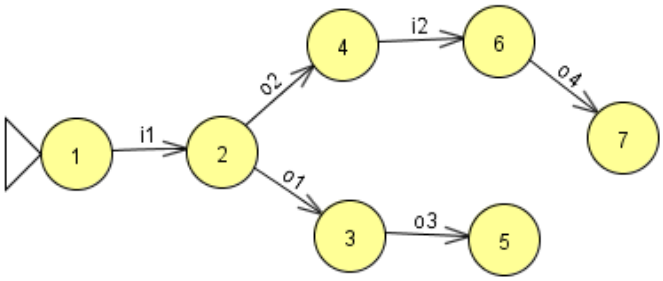
El resultado fue el esperado y se muestra a continuación en la siguiente gráfica obtenida.



Las máquinas del primer grupo, se crearon con un mínimo de 7 estados y las transiciones imprescindibles para que todos quedaran conexos, es decir, lo que se asemejaría más a un modelo real de una máquina no compleja. El resto de características se consideraron irrelevantes. Por las características de este tipo de máquinas, los posibles mutantes a generar fueron:

1. Eliminar un estado.
2. Añadir una transición.
3. Añadir un estado.
4. Sustituir un elemento de transición por una entrada.
5. Sustituir un elemento de transición por una salida.

Los mutantes que eliminaban una transición dejaron de ser posibles, al producir máquinas erróneas (estados inconexos). Este tipo de mutantes, al igual que el ejemplo siguiente, sabemos que no generan la muerte del mutante. Los mutantes donde se eliminaba el estado sí eran válidos, pero la probabilidad de que se seleccionaran fue muy reducida ya que sólo podrían eliminarse estados "finales", es decir, destinos de alguna transición y de los cuales no partiera otra transición. En la siguiente figura se puede apreciar que ninguna transición puede ser eliminada por lo descrito anteriormente y los únicos estados posibles de eliminar serían el 5 y el 7, por tanto una vez seleccionada dicha mutación, tendríamos un 28% de posibilidades de elegir uno de los estados nombrados. En caso de no haberlo seleccionado se elegiría otro tipo de mutación aleatoriamente.



En cuanto a los mutantes donde se añade una transición, se sabe que provocará la muerte del mutante en un número muy elevado de casos. El único caso donde ésta no se produciría sería si la nueva transición formara un ciclo (descrito anteriormente en las máquinas generadas aleatoriamente) pero sabemos que este caso es bastante remoto ya que al tener un número mayor de estados disponibles es más difícil que pueda producirse.

En cuanto al resto de mutantes se sabe que van a morir en el estudio. Añadir un estado, provocará siempre la muerte del mismo ya que dicho estado nos será alcanzable en el original, y sustituir un elemento por una entrada o una salida también ya que todas las transiciones de este tipo de máquinas son requeridas para realizar el

recubrimiento y se detectaría el cambio en la máquina original con respecto al mutante.

Los otros 4 grupos de máquinas se constituyeron incrementando lentamente el número de estados y transiciones de las mismas. Este aumento progresivo hace que vayan disminuyendo el número de mutantes muertos en grupos progresivos ya que cada vez encontramos más facilidad para aplicar mutantes de eliminación de estado y transición (los cuales sabemos que provocan la supervivencia del mutante en todos los casos).

El mutante que añade un estado sigue apareciendo y provoca la muerte del mutante y es la principal causa de muerte de mutantes en estos grupos.

Los otros 3 tipos de mutantes, según va aumentando el tamaño de las máquinas, provocan menos muertes. Ya que añadir una transición o sustituir un elemento de una transición por una entrada o una salida en máquinas cada vez más complejas favorece el factor de que la transición afectada no sea necesaria para el recubrimiento de la máquina y por tanto no detectemos el cambio en el mutante con respecto al original.

En conclusión, las máquinas menos complejas donde se ha producido un error, tenemos una mayor facilidad de detectarlo, y según aumenta la complejidad de las máquinas esta facilidad va disminuyendo. En una máquina compleja, al haber tantas transiciones, es muy difícil saber en qué punto de depuración se está, ya que meter una entrada es muy probable que lleve a varios estados, éstos quizá producen una misma salida aun tratándose de distintas transiciones y encontrándonos en estados distintos, por lo que es muy difícil saber en qué estado se encuentra. Este hecho dificulta mucho la labor del testeador sobre la máquina. Relacionando una máquina compleja (muchos estados y transiciones) con código, simularía código en el cual encontraríamos muchas bifurcaciones (transiciones de la máquina) que nos llevan a distintos puntos del programa donde ciertas variables tomarían determinados valores (estados de la máquina). Si el código es muy complejo y tiene muchos posibles valores para sus variables, es muy difícil determinar en qué punto del mismo se encuentra, y en definitiva, poder hallar posibles errores cometidos por parte del programador.

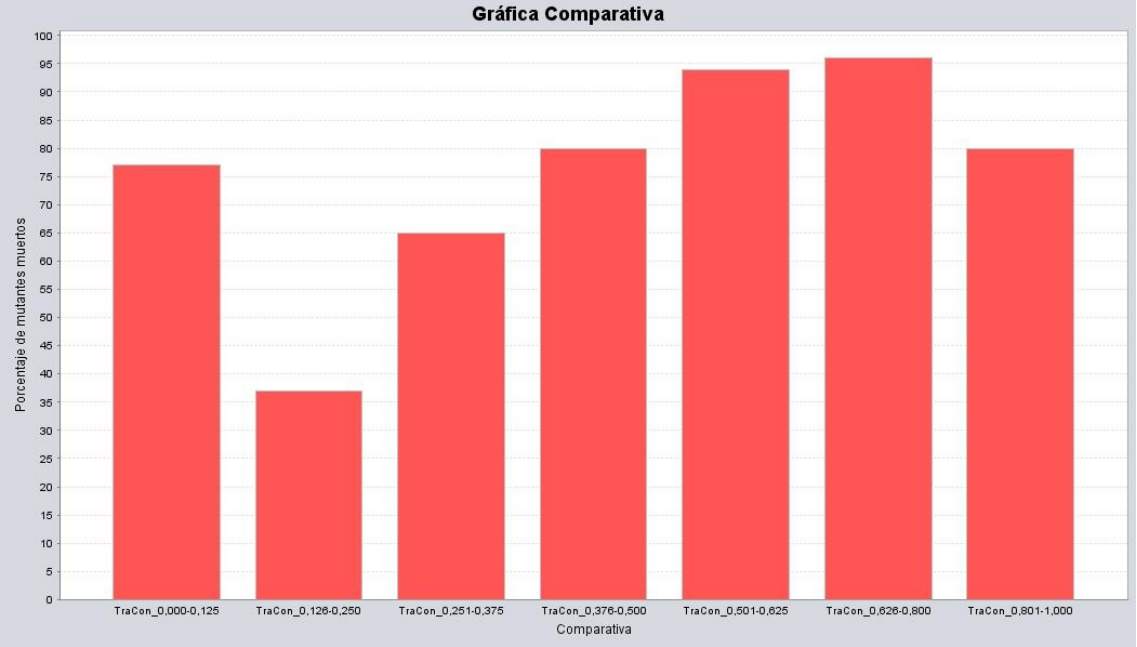
Transparencia

Debido a la necesidad de combinación de dos máquinas (máquina envoltorio y máquina objetivo), la creación de máquinas transparentes (bien sea de control o de observación) ha de realizarse de manera manual. Por tanto, la clasificación en carpetas también. Creando una carpeta que englobe a las carpetas que se creen como intervalos de estudio (donde han de clasificarse las máquinas de manera que queden consecutivas la máquina envoltorio con su correspondiente máquina objetivo), se permite un estudio propio de las máquinas transparentes, que consistirá en ir

cogiendo de manera consecutiva las máquinas de la carpeta, esto es, la primera con la segunda, la tercera con la cuarta y así sucesivamente, correspondiendo así a la máquina envoltorio y su objetivo.

Primero se analizará la transparencia de control. Para ello se han hecho varios estudios, recogiendo todos los datos y analizando el comportamiento, quedando dos ejemplos como una representación estable del conjunto de los casos.

Para la primera ejecución se han creado siete intervalos de valores dentro del rango 0-1. Cada rango tiene veinte máquinas (veinte máquinas envoltorio y veinte objetivo que conforman veinte máquinas al componerlas). Para cada máquina se han creado 10000 mutantes. La gráfica siguiente muestra el porcentaje de mutantes muertos dentro de cada intervalo.



Como puede observarse, la gráfica tiene una tendencia ascendente. No obstante posee dos intervalos que no cumplen dicha tendencia, más concretamente los intervalos inicial y final. Por ahora se aparca la explicación de este punto, algo que se retomará al final de la explicación de las máquinas según la transparencia de control.

Lo primero que se debe tener en cuenta es cómo debe ser una máquina para tener transparencia de control cero o uno. Para que una máquina tenga transparencia de control cero (o un resultado muy próximo a cero) pueden darse tres casos:

1. Máquina envoltorio no tiene transiciones con entradas
2. Máquina objetivo no tiene transiciones con entradas
3. Máquina objetivo tiene varias transiciones con distintas entradas para un mismo estado.

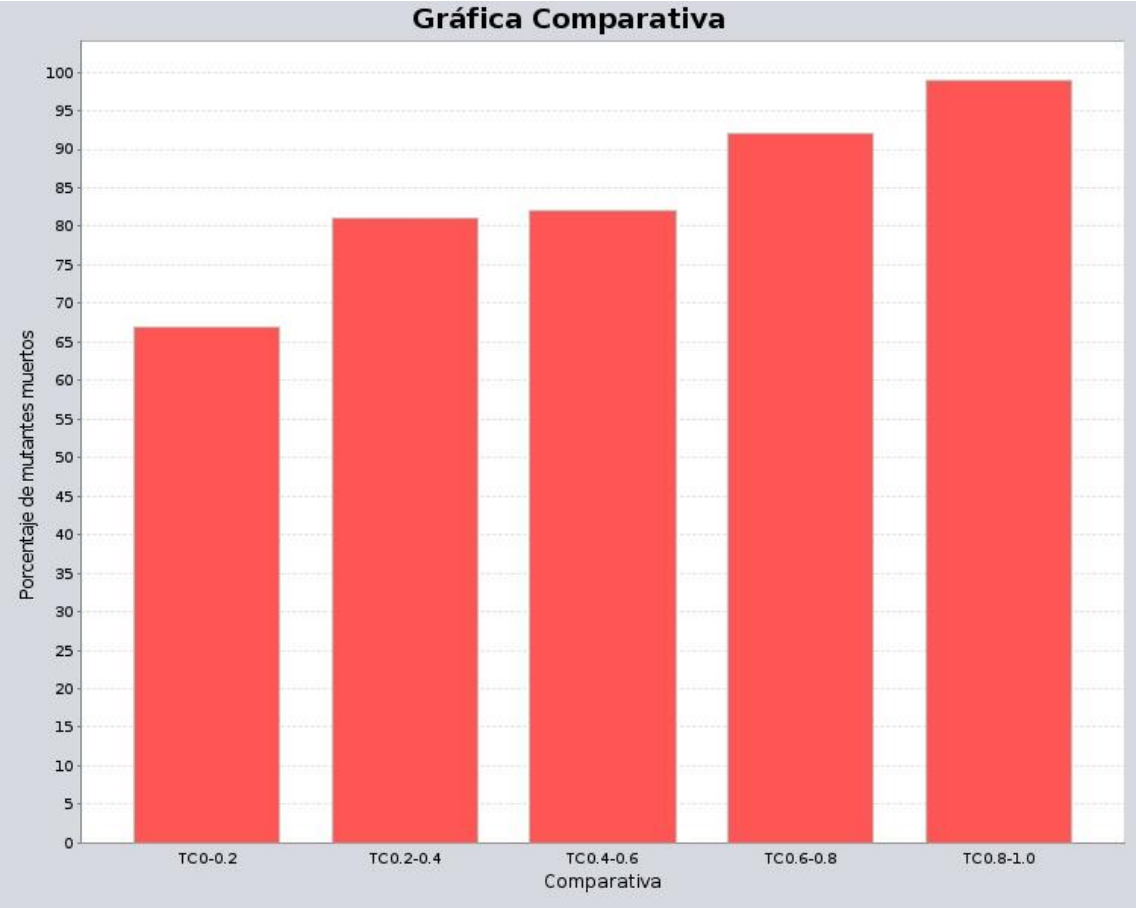
El primer caso se ha considerado como irrelevante, ya que suponer máquinas que no poseen entradas desde el exterior es un caso extremo que carece de interés. Para la gráfica anterior se han considerado máquinas objetivo con varias transiciones con distintas entradas para un mismo estado.

¿Puede explicar esto el hecho de que el primer intervalo obtenga un porcentaje tan elevado de mutantes muertos? Efectivamente, existe una relación entre ambos términos. Las máquinas creadas tienen varios estados, con varias transiciones por estado. Por tanto, tal y como está implementada la creación de mutantes, dichas máquinas pueden realizar cualquier tipo de mutación, ya sea eliminar un estado o una transición (mutaciones que no matan), como añadir un estado, una transición o cambiar una transición (mutaciones que matan). Como hay cuatro mutaciones que matan y dos que no, hay una gran probabilidad de que mueran bastantes mutantes. He aquí la explicación al elevado número de mutantes muertos.

Por otro lado, ¿por qué es ascendente la gráfica? El hecho de que progresivamente vaya muriendo un porcentaje cada vez más elevado de mutantes se debe a la forma de las máquinas. Para que el nivel de transparencia vaya aumentando, las máquinas objetivo cada vez deben tener menor número de transiciones con entradas para cada estado. Por tanto, cada vez hay menos estados y transiciones que se puedan eliminar, quedando tan sólo las cuatro mutaciones que provocan la muerte del mutante en la comprobación. Tal y como está implementada la aplicación, se escoge un tipo de mutación y, después, un estado al que aplicar dicha mutación. Si en dicho estado no puede aplicarse, se vuelve a escoger otro estado. Es evidente que, cuantos menos estados tengan transiciones eliminables o ellos mismos puedan serlo, mayor número de veces se aplicarán las técnicas de mutación que matan mutantes. Evidentemente podrían incluirse un número infinito de transiciones de salida, algo que no afectaría al cálculo de la transparencia de control. Este caso no ha sido considerado porque en el fondo no intervienen decisivamente en el cálculo de la métrica, pero sí desestabilizaría la gráfica.

La explicación al último intervalo también está definida por las características de las máquinas creadas. Como se ha explicado anteriormente, las máquinas con transparencia de control uno o cercano a uno, tienen una única transición de entrada para cada estado en la máquina objetivo. Además, se ha añadido alguna transición con salidas para algún estado. Por tanto, es factible la eliminación de estados y transiciones, lo que hace disminuir el porcentaje de mutantes muertos.

A continuación se muestra una gráfica con estas variaciones:

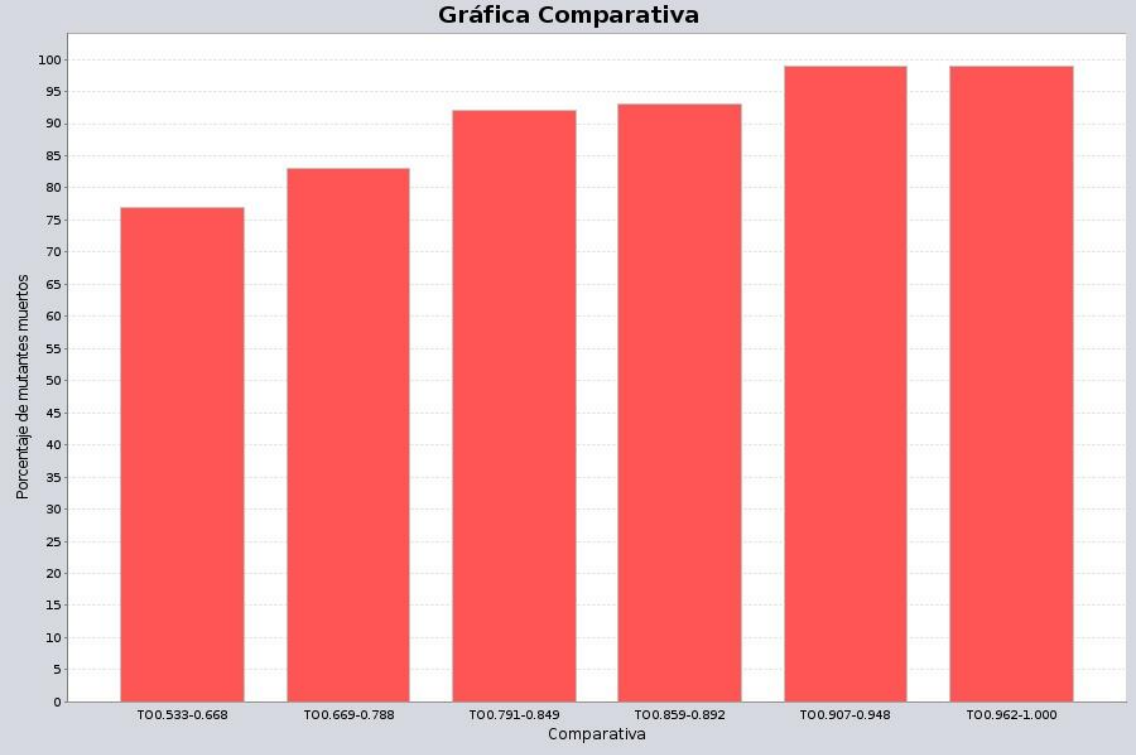


Como puede observarse, las características de la máquina afectan al resultado final. Una misma máquina puede tener transparencia de control cercana a cero con muchas transiciones o con pocas. Evidentemente, en función de una u otra alternativa, la gráfica puede variar. Pero su esencia, lo realmente importante, demuestra que tiende a matar más mutantes según tiende a transparencia de control cercana a uno.

A continuación, se analizará la transparencia de observación. Para ello se realizaron varios estudios y se procedió a analizar los resultados obtenidos.

El primero de los estudios significativo, comenzó elaborando varias máquinas aleatoriamente. A continuación se procedió a componerlas de manera aleatoria entre sí y se obtuvieron los valores de transparencia de observación que tenían. El objetivo era obtener un grupo de máquinas que tuviera valores variados para, a continuación, realizar el estudio. Se observó que la mayoría de las máquinas que se obtenían tenían un valor muy próximo a 1. Se procedió a clasificar algunas de ellas. Debido a la dificultad para obtener valores menores de dicha métrica, se decidió abandonar la idea de obtenerlas de manera aleatoria, procediendo a modificar las máquinas existentes para obtener máquinas con valores menores. Dichas modificaciones se realizaron de manera manual y, finalmente, se obtuvo un grupo de máquinas suficiente para la realización de estudio. Se obtuvieron 54 máquinas (108 contando máquina interfaz y

máquina objetivo), clasificadas en 6 grupos de 6 máquinas con valores de transparencia de observación desde 0,533 hasta 1.



Tomando la gráfica como punto de partida, se procede a analizar los resultados para la transparencia de observación en función de las características de las máquinas creadas. Lo primero destacable es que la gráfica tiene una clara tendencia ascendente. Para la realización de este estudio se han creado máquinas de manera manual, descartando todos aquellos factores superfluos que se escapan al estudio de dicha métrica. Cabe destacar que el estudio de los mutantes se realiza sobre la máquina objetivo, pero la transparencia de observación está influida por cambios tanto en la máquina objetivo, como en la máquina envoltorio. Evidentemente, se han realizado cambios en ambas máquinas, aunque la relevancia de dichos cambios sólo se vea reflejada en los cambios de la máquina objetivo. Como hemos visto en la definición de la transparencia de observación (sección 2.8), si introducimos una misma entrada, cabe esperar que el proceso interno de la máquina sea idéntico y nos acabe proporcionando una misma salida. Por tanto, las salidas de la máquina envoltorio nos importan de cara a la transparencia de observación. Si introduciendo una entrada “i1” (entrada de la máquina envoltorio), al final se puede recibir tanto una salida “o1” como “o2” (salidas de la máquina envoltorio), entonces la transparencia de observación se verá reducida. Pero todo esto tiene una visión puramente teórica, puesto que las modificaciones a la hora de crear mutantes se aplican en la máquina objetivo.

Así pues, dejando de lado el aspecto que afecta a la máquina envoltorio (que se aleja del análisis actual), las máquinas que tienen una transparencia de observación menor tienen las siguientes características:

1. Para un mismo estado de la máquina objetivo existen varias entradas.
2. Para un mismo estado de la máquina objetivo existen varias salidas.

¿Qué provocan estas dos situaciones? El hecho de tener varias transiciones desde cada estado, tal y como ha sido comentado en los puntos predecesores al actual, favorece la mutación en cualquiera de los supuestos posibles. Por tanto, existe un alto porcentaje de probabilidades de que el mutante acabe muriendo (cuatro de las seis posibles mutaciones matan). Pero también existe otro porcentaje relativamente importante de posibilidades de que el mutante acabe sobreviviendo, ya que al haber varias transiciones por estado puede eliminar una transición con elevada probabilidad (ya que al haber varias transiciones por estado, nos aseguramos que no exista peligro dejando un mutante huérfano).

¿Podría ocurrir que el porcentaje de mutantes muertos para valores muy bajos de transparencia de observación fuese extremadamente alto? Lo cierto es que es imposible. Como se ha comentado, para que una máquina tenga una transparencia de observación muy baja, se hace imprescindible que se vean afectados bastantes estados con varias entradas o salidas para la máquina objetivo. No son casos aislados los que disminuyen el resultado de manera significativa para esta métrica, sino una sucesión de modificaciones en el número de entradas o salidas comunes para un mismo estado. Esta imposición viene creada por la fórmula aplicada para calcular dicha métrica. Por tanto, como tiene que tener un elevado número de transiciones con esas características, esto hace que se activen aquellas mutaciones que no matan (eliminación de transición y de estado). Así que debemos descartar que suba el porcentaje de mutantes muertos de manera drástica para valores muy bajos.

¿Qué ocurre con los valores intermedios? Para este conjunto de valores la situación es diferente. La intuición nos muestra que una inversión de los porcentajes, es decir, que un intervalo más cercano a uno tenga un porcentaje de muertes menor que otro intervalo menor, es algo prácticamente imposible. En el fondo se traduce en probabilidades de éxito al intentar realizar una mutación. Una máquina que tiene mayor transparencia de observación que otra, al final, tiene menor número de transiciones que provocan ese bajón a la hora de realizar el cálculo. Por tanto, cuanto mayor es el valor para la métrica, menor probabilidades tiene la máquina de generar un mutante con una eliminación de un estado o eliminación de una transición. Debido a este hecho, va a tender a crear mutantes que mueran. Si la diferencia entre intervalos es muy pequeña, debido a que se ha elegido un gran número de intervalos (habida cuenta de que la transparencia de observación es prácticamente imposible que baje de 0.5, porque se calcula para cada estado y no todos pueden llegar a tener

caminos con cuatro transiciones como se requiere para que disminuya), entonces los porcentajes de muerte de mutantes para cada intervalo pueden quedar muy parejos.

Por último, en el caso de aquellas máquinas cuya transparencia de observación es uno, pueden producirse dos circunstancias:

1. La máquina tiene sólo aquellas transiciones que provocan que su transparencia de observación sea uno.
2. La máquina tiene aquellas transiciones que provocan que su transparencia de observación sea uno, pero también tiene otras transiciones que no influyen en la métrica

El caso propio del estudio es el primero. Las máquinas objetivo sólo tienen aquellas transiciones imprescindibles para que la transparencia de observación sea uno. Por tanto, las máquinas tienen una transición por cada estado. Además, en aquellos estados en los que ya no se encuentran problemas que hagan disminuir la métrica, no se han añadido varias transiciones que hagan modificar la tendencia de la gráfica y, además, tengan un carácter insustancial. Por tanto, las posibilidades de eliminación de una transición o un estado son bastante reducidas. ¿Cómo se explica entonces que no alcance el 100% de muertes? Evidentemente siempre hay transiciones que llevan a un estado al que ya se puede ir desde otro estado, por tanto se pueden eliminar. Así mismo, puede haber estados que no vayan a otro estado, o que vayan pero a ese otro estado se pueda llegar desde un estado intermedio, así que pueden eliminarse. Ahora bien, ¿cómo se explica el porcentaje tan alto de muertes de mutantes? Es la continuación de la explicación para los intervalos intermedios. Al haber pocas posibilidades de que se creen mutantes que no maten, se crean muchos que matan.

Ahora bien, ¿se podría reducir el porcentaje de muertes? Evidentemente, es un hecho que podría ocurrir. Mucho tiene que ver con este aspecto el punto segundo de los enumerados anteriormente. En las máquinas con transparencia de observación uno se puede introducir transiciones en puntos de la máquina que no afectan para el valor de la métrica. Este tipo de transiciones tienen dos efectos sobre la generación de mutantes. Por un lado favorecen la eliminación de transiciones, al haber varias para un estado. El segundo aspecto influye a la eliminación de estados. Como hemos visto en el párrafo anterior, ya se podían eliminar estados si en las máquinas no había transiciones insustanciales. Si añadimos transiciones en puntos que no afectan al cálculo de la métrica, evidentemente pueden provocar que a un estado se llegue desde varios estados distintos. ¿Qué provoca esto? Indudablemente aumenta las posibilidades de eliminación de estados. Antes se tenía un menor número de estados porque al haber sólo una transición por estado, muchos de los estados eran necesarios para acceder a otros. Sin embargo, ahora se pueden eliminar muchos más estados porque, aquellos estados a los que se llega transitando desde el estado a eliminar, tienen caminos alternativos que llegan hasta ellos.

Conclusiones finales

Hasta ahora se han analizado los resultados obtenidos para cada métrica. Pero, como se ha visto, no se pueden realizar afirmaciones categóricas basadas en los resultados hallados. ¿Es realmente la observación un factor influyente en el testing? Se ha analizado que, en función de la estructura interna de las máquinas, la observación puede ser favorable o no según las gráficas. Esto provoca una situación inesperada, un desconcierto respecto a los resultados que han aportado los experimentos. Sin embargo, se han ido obteniendo explicaciones para cada uno de los factores que afectan a los resultados esperados. Lo cierto es que, en un principio, existe una idea preconcebida en relación con el tema a tratar. Por ejemplo, cuando uno reflexiona sobre la complejidad, todas y cada una de las conclusiones conducen a afirmar que un código complejo empeora la detección de errores. De hecho, tiene sentido que esto sea así. Pero asociando esta métrica a las máquinas empleadas en los experimentos se observa que todo es relativo. Se ha analizado por qué es relativo y si tiene sentido y explicación pero, ¿qué significa?, ¿por qué hay que asumir que la estructura de las máquinas afecte significativamente en los resultados esperados?

Abordar esta cuestión se antoja como algo imprescindible, una vez que se sabe que no existe una afirmación categórica que elimine la opción de una explicación alternativa. El análisis de esta cuestión no puede sino acercar el debate sobre la proximidad entre métricas, es decir, ¿existe una posible relación que una a dos o más métricas para una máquina? Para profundizar en este concepto, debemos asociar puntos básicos de las definiciones de las métricas. ¿Qué objetivo debería deducirse de este análisis? Que esos cambios en los resultados analizados se producen debido a que dos máquinas cumplen la misma métrica, pero quizá una obtiene mejores resultados para otra métrica que su competidora. De pronto surge una nueva cuestión a analizar derivada de este pequeño esbozo inicial en el análisis, ¿se pueden relacionar todas las métricas entre sí? Se intentará profundizar en las dos últimas cuestiones a lo largo de los próximos párrafos.

Una máquina con observabilidad uno, se ha visto que puede matar muchos mutantes o todo lo contrario. Anteriormente se analizó que una máquina con observabilidad uno está conformada de dos maneras distintas. La primera suponiendo que no tiene ninguna salida o son máquinas con pocos estados y transiciones (este caso no es relevante). El segundo caso son aquellas máquinas que tienen un número razonable de estados, con bastantes transiciones por estado, aunque para cada salida sólo se puede ir a un estado. A priori el caso interesante, el de mayor interés y realismo es el segundo. A partir del primer caso, surge la idea de relacionar la métrica de observabilidad con otra métrica, de forma que entre ambas fuercen a la máquina parecerse más a la segunda máquina. Por ejemplo, suponiendo la máquina que no tiene transiciones de salida, se quiere conseguir que parezca a una con un número

razonable de transiciones. Si se introduce la necesidad de una transparencia de observación elevada, dicha máquina necesitará incluir transiciones de entrada y salida. Es más, necesita que sólo se introduzca una transición de entrada o salida para cada estado. Este requerimiento es imprescindible, ya que si no bajaría la medida de transparencia de observación. De esta forma la máquina es forzada a tener una determinada forma pero, además, esto no reduce drásticamente su observabilidad. Quizá una reducción de una décima en observabilidad sea conveniente si se tiene además que la misma máquina tiene una elevada métrica de transparencia de observación. Este razonamiento simplemente debe conducir a pensar que las distintas formas de generar máquinas con el mismo resultado para una métrica, son en realidad el mismo caso. De esta forma se debe concluir que, aunque se ha realizado el estudio por separado, en realidad existen métricas que se complementan.

Otro caso bastante llamativo es el que se produce entre las métricas de controlabilidad y observabilidad. Una máquina con observabilidad elevada a la que no se le han introducido transiciones de entrada resulta bastante irreal. Sin embargo, si se quiere que tenga una elevada controlabilidad, se pueden introducir transiciones de entrada de forma que aumente la controlabilidad y, no se reduce la observabilidad. Por tanto, se ha arreglado el problema de tener una máquina irreal al combinar ambas métricas. Este razonamiento ayuda a comprender que algunas métricas pueden complementarse para que las máquinas creadas tengan un parecido mayor a la realidad y, a la vez, conseguir valores representativos para dichas métricas.

Bibliografía

Brinksma01 Ed Brinksma and Jan Tretmans. Testing Transition Systems: An Annotated Bibliography. University of Twente _Faculty of Computer Science, Formal Methods and Tools Group. P.O. Box 217, 7500 AE Enschede, The Netherlands.

Tretmans96 Jan Tretmans. Conformance Testing with Labelled Transition Systems: Implementation Relations and Test Generation. Tele-Informatics and Open Systems Group Department of Computer Science University of Twente. P.O. Box 217 7500 AE Enschede. The Netherlands.

Poston12 Robin Poston, Jignya Patel, and Jasbir Dhaliwal. A Software Testing Assessment to Manage Project Testrability. Systems Testing Excellence Program, University of Memphis.

Lee96 *David Lee, Mihalis Yannakakis.* Principles and Methods of Testing Finite State Machines — A Survey. AT&T Bell Laboratories Murray Hill, New Jersey

Trabajos E-Prints Complutense. Facultad de Informática (742) Depto. de Sistemas Informáticos y Computación, http://eprints.ucm.es/view/divisions/457.html