***Métodos Formales:***

Los métodos formales brindan garantías de corrección notablemente más fuertes que las brindadas por técnicas informales, gracias a las sólidas bases matemáticas sobre las cuales están fundados.

En busca de proveer garantías del correcto funcionamiento del software, surgieron una variedad de técnicas y metodologías de desarrollo con sólidas bases matemáticas y lógicas conocidas como Métodos Formales. Estos se basan en el uso de notaciones matemáticas con semántica formal, que permiten razonar sobre modelos de las especificaciones de los sistemas. Dichos modelos deben reflejar las características del programa que se quiere construir. Si esto sucede, pueden verificarse propiedades sobre los modelos, aun antes de haber construido el software, lo que posibilita detectar errores en etapas tempranas del desarrollo. Además, la utilización de métodos formales ayuda a la comprensión del problema a resolver por parte de los desarrolladores, lo que contribuye a mejorar la calidad del programa.

Los métodos formales se usan en general en dos actividades: la especificación y la verificación. El proceso de especificación es el acto de describir las cosas de manera precisa. El principal beneficio de hacer esto es conseguir una comprensión más profunda del sistema que está siendo especificado. A través de este proceso de especificación, los desarrolladores descubren errores de diseño, inconsistencias, ambigüedades y partes incompletas. Un subproducto tangible de este proceso es un resultado que puede ser analizado formalmente, por ejemplo, chequeado para comprobar que sea consistente internamente o usado para derivar otras propiedades del sistema especificado. La especificación es un mecanismo de comunicación útil entre cliente y diseñador, entre diseñador e implementador, y entre implementador y verificador. Además, sirve como documentación del sistema.

El ***model ckecking*** es una técnica que consiste en la construcción de un modelo finito del sistema y comprobar que las propiedades deseadas se cumplen en el modelo. Es una técnica de verificación que examina todos los posibles escenarios del sistema de manera sistemática. Así se puede demostrar que un modelo de sistema dado satisface realmente una cierta propiedad. Es una técnica automatizada que, dado un modelo de estado finito de un sistema y una propiedad formal, comprueba sistemáticamente si esta propiedad es válida para (un estado dado en) ese modelo. Explora todos los posibles estados en un sistema. Requiere una declaración precisa y sin ambigüedad de las propiedades a examinar. Examina todos los estados del sistema pertinente para comprobar si satisfacen la propiedad deseada. Si encuentra un estado que viola la propiedad, el MC proporciona un contraejemplo que indica cómo el modelo podría alcanzar el estado no deseado. Actualmente se aplica al análisis de especificaciones de sistemas de software.

Se utilizan Sistemas de Transición (TS) para describir el comportamiento del sistema. Son grafos dirigidos donde el estado (nodos) describe información sobre el sistema en cierto momento de su comportamiento, y la transición (arcos) especifica cómo el sistema puede evolucionar de un estado a otro.

El ***SAT-Solving*** es una técnica que se basa en el chequeo de satisfacibilidad, es decir, determinar si las variables de cierta fórmula booleana pueden tener asignaciones en las cuales tal fórmula se evalúe a verdadero. Igualment

e importante, es determinar cuándo tales asignaciones no existen. En este último caso, se dice que la fórmula es insatisfacible, mientras que en otro caso la fórmula es satisfacible. Una herramienta que se basa en SAT-Solving, es Alloy Analyzer. Alloy Analyzer da soporte al análisis de modelos escritos en el lenguaje Alloy, uno de los más populares lenguajes formales derivados de Z, que se utiliza para describir programas utilizando una lógica relacional.

El desarrollo del *Alloy Analyzer* estuvo originalmente inspirado en el análisis automático provisto por el Model Checking. Sin embargo, el Model Checking es poco adecuado para el tipo de modelos especificados en Alloy, por lo cual el Alloy Analyzer incorpora SAT-Solving. Básicamente, se construye una fórmula booleana correspondiente a partir del modelo relacional descripto en Alloy, y se invoca a un SAT-Solver sobre dicha fórmula booleana. Una vez que se encuentran soluciones, se realiza la asociación de constantes a variables en el modelo lógico relacional.

El Alloy Analyzer, por lo tanto, es una herramienta que permite analizar especificaciones, en busca de ambigüedades, inconsistencias, comprensiones erróneas, etc, de manera completamente automática. Esto se logra mediante la reducción del análisis a un problema de satisfacibilidad de una formula proposicional, y el empleo de poderosos y sofisticados SAT Solvers.

Coloquio Métodos Formales – 20/12/2017

# Ejercicio 1:

### CONSIDERE LO PRESENTADO EN LA MATERIA SOBRE MODEL CHECKING Y RESPONDA:

1. ***Describa la teoría sobre la que se base Alloy.***

Alloy se basa en las siguientes teorías:

* Teoría de Modelos: Las propiedades se expresan a través de axiomas, el modelo se deriva de conjuntos. Se relaciona la consecuencia lógica con la consecuencia semántica (conjunto de las posibles secuencias de ejecución). La semántica de las sentencias lógicas esta dada por valores de verdad. Su simplicidad la convierte en una buena herramienta de especificación.
* Teoría de Tipos: Considera estructuras matemáticas (conjuntos con operaciones particulares). Esta aproximación ha tenido problemas a la hora de razonar sobre los modelos de forma segura. Su simplicidad se convierte en una buena herramienta de especificación.

Alloy no se basa estrictamente en la teoría de modelos. A diferencia de Alloy que utiliza restricciones para denotar propiedades, la teoría de modelos se expresa con axiomas. Además, Alloy se basa en la teoría de tipos, donde también se utilizan conjuntos para operar con ellos.

1. ***Remarque las diferencias que nota entre la aproximación adoptada por los desarrolladores de Alloy y la aproximación adoptada por los llamados Model Checkers tradicionales. Indique ventajas y desventajas más significativas de cada aproximación.***

Model checking es una técnica automatizada, dado un modelo de forma de autómata finito y una propiedad formal, chequea sistemáticamente si la propiedad se verifica para un dado estado en el modelo.

El SAT-Solving es una técnica que se basa en el chequeo de satisfacibilidad, es decir, determinar si las variables de cierta fórmula booleana pueden tener asignaciones en las cuales tal fórmula se evalúe a verdadero. Igualmente importante, es determinar cuándo tales asignaciones no existen. En este último caso, se dice que la fórmula es insatisfacible, mientras que en otro caso la fórmula es satisfacible. Una herramienta que se basa en SAT-Solving, es Alloy Analyzer. Alloy Analyzer da soporte al análisis de modelos escritos en el lenguaje Alloy.

El desarrollo del Alloy Analyzer estuvo originalmente inspirado en el análisis automático provisto por el Model Checking. Sin embargo, el Model Checking es poco adecuado para el tipo de modelos especificados en Alloy, por lo cual el Alloy Analyzer incorpora SAT-Solving. Básicamente, se construye una fórmula booleana correspondiente a partir del modelo relacional descripto en Alloy, y se invoca a un SAT-Solver sobre dicha fórmula booleana. Una vez que se encuentran soluciones, se realiza la asociación de constantes a variables en el modelo lógico relacional.

Model Checkers:

Ventajas:

* Es una aproximación general a la verificación que es aplicable a un rango amplio de aplicaciones (incluyendo sistemas embebidos, ingeniería de software y diseño de hardware).
* Soporta verificación parcial, es decir, las propiedades pueden chequearse individualmente permitiendo enfocarse primeramente en propiedades esenciales. No se requiere una especificación completa de requerimientos.
* No es vulnerable a la probabilidad de exposición de errores; a diferencia de la simulación y testeo donde su objetivo es identificar los defectos mas probables.
* Provee información de diagnóstico en caso de que una propiedad sea invalidada; muy útil para realizar debugging.
* No requiere mucha interacción con el usuario ni nivel de experiencia.
* Está despertando el interés de la industria rápidamente.
* Puede integrarse fácilmente en ciclos de desarrollo existentes.
* Posee una base sana y matemática (sustentado en algoritmos de teoría de grafos, estructuras de datos y lógica).

Desventajas:

* Es más apropiado para aplicaciones centradas en control. Las aplicaciones centradas en datos trabajan típicamente sobre dominios infinitos.
* Su aplicabilidad queda sujeta a aspectos de decibilidad; para sistemas con estados infinitos o razonar acerca de tipos de datos abstractos (que requieren lógica semi-decidible o no decidible), el model checking por lo general no es efectivamente computable.
* Verifica un modelo del sistema, no el sistema en si mismo. Se necesitan técnicas complementarias, tales como testing, para poder encontrar errores de fabricación (hardware) o en el código fuente (software).
* No garantiza completitud ya que no puede afirmar nada sobre las propiedades no chequeadas.
* Sufre del problema de “explosión de memoria”, el numero de estados necesarios para el modelo de un sistema real pueden exceder fácilmente la cantidad de memoria disponible de la computadora.
* Se requiere cierto grado de experiencia para encontrar abstracciones apropiadas para obtener modelos de sistemas más pequeños.
* Puede tener defectos de software.
* No permite chequear generalizaciones.

Alloy:

Ventajas:

* Liviano: Pequeño y fácil de utilizar. Capaz de expresar propiedades comunes de manera natural.
* Preciso: Cuenta con una semántica matemática simple y uniforme.
* Tratable: Análisis semántico susceptible a ser eficiente y completamente automatizado (considerando limitaciones de alcance).
* Puede utilizarse para modelados de datos en general:
  + Cualquier dominio de individuos y
  + Relaciones entre ellos
* Resulta particularmente útil para especificar clases, asociaciones entre ellas y restricciones sobre las asociaciones.
* Cuenta con un sistema de módulos que permiten la modularizacion y reuso de modelos. Un módulo define un modelo que puede ser incorporado como **submodelo** de otro.
* Para facilitar el reuso, los módulos pueden ser *paramétricos* para **una o más signaturas.**

Desventajas:

* Trabaja sobre el modelado del sistema, no en el sistema propiamente dicho.
* No permite trabajar con cuestiones numéricas mayores a 8 ya que posee un rango acotado.
* El mecanismo de análisis tiene limitaciones. La principal tiene que ver con la incompletitud del análisis. Más específicamente, cuando utilizamos la herramienta para comprobar una propiedad, pueden darse dos situaciones:
  + que la herramienta encuentre un contraejemplo, garantizando que la propiedad es inválida,
  + que la herramienta no encuentre contraejemplos dentro de las cotas establecidas por el usuario. En este caso, al no encontrar contraejemplos, ganamos confianza en la propiedad, aunque no tenemos garantías de su validez, ya que podrían en principio existir contraejemplos fuera de las cotas establecidas.

1. ***Explique de qué maneras se puede utilizar el Model Checker Alloy. Ejemplifique cada uso posible.***

Los métodos formales se usan en general en dos actividades: la especificación y la verificación. El proceso de especificación es el acto de describir las cosas de manera precisa. El principal beneficio de hacer esto es conseguir una comprensión más profunda del sistema que está siendo especificado. A través de este proceso de especificación, los desarrolladores descubren errores de diseño, inconsistencias, ambigüedades y partes incompletas. Un subproducto tangible de este proceso es un resultado que puede ser analizado formalmente, por ejemplo, chequeado para comprobar que sea consistente internamente o usado para derivar otras propiedades del sistema especificado. La especificación es un mecanismo de comunicación útil entre cliente y diseñador, entre diseñador e implementador, y entre implementador y verificador. Además, sirve como documentación del sistema.

1. ***Presente y explique las diferentes maneras de modelar y verificar dinámica en Alloy.***
2. ***Compare las aproximaciones presentadas en el inciso anterior bajo dos criterios de comparación. Ejemplos: eficiencia, legibilidad (facilidad de lectura y escritura de los modelos), facilidad de interpretación de los ejemplos mostrados por el verificador, etc.***

# Ejercicio 2:

### CONSIDERE LOS THEOREM PROVERS Y RESUELVA:

1. ***Explique la relación que existe entre la lógica de Hoare y las pruebas de correctitud en los Theorem Provers.***

Hoare propuso un sistema lógico, el cual proporciona una serie de reglas de inferencia para razonar sobre la corrección de programas imperativos. La principal característica de esta lógica, es la terna ”{Q} S {R}”, donde Q y R son predicados lógicos que deben satisfacerse para que el programa S funcione. Una importante ventaja del método de Hoare es que era composicional, tal es así que la demostración de un programa se obtenía desde la demostración de sus sub-programas.

Los Theorems Provers utilizan diseño por contrato para describir de manera precisa la semántica de una interfaz, es decir, las especificaciones que se deben cumplir. Se incluyen descripciones de comportamiento, efectos de la invocación a un método, operación o abstracción. Hacen énfasis en las responsabilidades en los roles y/o obligaciones en una especificación. Se establece un contrato entre el llamado y el llamador. Donde el llamado garantiza proveer el resultado previsto, y el que realiza la llamada garantiza los pre-requisitos; por lo que los contratos son bidireccionales, es decir, especifican las obligaciones y beneficios para cada parte. Los contratos deben describirse en un lenguaje preciso y matemático.

Los Theorem Provers se vinculan con la lógica de Hoare ya que hay relación entre los predicados lógicos Q y R con las precondiciones y postcondiciones de un contrato y, el propio programa S. Un par de pre/post condiciones para un método m se satisface para la implementación si: m es llamado en cualquier estado que satisface la precondición entonces en cualquier estado de terminación de m la postcondicion es verdadera. Lo mismo ocurre con la lógica de Hoare.

JML es un lenguaje de especificación. Integra especificación e implementación en un solo lenguaje. Provee el contexto para diseñar por contrato.

Basandonos en KeY como Theorem Prover, sabemos que utiliza la lógica dinámica. La lógica dinámica puede ser vista como una extensión de la lógica de Hoare. La fórmula de la lógica dinámica **Pre -> [P] Post** es similar a la de Hoare **{pre} P {post}.**  Pero en contraste con la lógica de Hoare, el conjunto de fórmulas de la lógica dinámica está cerrada bajo los operadores lógicos usuales. En la lógica de Hoare, las formulas pre y post son fórmulas de primer orden, mientras que en la lógica dinámica pueden contener programas. Al usar un programa en **pre**, es más fácil especificar que una estructura de dato de entrada no es cíclica, lo que es imposible en la lógica de primer orden.

1. ***Explique cómo se realiza la especificación en el caso concreto de KeY. Ejemplifique.***

Ejemplo:

@ public normal\_behavior

@ requires (precondiciones);

@ ensures (postcondiciones);

Finalmente, para probar que bajo el contrato el programa cumple con lo establecido se debe tener en cuenta: Las invariantes de clase, las precondiciones, las posibles asignaciones a elementos dentro del programa, la ejecución normal/excepcional y las postcondiciones.

# EJERCICIO 3

### LÓGICA:

1. ***¿Es suficiente con la lógica de primer orden para trabajar en los Theorem Provers? Justifique.***
2. ***¿Es suficiente con la lógica de primer orden extendida con tipos para los tipos de verificadores formales usados en la materia? Justifique. Explique además por qué esta extensión de la lógica es necesaria.***

Se debe utilizar una lógica de primer orden extendida con tipos para los distintos verificadores formales dado que, de esta forma, se permite realizar una transitividad entre las propiedades de objetos del mismo tipo o de un subtipo. Es decir, propiedades (invariantes) de una subclase que implican propiedades de la superclase.

1. ***Describa el tipo de lógica que se utiliza para trabajar en los Theorem Provers. Enfatice los aspectos propios de la misma.***
2. ***Describa el cálculo sobre el que se basan los Theorem Provers. Describa además el proceso de verificación formal.***

El calculo elegido para trabajar en los Theorem Provers es el calculo de secuentes (sequent calculus). Los datos básicos manipulados por las reglas de este calculo son secuentes, es decir, formulas de la forma:

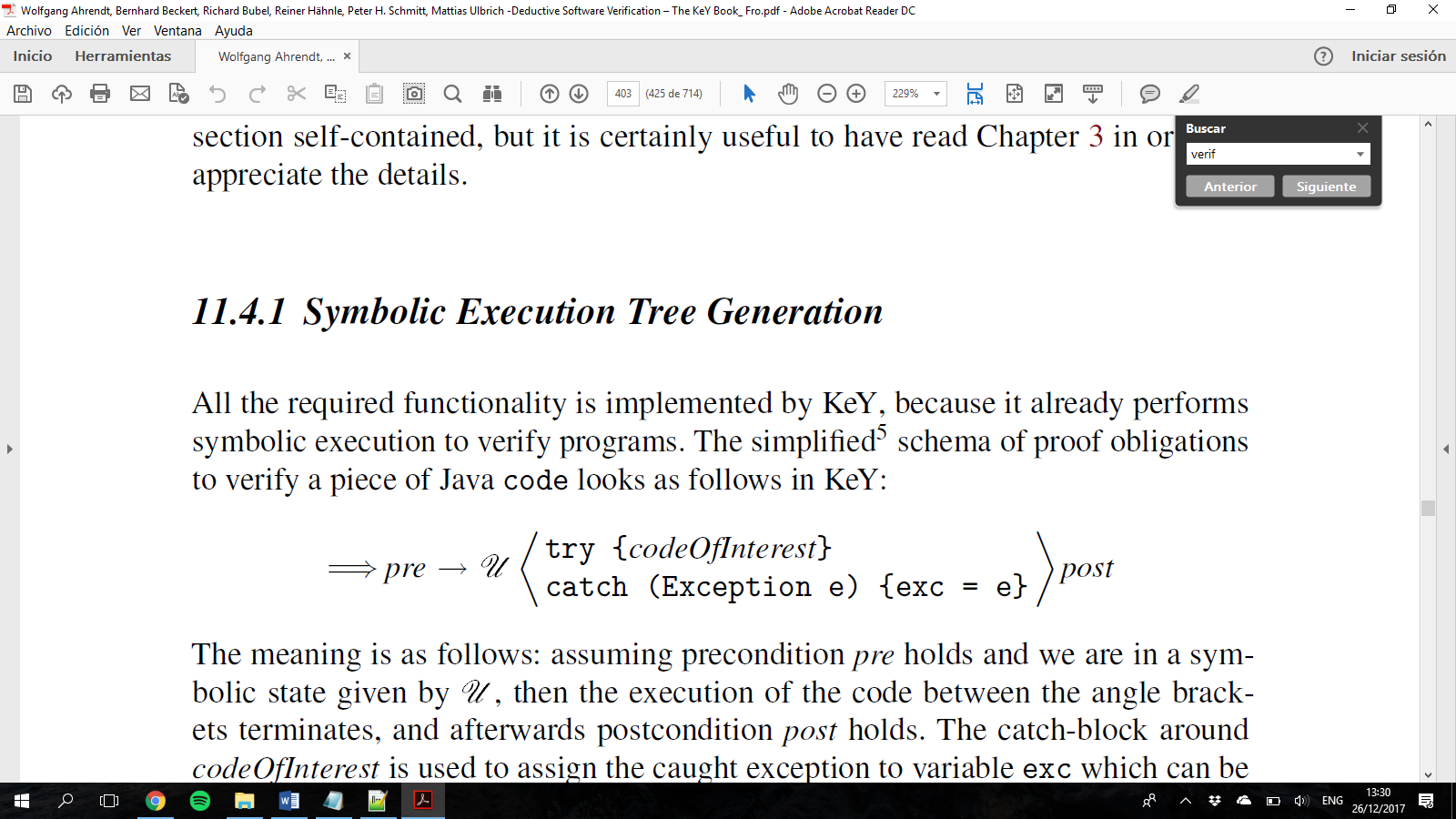
Ф1,…, Фn ⟹ Ѱ1,…, Ѱm

La fórmula previa, de sequent calculus, resulta válida cuando la fórmula Ф1 ⋀ … ⋀ Фn ⟹ Ѱ1 ⋁ … ⋁ Ѱm lo es.

Para el proceso de verificación formal, el cálculo realiza la interpretación simbólica del programa. El proceso sigue los siguientes lineamientos:

* Trabaja sobre la primera sentencia activa.
* Descompone sentencias complejas en sentencias más simples.
* Transforma asignaciones simples en updates.
* La acumulación de updates captura los cambios en los estados del programa.
* Las ramificaciones en el flujo de control inducen particiones en la prueba.
* Los updates computan la precondición más débil de U con respecto a la fórmula φ, Г⇒{U} φ

Los updates secuenciales y paralelos cubren una buena parte de lo que se requiere para evaluar una fórmula en un estado, pero no todo.



El significado es el siguiente: Asumiendo que la precondición *pre* se cumple y nos encontramos en un estado simbólico dado por *U*, entonces la ejecución del código entre los <> termina, y luego la postcondicion *post* se cumple. El bloque de captura de la excepción se utiliza para asignar la excepción que se captura a una variable *exc* que puede ser utilizada por la postcondicion para separar la terminación normal de la excepcional.

Las reglas que se aplican en la modalidad <código>post reescriben la primera declaración activa en el código y luego continúan con la ejecución simbólica. La ejecución simbólica se realiza a nivel de expresiones atómicas, por lo que las declaraciones y expresiones en Java se deben descomponer antes de que puedan ser ejecutadas.

Todas las reglas de ejecución simbólica dividen, si es necesario, el árbol de prueba para cubrir todos los caminos posibles que se pueden realizar durante la ejecución. Debe ser tarea de la prueba automática (o del usuario) comprobar la inviabildad de las premisas de prueba antes de continuar con la ejecución.

# EJERCICIO 4

***Explique de qué maneras se puede probar la correctitud de ciclos repetitivos en KeY. Remarque las diferencias. Indique en particular el propósito de la cláusula ‘decreases’ en las especificaciones JML.***

Coloquio Métodos Formales 11/12/17

# Ejercicio 1

### TENIENDO EN CUENTA LA HERRAMIENTA ALLOY:

1. ***¿Cuál es la principal diferencia semántica entre las aserciones y las expresiones de correctitud expresadas en la cláusula run?***

Aserciones:

Suele ocurrir que el usuario cree que el modelo cumple ciertas restricciones que no son expresadas de manera directa. Las aserciones expresan estas restricciones adicionales que el analizador chequea si se verifican:

* Si la aserción no se verifica entonces el analizador produce una instancia contraejemplo (que ilustra por qué la propiedad falla)
* Si la propiedad expresada como aserción falla, puede ser exprese algo que debiera ser un invariante o bien es necesario refinar la especificación o aserción.

Las aserciones se chequean a través del comando check (el cual debe cubrir el alcance para todas las signaturas).

Run:

El comando puede incluir condiciones:

* Se lo puede guiar para elegir instancias con ciertas características (por ejemplo, forzar a que ciertos conjuntos y relaciones sean no vacíos).
* Las condiciones que se agregan al comando no son parte de la especificación “real”.

Por lo tanto, la aserción verifica que no existan instancias que no cumplan con lo estipulado en la aserción para todo el modelo. Sin embargo, el comando run únicamente elige instancias que aseguren las características enunciadas, pero no asegura que el modelo por completo cumpla con las restricciones dadas.

1. ***Explique las alternativas que provee la herramienta para modelar y verificar dinámicamente. Establezca que ventajas y desventajas observa para cada una de ellas.***

El modelado dinámico describe transiciones entre estados, las propiedades son operaciones.

Alloy no cuenta con una noción embebida de estado, ni de transición de estados. Sin embargo, hay varias maneras de modelar los aspectos dinámicos del sistema:

* Solución nuevo átomo: Tener un nuevo átomo que refleje el cambio realizado. De esta manera se captura la noción de cambio.
  + Ventaja:
    - No es necesario agregar nuevas características al conjunto, por lo que es fácil de realizar.
  + Desventaja:
    - No se trata del mismo objeto, sino que de otro que en general es igual salvo por el efecto esperado de la operación.
    - Legibilidad: Los distintos estados están dispersos dentro del modelo, se deben identificar apropiadamente a partir del modo texto.
* Solución nuevo patrón: Trata a las acciones y operaciones en un estado global, que define el comportamiento de una máquina abstracta. El estado puede pensarse de manera global (State) o local (Time).
  + Ventajas:
    - *Legibilidad (facilidad de escritura y lectura de los modelos)*: Crea un ordenamiento total sobre los estados por lo que es posible identificar fácilmente todos los estados del modelo.
  + Desventajas:
    - Es necesario agregar una nueva columna para identificar el estado.
    - No es posible generalizar el modelo de cambio de estado ya que el comportamiento depende de la parte del modelo afectado.

# Ejercicio 2

### PARA CADA UNA DE LAS HERRAMIENTAS VISTAS EN LA MATERIA:

1. ***Explique por qué la lógica de primer orden resulta insuficiente.***

***Alloy*** es un lenguaje de especificación formal basado en la idea de conjuntos y relaciones. Las operaciones vinculadas con las propiedades (hechos, aserciones, predicados, funciones) utilizan un fragmento de la lógica de primer orden, ya que todo no es computable.

Se utiliza una lógica relacional que combina los cuantificadores de la lógica de primer orden con los operadores del cálculo relacional.

* En la lógica de primer orden, existen dos tipos de expresiones: nombres de relaciones, que son utilizadas como predicados, y tuplas formadas por variables cuantificadas.

Se utiliza para construir conjuntos o relaciones a partir de restricciones.

* En el cálculo relacional, las expresiones denotan relaciones, y no existen los cuantificadores.

Se utiliza para expresar de forma más concisa ciertas restricciones. Por ejemplo: no ^r & iden para decir que la relación r es acíclica.

Además, el cálculo de predicados no posee la clausura transitiva, por lo que ciertas propiedades de accesibilidad no se pueden expresar. El caculo relacional, por otro lado, no posee cuantificadores, y no todas las ocurrencias de cuantificadores del cálculo de predicados se pueden expresar relacionalmente. Por lo tanto, es necesario realizar una combinación de ambos.

1. ***Explique qué tipo de lógica se utiliza, descríbala y explique por qué cree que resulta apropiada para la herramienta.***

La *lógica* permite la construcción de los bloques del lenguaje. Todas las estructuras son representadas como relaciones, y todas las propiedades estructurales son expresadas mediante simples (pero poderosos) operadores relacionales. Los estados y ejecuciones son expresadas mediante fórmulas y expresiones booleanas, conocidas como constraints, permitiendo refinarlos mediante la introducción de nuevos constraints.

El *lenguaje* agrega pequeños detalles a la sintaxis de la lógica para estructurar la descripción. Alloy permite sub-tipos, unión de tipos y brinda un sistema simple de módulos que permite que declaraciones genéricas y constraints sean reusados en diferentes contextos.

La lógica utilizada debe ser pequeña, simple y expresiva. Alloy utiliza una lógica relacional que combina los cuantificadores de la lógica de primer orden con operadores del cálculo relacional. Esta lógica es fácil de aprender y sorpresivamente poderosa. Una de las características clave, que la distingue de las lógicas tradicionales, es la generalización de la noción del join (composición) relacional.

# Ejercicio 3

***¿Qué es un update para KeY? Explique por qué resulta una pieza esencial a la hora de demostrar que un servicio JAVA es correcto.***

Los updates denotan cambios de estado. A diferencia de los fragmentos del programa, los updates son más simples y más restrictivos. Por ejemplo: los updates siempre terminan, y las expresiones que ocurren en los updates nunca poseen efectos colaterales.

Los updates son relevantes en KeY dado que se necesita una notación para los cambios de estado simbólicos:

* La ejecución simbólica debe seguir el flujo de ejecución natural del programa.
* Requiere una representación breve de los cambios de estado efectuados por el programa en una ejecución simbólica ramificada.
* Se desea simplificar los efectos de la ejecución temprana de programas.

La lógica dinámica no almacena toda la información de los estados, sino que, al realizar un update, se guarda únicamente lo que se modificó evitando desperdiciar tiempo y espacio. Además, los estados se encuentran ordenados por un criterio.

# Ejercicio 4

1. ***Explique cómo se puede asegurar la ausencia de efectos colaterales en JML. Establezca las diferencias e indique en qué contexto utilizará una u otra opción.***
2. ***Indique por qué razón resulta tan crítico asegurar la ausencia de efectos colaterales en el contexto de los Theorem Provers.***

# Ejercicio 5

1. ***Describa las anotaciones JML que pueden usarse para verificar un ciclo repetitivo, indique el sentido de cada una de ellas.***

Diverges: Se utiliza para la correctitud parcial de un bucle, es decir, permite que el método no finalice. En este caso, igualmente es necesario poseer una invariante de ciclo para que se satisfaga durante la ejecución del mismo (al comenzar, en el cuerpo del ciclo y al finalizar).

Loop\_invariant: Una formula invariante cuya validez sea preservada por el ciclo repetitivo, tanto por la condición como por el cuerpo. De esta manera, la invariante fue validada al comenzar el ciclo y aún lo es luego de una cantidad arbitraria de interacciones. Utilizar una invariante de ciclo es esencialmente un argumento inductivo, asegurando que la invariante se mantiene para todas las iteraciones del ciclo y, por lo tanto, se mantiene cuando el ciclo termina.

Decreasing: Mientras que no exista una clausula “*diverges true”* se requiere probar que un método finaliza. En presencia de un ciclo, únicamente es posible si se expresa la cláusula decreasing, en conjunto con la invariante de ciclo. El termino que se decrementa debe estar bien formado, es decir, no puede decrementarse por siempre. Se debe mostrar que se decrementa estrictamente para cada uno de los ciclos de la iteración y que evalúa a un valor no-negativo para cualquier estado que satisface la invariante. Por lo tanto, es suficiente para concluir que el bucle finaliza.

Assignable: Esta cláusula para ciclos indica qué puede modificarse.

1. ***¿Es posible demostrar correctitud total? En caso afirmativo justifique, en caso negativo explique qué características debe tener el ciclo para poder realizar una demostración de correctitud total.***

La regla de la invariante de ciclo para correctitud total puede derivarse de la versión de correctitud parcial de manera directa, simplemente se añade un termino de decrecimiento con las siguientes obligaciones de prueba:

* Se debe restringir la invariante (inv) para que el termino que decrece (d) sea siempre positivo, es decir, se debe cumplir inv ⋀ d ≥ 0.
* La postcondicion del cuerpo del ciclo debe asegurar que el valor de d es estrictamente menor que al principio de la ejecución del cuerpo.

Al nivel de JML, la correctitud total se logra a partir de:

1. Remover la directiva de correctitud parcial “diverges true”, del contrato establecido.
2. Añadir una directiva “decreasing d;”, donde d es el termino que decrece.

Siempre que no se defina una clausula diverges, es necesario probar que un método termina. En presencia de un bucle, esto solo es posible si una clausula decreciente (también denominada variante) se proporciona junto con la invariante de ciclo. El término decreciente debe estar bien formado, lo que significa que no puede disminuir para siempre. Para la cláusula decreciente, se debe demostrar que es estrictamente decreciente para cada iteración del ciclo y que evalúa a un valor no negativo en cualquier estado que se satisfaga el invariante. Por lo tanto, esto es suficiente para concluir que el ciclo termina.

1. ***De acuerdo a lo expresado en los incisos anteriores muestre las opciones que tiene para la verificación de ciclos repetitivos (provistas desde JML). Indique las diferencias.***

Coloquio Métodos Formales 5/12/2016

# Ejercicio 1:

***Explique de qué manera el model checker establece si una dada propiedad es válida o no. Indique además por qué internamente el model checker niega la propiedad a probar.***

El model checking es una técnica automatizada. Dado un modelo de un sistema en forma de autómata finito y una propiedad formal (definida en lógica temporal), chequea sistemáticamente si se verifica para un dado estado en el modelo.

Se pueden distinguir cuatro fases:

1. Fase de modelado: Se modela el sistema usando el lenguaje de descripción para el mismo que provee el model checker a utilizar. Se realizan algunas simulaciones para realizar el primer chequeo de sanidad. Formalizar la propiedad a chequear utilizando el lenguaje de especificación de propiedades.
2. Fase de ejecución: Se ejecuta el model checker para asegurar la validez de la propiedad en el modelo del sistema.
3. Fase de análisis: Si se satisfizo la propiedad entonces se debe chequear la próxima (si es que la hay). Si la propiedad no fue satisfecha, analizar el contraejemplo provisto por la simulación; refinar el modelo, diseño o propiedad; repetir el proceso de forma completa. Si no hay memoria suficiente, tratar de reducir el modelo e intentarlo nuevamente.
4. Organización de la verificación: El proceso de verificación debe ser planificado, administrado y organizado.

Internamente el model checker trabaja con un autómata del modelo y otro para la propiedad (negada), y luego, trabaja con un autómata intersección de estos, ya que reconoce el lenguaje intersección.

¿La propiedad representada por la fórmula ϕ es siempre verdadera en T?

* Se representa el sistema T como un autómata Büchi que acepta aquellas palabras que corresponden a ejecuciones de T. llamémoslo Bt.
* Construir un autómata de Büchi para la negación de la fórmula ϕ. Llamémoslo B¬ϕ.

Demuestra (MC) que se cumple la propiedad negándola y buscando un contraejemplo, estilo prueba por el absurdo. Para verificar la condición se construye al autómata intersección y se busca un ciclo hacia el estado aceptador. Si el lenguaje generado por Bt ⋂ con el lenguaje generado por B¬ϕ es vacío, entonces ϕ se verifica. Si no, cada elemento del conjunto resultante es un contraejemplo para la fórmula ϕ.

Si no negara la propiedad al verificar (MC), tendría que verificar que la propiedad se cumpla en todas las ejecuciones posibles que pueden suceder en el programa. Por lo que es una propiedad insatisfacible, ya que la intersección es potencialmente infinita y, para probar que la propiedad vale, se debería probar que todas las cadenas que pertenecen al lenguaje pertenezcan al resultado, lo cual no es posible (problema de lenguajes recursivos).

# Ejercicio 2:

***Explique cómo se realiza la especificación en herramientas “theorem provers” como KeY. Indique por qué considera que se elige esta forma de especificación.***

Los theorem provers realizan las especificaciones a través de contratos. Deben demostrar que un estado (la conjetura) es una consecuencia lógica de un conjunto de estados (axiomas e hipótesis).

El diseño por contrato permite una propagación al código de las propiedades semánticas definidas en el diseño (trasladadas al modelo). Soportan varias formas de validación para esas mismas propiedades.

Cualquier software puede verse como:

* Un conjunto de componentes que se comunican.
* Toda interacción está gobernada por contratos.
* Los contratos son especificaciones formales de obligación mutua.

Se elije esta forma de especificación ya que para probar que la fórmula es válida es posible realizar una ejecución simbólica, es decir, que siga el flujo de control natural al momento de analizar un programa.

Key es un sistema diseñado como una herramienta de métodos formales que integra el diseño, la implementación, la especificación formal y la verificación formal de la manera más transparente posible. Proporciona interfaces y herramientas que permiten a los no-especialistas usar y comprender las herramientas formales. Es necesario:

* Un lenguaje de especificación formal que es lo suficientemente expresivo para captar los requerimientos que un diseño estipula en una implementación.
* Un marco que permita demostrar formalmente que una implementación dada satisface sus requerimientos.

# Ejercicio 3:

1. ***¿Es suficiente con la lógica de primer orden para realizar la especificación? Justifique. En caso de no ser suficiente ¿qué tipo de lógica se utiliza? Descríbala.***

La lógica de primer orden no es suficiente para realizar la especificación ya que solo puede expresar propiedades estáticas: Relaciones entre objetos, valores de atributos en un cierto rango, propiedades (invariantes) de una subclase que implican propiedades de la superclase, etc. Por lo tanto, para permitir describir y razonar sobre el comportamiento de los programas (se requiere considerar no solo uno, sino que varios estados del programa) se extiende la lógica y el cálculo. Para poder trabajar con el significado de una fórmula sobre la evolución de estados será necesario aplicar la lógica dinámica. La lógica dinámica es la extensión de la FOL con interpretaciones dinámicas y programas que describen cambios de estado.

La idea es poder expresar propiedades funcionales del programa por lo que la lógica debe permitirnos:

* Relacionar los diferentes estados del programa (antes y después de la ejecución en una sola formula).
* Las variables/atributos del programa representados mediante símbolos constantes que dependen del estado del programa. Agrega dos operadores modales nuevos: <p> (correctitud total) y [p] (correctitud parcial).

Una signatura en DL esta definida de la misma manera que para FOL, pero se agregan símbolos flexibles:

* *Rígido*: Son los símbolos que tienen la misma interpretación en todos los estados del programa.

Variables de primer orden – funciones y predicados built-in.

* *Flexible*: Son los símbolos cuya interpretación depende del estado de la ejecución.

Variables del programa y atributos.

Las variables de primer orden son aquellas que están declaradas localmente en los cuantificadores (no pueden ser variables del programa).

1. ***¿Qué significan <> y [] en el contexto de la herramienta Key?***

Son operadores modales que la lógica dinámica (DL) agrega. Sea p un programa y q otra fórmula DL.

* <p>q: p termina y la fórmula q se verifica en el estado final (correctitud total)
* [p]q: si p termina entonces la formula q se verifica en el estado final (correctitud parcial).

# Ejercicio 4:

***Indique:***

* 1. ***Que opciones tiene para la verificación de ciclos repetitivos (provistas desde la especificación JML). Indique las diferencias.***

Si el numero de iteraciones de un bucle es infinito, el ciclo debe verificarse utilizando:

* Regla de invariante: Una formula invariante cuya validez sea preservada por el ciclo repetitivo, tanto por la condición. De esta manera, la invariante fue validada al comenzar el ciclo y aún lo es luego de una cantidad arbitraria de interacciones. Utilizar una invariante de ciclo es esencialmente un argumento inductivo, asegurando que la invariante se mantiene para todas las iteraciones del ciclo y, por lo tanto, se mantiene cuando el ciclo termina.
  + Correctitud total de ciclos: Se debe asegurar que el ciclo siempre termina, por lo que se utilizan las cláusulas Loop invariant, decreasing ‘x’. Donde x es una variable que decrece en cada ciclo hasta llegar a 0, con esto aseguramos que el ciclo termina.
  + Correctitud parcial de ciclos: En este caso, se permite la no terminación del ciclo. Sin embargo, en todo momento (tanto al ingresar, durante la ejecución del cuerpo y al finalizar normalmente) se debe cumplir con la invariante de ciclo. En este caso, se utilizan las cláusulas Loop invariant, con diverges true (la no-terminación está permitida). ((Diverges false especifica la condición que debe satisfacerse antes de llamar a la operación si es que la operación no termina)).
  1. ***Como se puede asegurar la ausencia de efectos colaterales en JML. Nuevamente indique las diferencias.***

En JML se puede asegurar la ausencia de efectos colaterales mediante las cláusulas:

* “pure”: Establece como obligación del implementador que el método no produzca efectos colaterales, pero permite su uso en anotaciones JML.
* “assignable\nothing”. Establece que ninguna locación puede modificarse, es decir, que bajo dicho contrato no tiene efectos colaterales, pero si se utiliza en otro entorno, puede que existan.

Estos dos se diferencian en el hecho que:

* El modificador “pure” es global al método y además prohíbe la no terminación y las excepciones. No es utilizable en contextos particulares.
* La cláusula “assignable” es local a un caso de especificación
  1. ***Por qué resulta tan crítico asegurar la ausencia de efectos colaterales.***

Es importante asegurar la ausencia de efectos colaterales para verificar que el método no modifique el valor de una variable, cuyo valor no debería ser modificado. Se puede ver como algo que está sucediendo (afecta al cómputo) pero no está en el contrato. En particular, afecta a los ciclos repetitivos.

# Ejercicio 5:

***Indique, justificando apropiadamente, los puntos fuertes y débiles de las aproximaciones para verificación formal vistas en la materia. Para cada una de las aproximaciones vistas, mencione al menos dos puntos fuertes y dos puntos débiles.***

Alloy/Model Checker:

Ventajas de Alloy:

* Soporta verificación parcial (se verifica propiedad a propiedad): No es necesario realizar un modelo elaborado para que sea posible verificarlo. En cambio, es posible desarrollar un modelo incrementalmente hasta que logremos el objetivo buscado.
* Fácil de utilizar. No es necesario utilizar matemática compleja. El lenguaje de Alloy se basa en notaciones simples de la lógica básica, y un operador espacial para navegar entre las relaciones.
* No es necesario escribir código ejecutable para obtener una muestra de estados o trazas. Es posible escribir una descripción parcial de una operación que permita varios comportamientos.
* No es necesario escribir casos de prueba.
* El analizador es completamente automático, únicamente es posible otorgarle un alcance para el análisis.

Desventajas:

* Un alcance pequeño puede omitir instancias. Sin embargo, un alcance grande puede tomar un tiempo mayor al analizar el modelo (y tiende a producir instancias grandes, menos legibles)
* Verifica un modelo del sistema. No hay nada que garantice que luego de la implementación el sistema cumplirá todas las restricciones enunciadas.
* Alcance acotado ya que posee problemas de tamaño, es decir, sufre “explosión de memoria”. Modelos reales son muy grandes para poder verificarse.
* Requiere experiencia para encontrar abstracciones apropiadas.

Key / Theorem Prover:

Ventajas:

* Se escribe en un lenguaje más cercano a la programación, lo que facilita su uso. Impulsa a que el propio programador realice el contrato, siendo lo ideal ya que es quien conoce su propio código.
* Ofrece una gran variedad de conceptos en el nivel de implementación (manejo de excepciones, cláusulas de asignación, e invariante de ciclos).
* Existen herramientas de verificación que utilizan anotaciones JML, tanto para análisis estático de código como comprobación en tiempo de ejecución.

Desventajas:

* No es una herramienta automática, sino que es semi-automática.
* Es necesario tener conocimiento para entender lo que el programa nos brinda e intervenir para otorgar el mejor caso a la hora de probar.
* En el caso de JML, está asociado al lenguaje Java. JML no está estandarizado, la mayoría de las herramientas actuales no implementan esta semántica.