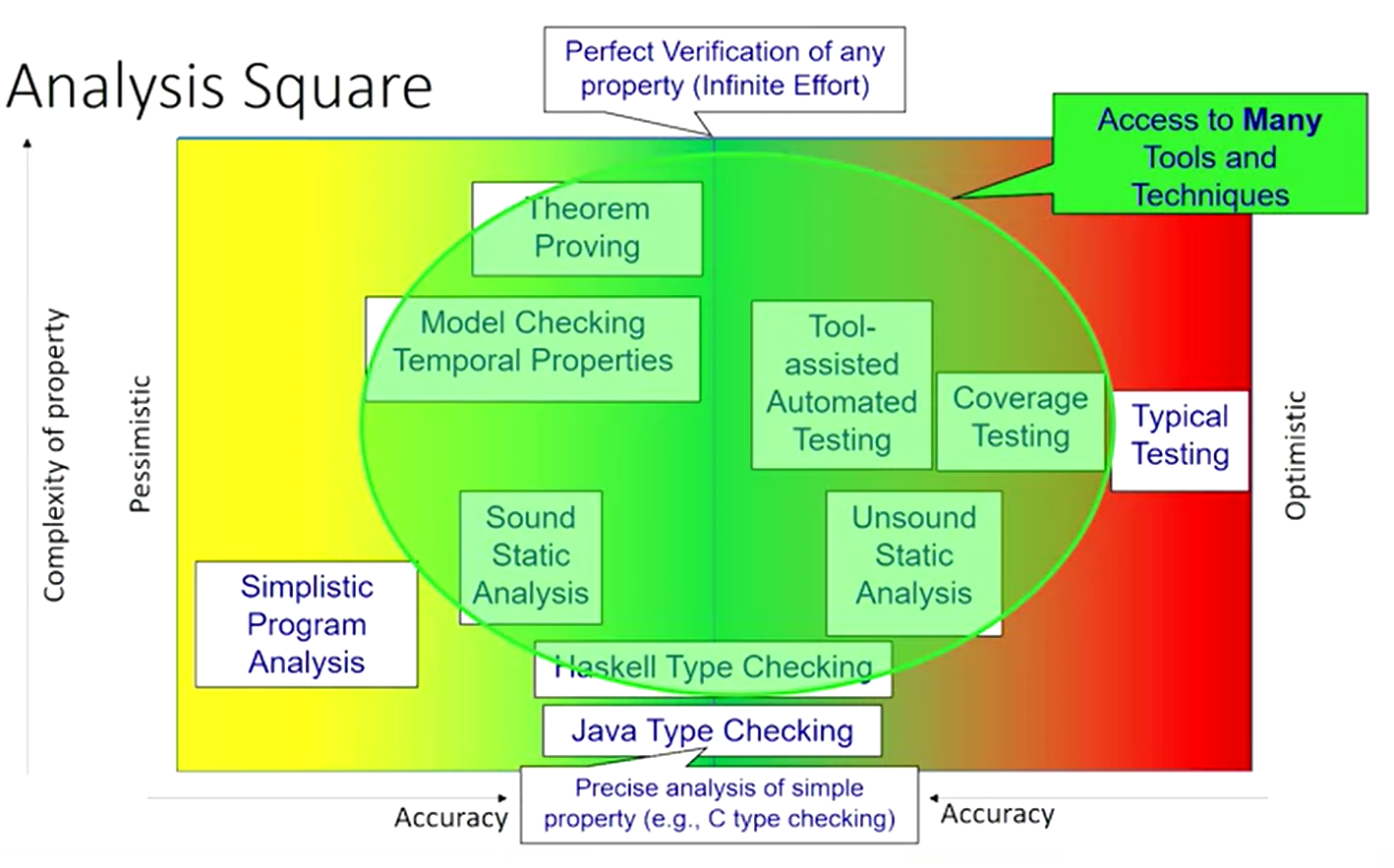
Introducción al análisis automatizado de pruebas

En este vídeo, les hablaré sobre el análisis automatizado. Es decir, incorporar la automatización como ayuda, dejar que la computadora lo ayude a probar el software que está intentando crear.

La pregunta que intentamos hacernos cuando estamos haciendo pruebas es: ¿cómo verificamos nuestro software de forma eficaz? Por eso, en los últimos cursos, hemos estado estudiando técnicas para hacer que nuestras pruebas sean más rigurosas. Por eso, hemos analizado la cobertura estructural para asegurarnos de que contamos con suficientes pruebas para asegurarnos de que hemos probado todas las líneas de código del programa. Hemos analizado técnicas para automatizar la ejecución de las pruebas, así que, una vez que las escribimos, no tenemos que volver a reproducirlas manualmente. Pero lo que realmente queremos es algo más riguroso. Queremos que la computadora nos ayude a escribir las pruebas, de modo que podamos cubrir muchos más casos de los que podríamos cubrir si tuviéramos que escribirlas a mano. Así que recordemos nuestro cuadrado de análisis.



Hacia el centro, tenemos la precisión, de manera que en el medio, tenemos una precisión perfecta, y las pruebas o análisis que realicemos siempre determinarán perfectamente si la propiedad objeto de interés es válida o no.

Por el lado de la correctitud, tenemos la visión optimista, porque a veces, las pruebas nos dicen que el sistema está bien cuando no lo está en realidad, podemos preguntar sobre propiedades interesantes del sistema, pero si no tenemos cuidado, es probable que obtengamos una respuesta que nos diga que el sistema funciona bien cuando, de hecho, todavía tiene bastantes errores, por el contrario tenemos análisis pesimistas, por lo tanto, los análisis pesimistas son aquellos que dicen que el sistema no funciona cuando en realidad funciona correctamente.

También tenemos un análisis de programas simplista que podemos realizar como por ejemplo usando el Lint, pero que muchas veces, nos dan montones y montones de advertencias, y es posible que la mayoría de esas advertencias no sean importantes para nuestro programa.

La otra cosa de la que hablamos anteriormente es la escala vertical, y tiene que ver con el tipo de propiedades que podemos pedirle a este sistema, para las pruebas, podemos preguntar sobre propiedades bastante sofisticadas, ¿cumple con nuestros requisitos? Para un análisis de programas simplista como Lint, solo podemos preguntar propiedades simples como: ¿hay alguna excepción de puntero nulo en alguna parte? Así que esa es la escala vertical, y hay algunas cosas que podemos hacer con precisión. Por lo tanto, la verificación tipográfica es algo muy simple, y solo podemos hacer preguntas sencillas, pero obtenemos respuestas exactas. ¿Podemos realizar esta operación en un número entero? Bueno, el comprobador de tipos nos lo dirá. Y a lo que nos gustaría llegar es a una verificación perfecta. Por lo tanto, nos gustaría poder hacer cualquier pregunta que deseemos sobre cualquier programa que tengamos y determinar si cumple o no con esa propiedad. Sin embargo, debido al problema de la paralización (el halting problem), sabemos que eso es indecidible en general. Así que nos vamos a centrar en este espacio intermedio, en cosas que sean bastante precisas y rigurosas.

En la parte inferior, tenemos cosas como la verificación de tipos de Java. Si utilizamos un lenguaje de programación más sofisticado como Haskell, podemos hacer preguntas más interesantes al sistema de tipos, luego analizaremos el análisis estático, lo que nos permitirá determinar de manera concluyente, en algunos casos, si el programa está bien formado o no, si hay excepciones a los punteros nulos, si se divide por ceros u otros tipos de propiedades de formato correcto que serían comunes a cualquier programa, así que nunca querrás que un número entero se desborde o se divida por cero, y puedes hacer este tipo de preguntas con las herramientas de análisis estático.

Entonces, ya hemos analizado las pruebas de cobertura, este es un intento de hacer que nuestro régimen de pruebas sea más riguroso, lo que significa que será más preciso a la hora de decirnos si el programa tiene errores o no, ahora, en este curso, vamos a empezar a analizar los enfoques automatizados para hacer preguntas interesantes sobre los programas. Y una cosa de la que voy a hablar es de la verificación por modelos de las propiedades temporales (model checking of temporal properties). Así que vamos a hacer preguntas interesantes, ¿el programa siempre cumple con sus requisitos? Y la otra cosa en la que me voy a centrar principalmente son las pruebas automatizadas asistidas por herramientas (tool-asisted automated testing). Así que, en este caso, lo que vamos a hacer es pedirle a la computadora que escriba las pruebas para nosotros y, luego, lo que podemos hacer es comprobar si, en todos esos casos de prueba, el programa se comporta según lo previsto. Y la diferencia es que, si bien un ser humano puede escribir 100 o 1000 pruebas, la computadora escriba millones de pruebas y puede hacerlo muy rápido, y muchas de esas pruebas son como millones de monos en Shakespeare. Así que no serían significativas para el ser humano, pero si puedes determinar qué debe hacer el programa en esos casos, de hecho, son muy útiles para comprobar si, en función de las condiciones límite del programa, se comporta o no según lo previsto.

Y una última cosa que podrías hacer es demostrar teoremas, en la que, de forma similar a como se hace con las demostraciones en lápiz y papel, tratas de demostrar las propiedades de tu programa utilizando otros programas que comprueban tu trabajo mientras demuestras el teorema. Pero no vamos a hablar de eso porque se requieren muchos conocimientos especializados para hacerlo bien. Por lo tanto, este curso se centrará en el análisis estático y las pruebas automatizadas asistidas por herramientas, con un poco de discusión sobre la verificación de modelos.

Cuando hablamos de automatización y rigor, sabemos que es necesario para que las pruebas de software sean eficaces. Lo hemos usado con JUnit. Lo que estamos haciendo es automatizar la ejecución de las pruebas para que cada vez que hagamos una compilación o cada vez que hagamos un registro, podamos realizar una serie de pruebas en el programa. Por lo tanto, hay muchas otras herramientas que permiten hacer esto, pero necesitamos más para ser probadores realmente eficaces. Lo que necesitamos son herramientas que automaticen la generación de pruebas. Y hemos visto, hasta cierto punto, cierta automatización en términos de Cucumber. Así que escribes tus requisitos y luego los conviertes en pruebas para ti, pero hay mucho, mucho más. Y en cuanto a la creación de entradas de prueba, veremos algunos ejemplos en los que la computadora puede crear casos de prueba que nunca se nos hubieran ocurrido. Y cuando hacemos esto, podemos, por ejemplo, obtener una mejor cobertura que la que es posible con las pruebas manuales. Podemos ejecutar de manera muy, muy rigurosa los diferentes conjuntos de declaraciones que tenemos. Y en algunos casos, utilizando herramientas simbólicas, podemos obtener una cobertura exhaustiva. Por lo tanto, cuando comprobamos el modelo, en realidad comprobamos todas las entradas posibles del sistema y determinamos si el programa hace lo correcto o no. Así que permítanme darles un ejemplo. Vamos a utilizar algo llamado generación de pruebas aleatorias con una herramienta llamada QuickCheck, y vamos a hacer una demostración en el microondas que hemos utilizado en cursos anteriores. Así que aquí hay algo que nos permite hacer muchísimas pruebas con el microondas a la vez, por así decirlo. Lo que tenemos es algo que se parece mucho a JUnit. Pero lo que hacemos es, en cambio, instanciarlo con algo llamado JUnitQuickCheck. Y lo interesante de lo que estamos haciendo aquí en este caso de prueba es que, en lugar de llamarlo prueba, lo llamamos propiedad. Y lo que eso significa es que vamos a tener una prueba que incorpore las entradas. Y el sistema que estamos usando, JUnitQuickCheck, intentará proporcionar aleatoriamente entradas dentro de los rangos que proporcionamos para ejecutar esta prueba.

Así que lo que estamos haciendo aquí en esta prueba en particular es comprobar cómo se actualiza la pantalla. Y lo que queremos hacer es que, en trabajos anteriores, dijimos que permitiera al usuario elegir un par de combinaciones de valores de entrada. Por lo tanto, puede cocinar durante 20 segundos o 30 segundos, pero sabemos que el microondas puede cocinar hasta 99 minutos y 99 segundos. Si presionas 9999 en la pantalla, ese es el tiempo máximo que puede cocinarse. Así que lo que queremos hacer es probar todo tipo de entradas diferentes en ese espacio. Así que podemos hacerlo con JUnitQuickCheck. Y lo que vamos a hacer es, una vez que obtengamos los dígitos que describen cuánto tiempo hay que cocinar, diremos, bueno, ¿cuánto tiempo ha transcurrido? Así que encenderemos el microondas, dejaremos que se cocine durante un tiempo determinado, que también es variable, y luego comprobaremos si la pantalla coincide con nuestras expectativas. Entonces, lo que hacemos es tomar la entrada que describe el tiempo de cocción y la dividimos en dígitos, y usamos nuestra prueba de pepino que hemos escrito anteriormente, luego presionamos la tecla de inicio y dejamos que el microondas se cocine durante un período de tiempo aleatorio. Y luego, lo que deberíamos ver en el tiempo restante es que la pantalla coincide con el tiempo original menos el tiempo transcurrido. Y así, también podemos comprobar el modo del microondas. Si el tiempo restante llega a cero, entonces deberíamos volver al modo de configuración. De lo contrario, deberíamos estar en el modo de cocción. Así que he configurado esto, donde los valores de cuatro dígitos están entre 0 y 9999, el tiempo transcurrido puede prolongarse hasta llegar a 11.000. Así que deberíamos llegar al caso cero en el que el microondas deja de cocinar. Y he dicho que lo ejecutes 40 veces. Así que la herramienta se activará y hará esto por nosotros. Muy bien. Ahora lo encenderemos. Así que dirigiré una construcción.

Y lo que puede ver es que el proceso de ejecución lleva bastante más tiempo del que estábamos acostumbrados para nuestras pruebas de JUnit. Pero lo interesante es que encontró un error. Y la realidad de la situación es que en este microondas, probé unas tres docenas de pruebas escritas manualmente, y no encontré ningún error y pensé que estaba libre de errores. Así que, cuando empecé a ejecutar Quickcheck en él, descubrí un nuevo error que no había previsto y pensé que el sistema estaba bastante bien probado. Así que echemos un vistazo a lo que encontramos aquí.

En la carpeta de compilación, si miro los informes, lo que veré es que el 99 por ciento de las pruebas se ejecutaron correctamente, pero tenemos una prueba fallida. Y ahora puede ver cuál es el resultado de ejecutar este JUnitQuickCheck. Y lo que vemos es que el sistema no logra hacer lo que se esperaba cuando cocinamos durante mucho tiempo. Así que 42 minutos 79 segundos, y se supone que hay que esperar 3.291 segundos antes de comprobar el resultado. Así que, claramente, este valor es menor que este valor. Entonces, todavía debería estar en el modo de cocción, pero de alguna manera estoy en el modo de configuración. Así que algo salió mal. Ahora, la parte difícil es que todavía tengo que entrar y averiguar qué salió mal. Pero en este caso, voy a cortocircuitar eso. Y voy a ir al archivo que es un error. Y lo que había hecho era no pasar los dígitos en una configuración determinada en la que, básicamente, el dígito de las decenas de minutos no aparecía al mismo ritmo. Por lo tanto, necesitaba convertir esta «i > 0" en una «i >= 0". Y ahora, cuando vuelvo a ejecutar, estoy haciendo muchísimas pruebas con esta función en particular, y puedes ver que pasa. Y puedo hacer no solo 40 pruebas, sino que podría hacer 100 pruebas, o 1000 pruebas, o tantas como considere que tengo tiempo para aumentar el nivel de rigor y asegurarme de haber analizado más casos de prueba. Así que este es solo un ejemplo, y en realidad es el ejemplo más simple, de lo que las pruebas automatizadas podrían hacer por ti.

En resumen, para que las pruebas sean rigurosas, necesitamos automatizar la generación y la ejecución de las pruebas. Y nada más sacarlo de la caja, sin necesidad de crear un oráculo, algo que hice allí como un oráculo un poco complicado, las pruebas automatizadas pueden detectar excepciones y errores de bloqueo. Así que, si no me molestaba en decir cuál es el valor de visualización correcto, aún podría hacer un montón de pruebas aleatorias para asegurarme de que el programa no fallaba en ningún caso. Pero si queremos que las pruebas automatizadas sean más eficaces, necesitamos tener una idea de los requisitos de los ejecutables. Por eso, dedicaremos bastante tiempo a lo largo de este curso a describir cómo crear esos requisitos y cómo realizar esas pruebas automatizadas.

​

**1.**Pregunta 1

Cuando todas las pruebas de un conjunto de pruebas pasan aunque el software contenga errores, es un ejemplo de verificación \_\_\_\_\_\_.



Verificación optimista



Verificación pesimista

Correcto

Sí, la verificación optimista se produce cuando un conjunto de pruebas "pasa" pero siguen existiendo fallos en el software.

**2.**Pregunta 2

¿Por qué no es práctica la validación perfecta?

El problema de detención; es indecidible.

No es en absoluto poco práctico. La validación perfecta se realiza regularmente con sistemas de seguridad crítica.

Correcto

Esto es correcto. Debido al problema de detención, no podemos realizar una validación perfecta.

**3.**Pregunta 3

Marque todas las que correspondan: El análisis estático determina si un programa está bien formado. Esto incluye la comprobación de:

Dividir por cero excepciones

Excepciones de puntero nulo

Desbordamiento de enteros

Pruebas de cobertura

**4.** Pregunta 4

La ejecución automatizada de las pruebas es importante para que las pruebas de software sean eficaces.

Verdadero

Falso

Incorrecto

Si escribimos a mano múltiples pruebas para el software, sin duda podremos encontrar fallos e incluso probar algunas condiciones límite. El problema es que sólo podemos escribir a mano un número limitado de pruebas. La ejecución automatizada de pruebas proporciona una gama mucho más amplia de pruebas en un número mucho mayor. Esto puede detectar cosas que se pasan por alto con las pruebas manuales.

**5.**Pregunta 5

Rellene los espacios en blanco. Para que las pruebas sean rigurosas, necesitamos automatizar la \_\_\_\_\_\_\_\_\_\_\_\_\_\_ y \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ de las pruebas.

Generación; Ejecución

Ejecución; Rigor

Ejecución; Efectiva

Correcto