MODEL CHECKING

Explique brevemente como realiza a nivel conceptual, un MC la verificación de una propiedad.

Model checking 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.

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.

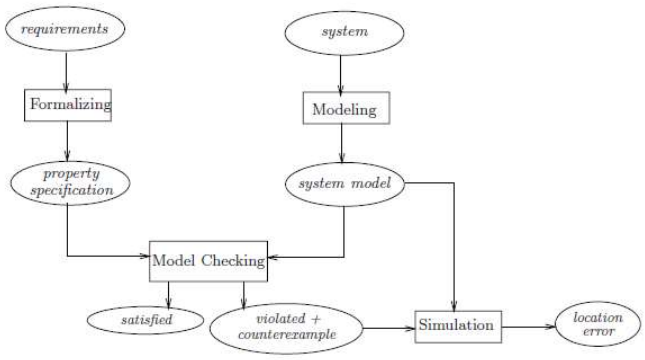

Figure 1.4: Schematic view of the model-checking approach.

Explique por qué razón la lógica de primer orden (FOL) no resulta suficiente para especificar las propiedades a verificar por este tipo de herramienta.

No resulta suficiente porque la FOL permite hacer referencia a un estado en particular, el estado actual. Con la FOL se evalúa estáticamente en esa situación cuál es el valor de verdad de los literales que se usa. Y la idea es asumir una lógica temporal (LT) de forma que sea posible verificar si las propiedades deseables del programa, expresada como fbf de esa lógica, se satisface siempre, o al menos eventualmente. Es decir, es necesario hacer referencia a varios estados de la ejecución.

Explique qué ventajas y desventajas observa en el uso de una herramienta como JSpin. Utilice como referencia su experiencia de uso durante el cursado.

Ventajas:

- Es una aproximación general a la verificación que es aplicable a un rango amplio de aplicaciones.
- Soporta verificación parcial (se verifica propiedad a propiedad)
- No es vulnerable a la exposición del error
- Provee información de diagnostico
- No requiere mucha interacción por parte del usuario
- Fácil integración en los ciclos de desarrollo existentes.

Desventajas:

- Más apropiado para aplicaciones centradas en control
- Su aplicabilidad queda sujeta a aspectos de decidibilidad
- Verifica un modelo del sistema, no el sistema en sí mismo
- No garantiza completitud, pues no se puede afirmar nada sobre las propiedades no chequeadas.
- Sufre "explosión de memoria". Modelos reales son muy grandes para "caber"
- Requiere experiencia para encontrar abstracciones apropiadas
- No permite chequear generalizaciones, por ejemplo, sistemas con tipos parametrizados.

MODEL PROVER

Explique brevemente cómo se realiza la especificación en herramientas que siguen esta aproximación (ej: Key)

Para probar que una fórmula es válida, es decir, se satisface en todos los estados se debe seguir una ejecución simbólica, la cual sigue el flujo de control natural al momento de analizar el 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.

¿Es suficiente con la FOL para realizar la especificación?

NO es suficiente, ya que la FOL nos sirve para describir y razonar sobre ED, relaciones entre objetos, valores de variables pero no así sobre los estados de programas. La idea es extender la lógica y el cálculo para describir y razonar sobre el comportamiento de los programas. Para poder trabajar con el significado de una fórmula sobre la evolución de estados será necesario aplicar la lógica dinámica. Esta, extiende a la FOL con interpretaciones dinámicas, y programas que describen cambios de estados. Nos permite:

- Relacionar diferentes estados del programa
- Las variables/atributos del programa representados mediante símbolos constantes que dependen del estado del programa. Agrega dos operadores modales nuevos: <p> y [p].

En el contexto de Theorem Prover, ¿Se definen propiedades como en los MC? ¿Qué es lo que se verifica?

En los Theorem Prover no se definen propiedades, sino que se verifican especificaciones como contratos. Hacen énfasis en las responsabilidades en los roles y/o obligaciones en una especificación. Se habla, entonces, de una metodología de diseño por contrato.

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. Los contratos deben describirse en un lenguaje preciso y matemático.

Un par de pre/post condiciones para un método *m* se satisface para la implementación de *m* si: cuando *m* es llamado en cualquier estado que satisface la precondición, entonces en cualquier estado de terminación de *m* la post-condición es verdadera.

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

¿Cuál es la importancia de asegurar la ausencia de efectos colaterales? ¿Cómo se puede asegurar en JML?

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

En JML se puede asegurar mediante “pure” y “assignable\nothing”. Este último, establece que ninguna locación puede modificarse, es decir, que en ese punto no tiene efectos colaterales, pero si se utiliza en otro lado, puede que haya. “Pure” establece como obligación del implementador que el método no produzca efectos colaterales, pero permite su uso en anotaciones JML. Es similar al anterior, pero global a métodos. 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

¿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).

Utilizando como punto de partida su experiencia, ventajas y desventajas de la herramienta

Ventajas:

- Es muy cercano al código Java, lo que facilita su uso
- 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 de JML:

- Está asociado al lenguaje Java
- No está estandarizado, la mayoría de las herramientas actuales no implementan la semántica de JML.