Tema 5. Tecnologías de soporte

Lenguajes, Tecnologías y Paradigmas de Programación (LTP)

DSIC, ETSInf

Objetivos del tema

- Conocer técnicas basadas en la semántica para el análisis, testeo y verificación de programas.
- Comprender el impacto de las tecnologías de soporte para el desarrollo de *software*
- Identificar los ingredientes para la automatización de procesos de *software*

Desarrollo de *software* de calidad

- Para garantizar la calidad del *software* desarrollado es **imprescindible** usar herramientas y tecnologías que den soporte (automático) a distintos procesos.
- Podemos diferenciar dos tipos de estrategias:
  - De *prevención*: construimos *software* de calidad usando metodologías adaptadas, invirtiendo en un buen diseño, integrando el análisis, *testing* y depuración desde las etapas tempranas de desarrollo, etc.
  - De *corrección*: cuando se detectan errores o disfunciones, se aplican técnicas de análisis, verificación, *testing*, depuración, métodos formales, etc.

Desarrollo de *software* de calidad

- Para garantizar la calidad del *software* desarrollado es **imprescindible** usar herramientas y tecnologías que den soporte (automático) a distintos procesos.
- Podemos diferenciar dos tipos de estrategias:
  - De *prevención*: construimos *software* de calidad usando metodologías adaptadas, invirtiendo en un buen diseño, integrando el análisis, *testing* y depuración desde las etapas tempranas de desarrollo, etc.
  - De *corrección*: cuando se detectan errores o disfunciones, se aplican técnicas de análisis, verificación, *testing*, depuración, métodos formales, etc.

Más en...

ISW (3A) y/o en la rama de Ingeniería del Software (MFI, AVD, . . .)

Tecnologías de soporte

Las tecnologías y herramientas que dan soporte al desarrollo de *software*:

- Están **basadas en la semántica** de los lenguajes
- Pueden ser estáticas (tiempo de compilación) o dinámicas
- Pueden ser métodos formales o no
- Pueden ser automáticas o semiautomáticas

Tecnologías de soporte

Aplicaciones

- Comprobar que el sistema satisface los requisitos del usuario (hace lo que debe),
- garantizar que no ocurren errores inesperados en tiempo de ejecución,
- analizar el rendimiento del sistema (tiempo de respuesta, consumo de recursos, etc.),
- optimizar el código de forma automática,
- ...

Tecnologías de soporte

Ejemplos (*Software testing*)

Software testing

- Objetivo: detectar posibles fallos en el código
  - No garantizar la ausencia de fallos
- se ejecuta un conjunto de casos de prueba
  - No es exhaustivo
- lo más complicado: diseñar las pruebas
  - existen métodos para la generación automática de pruebas, por ejemplo usando ejecución simbólica

Tecnologías de soporte

Ejemplos (*Software testing*)

Pasos del *software testing*

- Diseño de los casos de prueba
- Ejecución de las pruebas
- Evaluación del resultado
  - Si se encuentra un error: corrección y volver a probar
  - Si no se encuentra error alguno: ¿hemos hecho suficientes pruebas?
- Gestión de los casos de prueba

- La fase más costosa es la de diseñar los casos de prueba

Tecnologías de soporte

Ejemplos (*Static analysis*)

Análisis estático de programas

Técnicas para predecir *en tiempo de compilación* el comportamiento dinámico (en tiempo de ejecución) del programa.

- Es un problema indecidible: debemos renunciar a ser totalmente precisos para tener un método efectivo.
- Basado en la semántica

Tradicionalmente se usa para:

- Optimizaciones de los compiladores
- Verificación
- Integrados en los IDEs

Tecnologías de Soporte

Ejemplos (*Static analysis*)

Análisis estático de programas.

- Tipo de análisis: Propagación de constantes
- Objetivo: optimización de código

```
y:=4;
x:=1;
while (y>x) do
      (z := y;
       x := y*y);
x:=z
```

Análisis:

Para cada instrucción: ¿tienen las variables usadas en ese punto un valor constante?

Tecnologías de Soporte

Ejemplos (*Static analysis*)

Análisis estático de programas.

- Tipo de análisis: Propagación de constantes
- Objetivo: optimización de código

```
y:=4;
x:=1;
while (y>x) do
      (z := y;
       x := y*y);
x:=z
```

- la variable `y` es constante en el bucle
- la variable `x` NO es constante en el bucle

Tecnologías de Soporte

Ejemplos (*Static analysis*)

Análisis estático de programas.

- Tipo de análisis: Propagación de constantes
- Objetivo: optimización de código

```
y:=4;
x:=1;
while (4>x) do
      (z := 4;
       x := 4*4);
x:=z
```

- podemos optimizar el código evitanto accesos a memoria y cómputos innecesarios

Tecnologías de soporte

Ejemplos (Depuración)

Depuración

- Principal objetivo: detección de errores
- Alternativas:
  - a mano: *tracers*, *print debugging*
  - dirigida: depuración algorítmica
  - automática: asertos, *abstract debugging*

- La aproximación más extendida es la manual o semi-automática

Ejecución simbólica

Muchas de las herramientas de soporte tienen como base la ejecución simbólica, basada en la semántica ***small-step*** del lenguaje.

Idea

- Se usan valores simbólicos como argumentos de entrada, de forma que
- ... se construye un árbol que representa todas las posibles ejecuciones del programa,
- ... asociando a cada ejecución cuáles son las condiciones que deben satisfacer los valores de entrada de los argumentos para seguir ese camino.

Ejecución simbólica *vs* concreta

Ejemplo

- Asumamos que $X1$ y $X2$ son parámetros de entrada del programa

  if $X1 > X2$ **then** $X1 := X1 - X2$
  else $X2 := X2 - X1$

- Ejecución concreta. Trazas posibles:

$$\langle \textbf{if } \ldots, \{X1 \mapsto 3, X2 \mapsto 6\} \rangle \rightarrow \ldots \rightarrow \langle \textbf{skip}, \{X1 \mapsto 3, X2 \mapsto 3\} \rangle$$
$$\langle \textbf{if } \ldots, \{X1 \mapsto 3, X2 \mapsto 8\} \rangle \rightarrow \ldots \rightarrow \langle \textbf{skip}, \{X1 \mapsto 3, X2 \mapsto 5\} \rangle$$
$$\langle \textbf{if } \ldots, \{X1 \mapsto 4, X2 \mapsto 2\} \rangle \rightarrow \ldots \rightarrow \langle \textbf{skip}, \{X1 \mapsto 2, X2 \mapsto 2\} \rangle$$
$$\vdots$$

Ejecución simbólica *vs* concreta

Ejemplo

- Asumamos que $X1$ y $X2$ son parámetros de entrada del programa

  if $X1 > X2$ **then** $X1 := X1 - X2$
  else $X2 := X2 - X1$

- Ejecución simbólica. Árbol de ejecución:

Valores simbólicos

path condition

$\langle \textbf{if}\ X1{>}X2\ \ldots, \{X1 \mapsto ?X1, X2 \mapsto ?X2\}, \mathit{true} \rangle$

$\langle X1 := X1{-}X2, \{X1 \mapsto ?X1, X2 \mapsto ?X2\}, ?X1{>}?X2 \rangle$

$\langle \textbf{skip}, \{X1 \mapsto ?X1{-}?X2, X2 \mapsto ?X2\}, ?X1{>}?X2 \rangle$

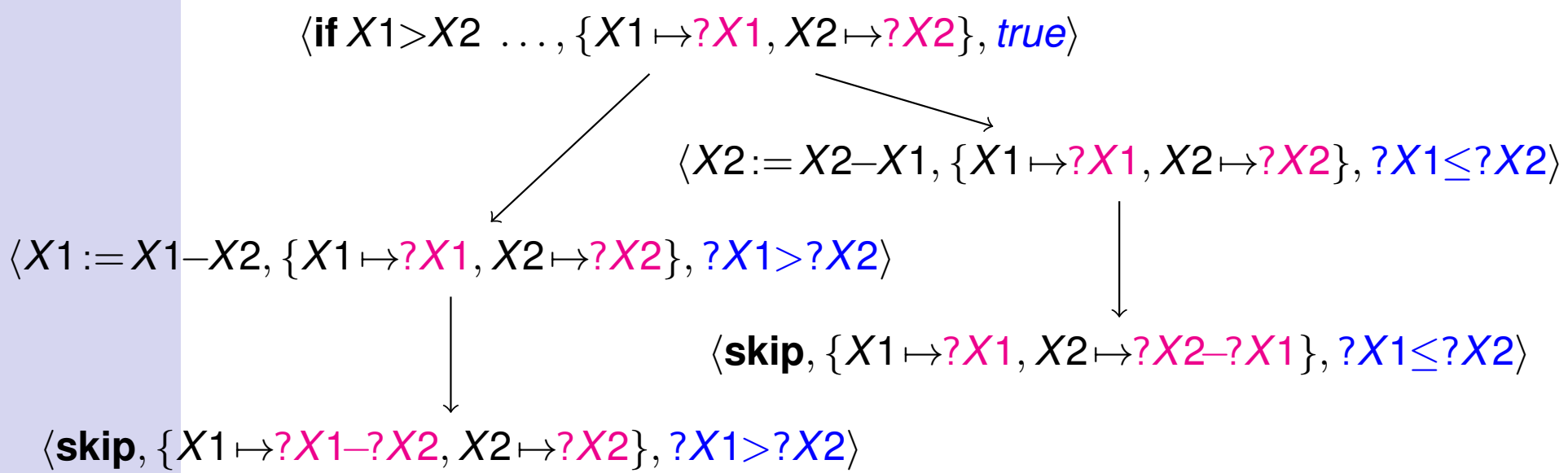
Ejecución simbólica *vs* concreta

Ejemplo

- Asumamos que $X1$ y $X2$ son parámetros de entrada del programa

  if $X1 > X2$ **then** $X1 := X1 - X2$
  else $X2 := X2 - X1$

- Ejecución simbólica. Árbol de ejecución:



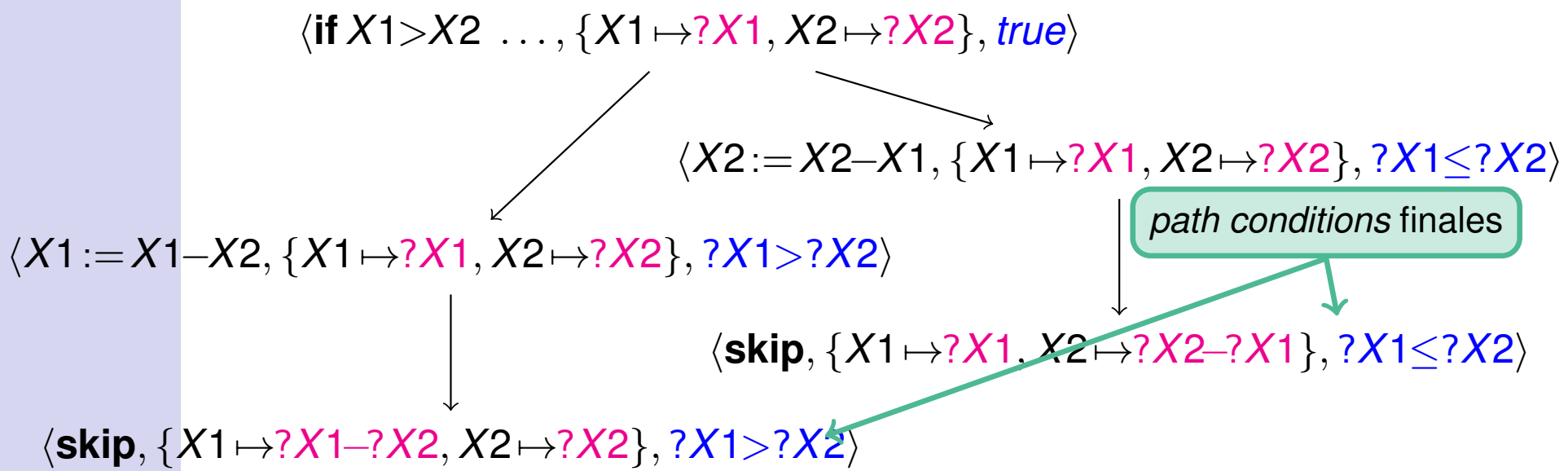
Ejecución simbólica *vs* concreta

Ejemplo

- Asumamos que $X1$ y $X2$ son parámetros de entrada del programa

  if $X1 > X2$ **then** $X1 := X1 - X2$
  else $X2 := X2 - X1$

- Ejecución simbólica. Árbol de ejecución:



Ejecución simbólica para SIMP

- Una configuración de la máquina concreta consistía en un par:

$$\langle instr, e \rangle$$

  - un estado e es una función que asigna valores a las variables, por ejemplo $\{X \mapsto 0, Y \mapsto 5\}$.
- Una configuración de la máquina simbólica consiste en una terna:

$$\langle instr, se, pc \rangle$$

  - un estado simbólico se es una función que asigna a cada variable *una expresión simbólica*, por ejemplo $\{X \mapsto ?X, Y \mapsto ?X + ?Y\}$.
  - Una *path condition* pc es una condición booleana sobre los valores simbólicos iniciales de los parámetros de entrada.

Semántica operacional simbólica para SIMP

- Muy parecida a la semántica *small-step* vista en el Tema 2
- Estado inicial: se asigna un **valor simbólico** (denotado con un ? al principio) inicial a cada variable de entrada
- Cuando se alcanza un punto de bifurcación (condicional o bucle), se acumula la condición en la *path condition* correspondiente

Semántica operacional simbólica

Evaluación de expresiones simbólicas

- Evaluación de expresiones:
  $\langle a, se \rangle \Rightarrow$ *sexp* representa la evaluación simbólica de la expresión aritmética a.
  Se tratará de evaluar cada una de las expresiones en a y usar el valor (simbólico) en la expresión. Si hay valores concretos se usarán para simplificar la expresión.

Ejemplos:

- $\langle X + Y, \{X \mapsto ?X, Y \mapsto 3\} \rangle \Rightarrow \ ?X + 3$
- $\langle X + Y, \{X \mapsto 5, Y \mapsto 3\} \rangle \Rightarrow \ 8$
- $\langle (X * X) + (Y * Y), \{X \mapsto ?X, Y \mapsto 3\} \rangle \Rightarrow \ (?X * ?X) + 9$

Semántica operacional simbólica

Reglas de ejecución

- Secuencia:

$$\frac{}{\langle \textbf{skip}; i_1, se, p\rangle \rightarrow \langle i_1, se, p\rangle}$$

$$\frac{\langle i_0, se, p\rangle \rightarrow \langle i_0', se', p'\rangle}{\langle i_0; i_1, se, p\rangle \rightarrow \langle i_0'; i_1, se', p'\rangle}$$

- Asignación:

$$\frac{\langle a, se\rangle \Rightarrow sexp}{\langle X := a, se, p\rangle \rightarrow \langle \textbf{skip}, se[X \mapsto sexp], p\rangle}$$

En general, el resultado calculado en *sexp* no será un valor sino una expresión simbólica.

Semántica operacional simbólica

Reglas de ejecución

- Condicional: Se genera una bifurcación en el árbol mediante la ejecución de las siguientes dos reglas:

$$\frac{\langle b, se\rangle \Rightarrow sb}{\langle \textbf{if } b \textbf{ then } i_0 \textbf{ else } i_1, se, p\rangle \rightarrow \langle i_0, se, p \wedge sb\rangle}$$

$$\frac{\langle b, se\rangle \Rightarrow sb}{\langle \textbf{if } b \textbf{ then } i_0 \textbf{ else } i_1, se, p\rangle \rightarrow \langle i_1, se, p \wedge \neg sb\rangle}$$

Semántica operacional simbólica

Reglas de ejecución

- Condicional: Se genera una bifurcación en el árbol mediante la ejecución de las siguientes dos reglas:

$$\frac{\langle b, se\rangle \Rightarrow sb}{\langle \textbf{if } b \textbf{ then } i_0 \textbf{ else } i_1, se, p\rangle \rightarrow \langle i_0, se, p \wedge sb\rangle}$$

$$\frac{\langle b, se\rangle \Rightarrow sb}{\langle \textbf{if } b \textbf{ then } i_0 \textbf{ else } i_1, se, p\rangle \rightarrow \langle i_1, se, p \wedge \neg sb\rangle}$$

- Podemos usar un *motor lógico* para comprobar si $p \wedge sb$ (y $p \wedge \neg sb$) son satisfacibles. Así podemos *podar* ejecuciones imposibles.

Semántica operacional simbólica

Reglas de ejecución

- Bucle: Se aplican las dos siguientes generándose una bifurcación. (Similar al condicional)

$$\frac{\langle b, se\rangle \Rightarrow\ sb}{\langle \textbf{while}\ b\ \textbf{do}\ i, se, p\rangle \rightarrow\ \langle \textbf{skip}, se, p \wedge \neg sb\rangle}$$

$$\frac{\langle b, se\rangle \Rightarrow\ sb}{\langle \textbf{while}\ b\ \textbf{do}\ i, se, p\rangle \rightarrow\ \langle i; \textbf{while}\ b\ \textbf{do}\ i, se, p \wedge sb\rangle}$$

Semántica operacional simbólica

Reglas de ejecución

- Bucle: Se aplican las dos siguientes generándose una bifurcación. (Similar al condicional)

$$\frac{\langle b, se\rangle \Rightarrow\ sb}{\langle \textbf{while}\ b\ \textbf{do}\ i, se, p\rangle \rightarrow\ \langle \textbf{skip}, se, p \wedge \neg sb\rangle}$$

$$\frac{\langle b, se\rangle \Rightarrow\ sb}{\langle \textbf{while}\ b\ \textbf{do}\ i, se, p\rangle \rightarrow\ \langle i; \textbf{while}\ b\ \textbf{do}\ i, se, p \wedge sb\rangle}$$

¡Atención!

El bucle puede hacer que el árbol sea infinito. Ocurrirá si existen infinitas posibles ejecuciones (un número arbitrario de iteraciones del bucle)

Ejercicio

Escribe el árbol resultante de la ejecución simbólica del siguiente programa:

$$\textbf{if } X > 3 \textbf{ then } (Y := 2; X := X - 2) \textbf{ else } Y = 6$$

Ejecución simbólica

Aplicación: Generación de casos de prueba

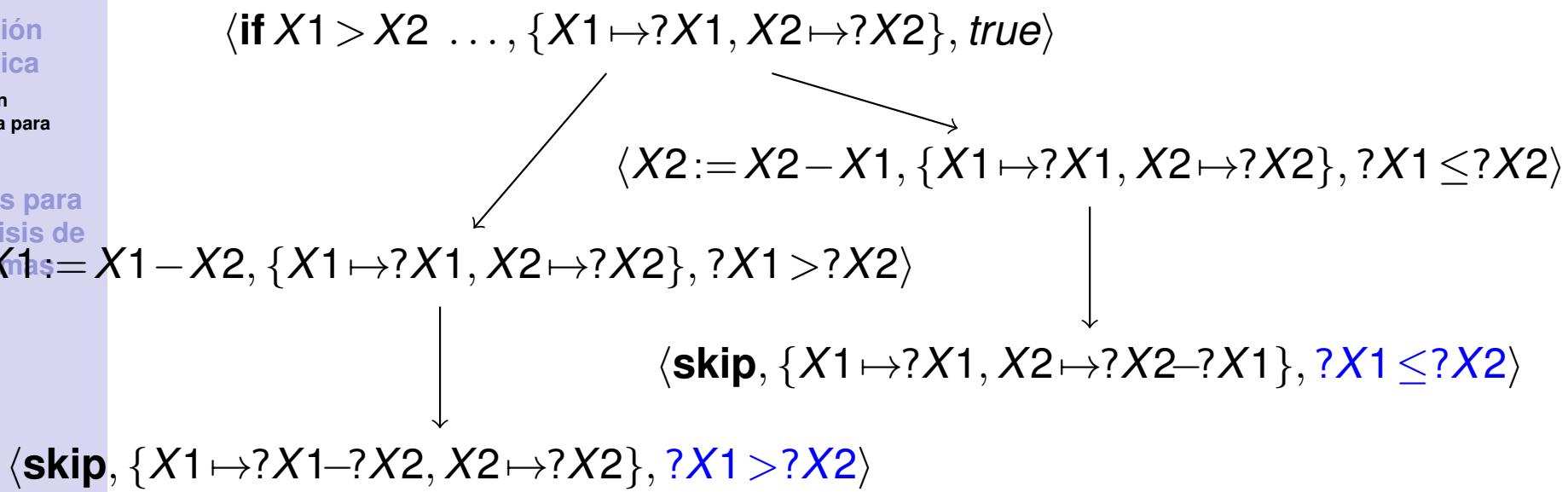
- Podemos usar la ejecución simbólica para generar automáticamente casos de prueba de un programa

Idea

1. Si el árbol es infinito, se acota. Las dos formas tradicionales de hacerlo son:
   - Estableciendo límite de profundidad del árbol, o
   - Estableciendo límite de iteraciones de los bucles
2. Se extraen las *path conditions* de las hojas del árbol, las llamamos *path conditions* **finales**
3. Se usa un motor lógico para calcular los casos de prueba: serán valores de las variables que hacen cierta cada *path condition*

Ejecución simbólica

Aplicación: Generación de casos de prueba



- Si *extraemos* las *path conditions* finales y encontramos valores para $?X1$ y $?X2$ que las satisfagan, tendremos casos de prueba para todos los caminos explorados.

  Caso 1: $?X1 = 2, ?X2 = 1$

  Caso 2: $?X1 = 2, ?X2 = 2$

Ejecución simbólica

Aplicación: Generación de casos de prueba

La importancia de la automatización

- Automatizar permite ahorrar tiempo de desarrollo
- Definir los casos de prueba puede ser muy costoso
- Por eso existen herramientas que, usando ejecución simbólica, generan de forma automática los casos de prueba

Ejemplo

Define casos de prueba que cubran todas las sentencias

if $(X1 >= X2)\&\&(X3 < X2 + X1)$
  then if $(X1 < 256)$ **then** $X1 := X2/(X1 - X2)$
    else $X1 := 7$
  else $X1 := 0$

Herramientas de soporte al testing

Basadas en ejecución simbólica

- Klee: proyecto de la University of Illinois UC
  - generación de casos de prueba
  - para C
  - https://klee.github.io/
  - instalada en los laboratorios del DSIC (linux)
  - Demostración de uso en PoliformaT
- Java Pathfinder: proyecto de la NASA con muchas funcionalidades, pero en particular:
  - generación de casos de prueba (en formato JUnit)
  - para Java
  - https://github.com/javapathfinder/jpf-core/wiki
  - Demostración de uso en PoliformaT

Otras herramientas

Basadas en ejecución simbólica

- KeY: Herramienta de análisis para Java
  - Usa anotaciones JML
  - Deducción basada en ejecución simbólica
  - Generación de casos de prueba
  - Depurador basado en ejecución simbólica
  - http://www.key-project.org/

Herramientas de soporte al testing

Basadas en otras tecnologías

- PEX: herramienta de Microsoft
  - Generación automática de casos de prueba
  - Para .NET
  - http://research.microsoft.com/en-us/projects/pex/
- QuickCheck: Original de Chalmers University pero exportado a otros lenguajes y modelos de negocio
  - Generación aleatoria basada en la especificación de propiedades
  - para Haskell, posteriormente adaptado a otros lenguajes como C, Java, JavaScript, Erlang, etc. (ver https://en.wikipedia.org/wiki/QuickCheck)
  - https://hackage.haskell.org/package/QuickCheck

Análisis de las propiedades de un programa

Especificación y propiedades

Cuando hablamos de analizar o verificar un programa, tenemos que tener presente que lo haremos con respecto a la especificación de una propiedad. Ejemplos de propiedad:

- las ternas de Hoare vistas con la semántica axiomática establecían una propiedad basada en pre- y post-condiciones
- para programas concurrentes (y/o reactivos) podemos especificar propiedades como *ausencia de deadlocks*, o *necesidad de que una máquina responda ante un evento como la pulsación de un botón*
- podemos tener métodos que implementan la comprobación de una propiedad concreta, como *ausencia de punteros null* o *ausencia de divisiones por cero*

Análisis de las propiedades de un programa

Técnicas de análisis

Existen numerosas aproximaciones al análisis de programa. Destacamos dos ejemplos:

- En *model checking*: especificamos una propiedad sobre el comportamiento temporal de un programa (por ejemplo, si un programa concurrente llegará alguna vez a un *deadlock*), y el análisis verifica si el programa cumple esa propiedad.

- Mediante la inclusión de asertos (fórmulas de alguna lógica) en el código, existen herramientas que comprueban si dichos asertos se violan en tiempo de compilación (o de ejecución).

Análisis de las propiedades de un programa

Técnicas de análisis

Existen numerosas aproximaciones al análisis de programa. Destacamos dos ejemplos:

- En *model checking*: especificamos una propiedad sobre el comportamiento temporal de un programa (por ejemplo, si un programa concurrente llegará alguna vez a un *deadlock*), y el análisis verifica si el programa cumple esa propiedad.
  - Más en MFI (rama de IS)
- Mediante la inclusión de asertos (fórmulas de alguna lógica) en el código, existen herramientas que comprueban si dichos asertos se violan en tiempo de compilación (o de ejecución).
  - Vemos dos ejemplos a continuación

Análisis de las propiedades de un programa

Análisis mediante el uso de asertos

- Recordemos: Los análisis deben estar siempre basados en una semántica formal
- Usando una semántica axiomática podemos
  - analizar las propiedades de los programas
  - usar información sobre esas propiedades para mejorar otras técnicas de análisis, verificación o testing

JML: Java Modeling Language

- Se trata de una notación, que luego puede ser usada por distintas herramientas, las cuales pueden ser
  - dinámicas (*runtime assertion checking*):
    - comprobación *runtime*: `jmlc` (`http://www.eecs.ucf.edu/~leavens/JML-release/`)
    - generación de casos de prueba: `jmlunit` (`http://www.eecs.ucf.edu/~leavens/JML-release/docs/man/jmlunit.html`)
  - estáticas (*static verification*):
    - comprobación de aserciones: `ESC/Java2` (`http://kindsoftware.com/products/opensource/ESCJava2/`),
    - basado en cálculo de precondición más débil: `JACK` (`http://www-sop.inria.fr/everest/soft/Jack/jack.html`),
    - verificación deductiva: `Krakatoa` (`http://krakatoa.lri.fr/`)

JML: Java Modeling Language

- Incorpora la posibilidad de especificar propiedades relacionadas con conceptos como *aliasing*, herencia, *side effects*, etc.

Las anotaciones JML se introducen como comentarios especiales en los programas Java:

```
//@ <JML specification>
/*@ <JML specification> @*/
```

JML: Java Modeling Language

Notación

keyword	uso
`requires`	precondición
`ensures`	postcondición
`assert`	aserción
`pure`	método libre de *side effects*
`invariant`	invariante de la clase
`loop_invariant`	invariante del bucle
`signals`	postcondición cuando hay excepción
`signals_only`	excepciones que pueden darse dada una precondición
`assignable`	campos asignables por los métodos
`also`	para combinar especificaciones
`spec_public`	hace pública (a la espec.) una variable

JML: Java Modeling Language

Expresiones disponibles:

expresión	significado
`\result`	valor devuelto por el método
`\old(<expression>)`	valor de la expresión a la entrada del método
`a ==> b`	implicación
`a <== b`	`b` implica a `a`
`a <==> b`	si y solo si

Además, las cuantificaciones universal y existencial:

expresión
`(\forall <decl>; <range-exp>; <body-exp>;)`
`(\exists <decl>; <range-exp>; <body-exp>;)`

JML: Java Modeling Language

Ejemplo 1

```
public class TickTockClock {

  ...some code here ...

  //@ protected invariant 0<=second && second<=59;
  protected int second;

  ...some code here ...

  //@ ensures 0 <= \result;
  //@ ensures \result <= 59;
  public /*@ pure @*/ int getSecond() {
    return second;
    }
```

JML: Java Modeling Language

Ejemplo 2

```
public class BankingExample {
  public static final int MAX_BALANCE = 1000;
  private int balance;

  //@ private invariant balance>=0 && balance<=MAX_BALANCE;

  //@ ensures balance == 0;
  public BankingExample() { balance = 0; }

  //@ requires 0<amount && amount + balance<MAX_BALANCE;
  public void credit(int amount) { balance += amount;}

  //@ requires 0<amount && amount <= balance;
  public void debit(int amount) { balance -= amount;}
  }
```

Dafny: análisis en .NET

Notación similar a JML+análisis

- Herramienta para el análisis estático de programas
- Se usa un lenguaje de especificación para proporcionar asertos, precondiciones, postcondiciones, ...
- Las especificaciones se usan en la fase de verificación

Característico de Dafny

- Es un lenguaje híbrido: funcional y orientado a objetos
- Se ejecuta como **parte del compilador**
- El programador interactúa con él modificando el programa
  - como cuando corregimos tipos o errores detectados por los análisis que hacen los IDEs
- Se puede usar la versión web, o descargar y usar con Visual Studio

Dafny: análisis en .NET

Notación

Algunas construcciones:

keyword	uso
`requires`	precondición
`ensures`	postcondición
`modifies`	elemento que puede modificarse
`assert`	aserción

expresión	significado
`multiset`	maneja conjunto de variables
`old(exp)`	valor de `exp` al inicio del método
`predicate...`	define un predicado
`forall...`	cuantificación universal
`exists...`	cuantificación existencial
`if [...then ...else]`	condicional

Dafny: análisis en .NET

Ejemplo

```
var x: int;
var y: int;
var tmp: int;

method Swap()
  modifies this;
  ensures x==old(y) && y==old(x);
  { tmp := x;
    x := y;
    y := tmp;
  }
```

Bibliografía (1/2)

Software Testing:

- *Software Reliability Methods*. Doron A. Peled. Springer, 2001. (Capítulo 9).

Semántica:

- Glynn Winskel. *The Formal Semantics of Programming Languages: An Introduction*. MIT Press, 1993 (Capítulo 2).

Generación basada en ejecución simbólica:

- James C. King. *Symbolic execution and program testing*. Comm. of the ACM, 19(7):385-394, 1976.
- L.A. Clarke. *A System to Generate Test Data and Symbolically Execute Programs*. IEEE Transacions on Software Engineering. 2(3):215-222, 1976.

Bibliografía (2/2)

Java Modeling Language (ESC/Java2):

- *Advanced Specification and Verification with JML and ESC/Java2*. P. Chalin, J. R. Kiniry, G. T.Leavens and E. Poll. Formal Methods for Components and Objects, 2006.

Dafny:

- *Using Dafny, an Automatic Program Verifier*. L. Herbert, K. Rustan, M. Leino and J. Quaresma. 2011.

Análisis estático:

- *Principles of Program Analysis*. F. Nielson, H. R. Nielson and C. Hanking. Springer, 2004.