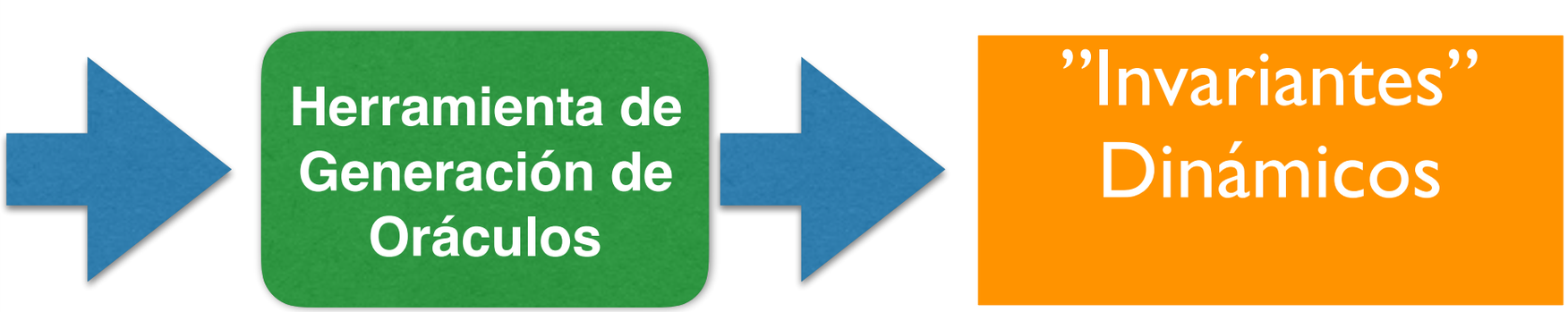
Taller #5: Daikon

Generación Automática de Casos de Test - 2016

Generación Automática de Tests

Programa

```
public static void Main()
{
    byte[] data = new byte[10
    TcpClient server;
    try{
        server = new TcpClien
    }catch (SocketException
        Console.WriteLine("
        return;
    }
        kstream ns = se
```



JUnit

Test Suite

Detección de Invariantes Dinámicos

Ejecutar

Trazas

Filtrar

Invariantes

Resultados

Post-condición

```
b[] = orig(b[])
return == sum(b)
```

plse.cs.washington.edu

The Daikon invariant detector

Daikon is an implementation of dynamic detection of likely invariants; that is, the Daikon invariant detector reports likely program invariants. An invariant is a property that holds at a certain point or points in a program; these are often seen in assert statements, documentation, and formal specifications. Invariants can be useful in program understanding and a host of other applications. Examples include "x.field > abs(y)"; "y = 2*x+3"; "array a is sorted"; "for all list objects lst, lst.next.prev = lst"; "for all treenode objects n, n.left.value < n.right.value"; "p != null => p.content in myArray"; and many more. You can extend Daikon to add new properties (see Enhancing Daikon output, or see New invariants in *Daikon Developer Manual*).

Dynamic invariant detection runs a program, observes the values that the program computes, and then reports properties that were true over the observed executions. Daikon can detect properties in C, C++, C#, Eiffel, F#, Java, Perl, and Visual Basic programs; in spreadsheet files; and in other data sources. (Dynamic invariant detection is a machine learning technique that can be applied to arbitrary data.) It is easy to extend Daikon to other applications.

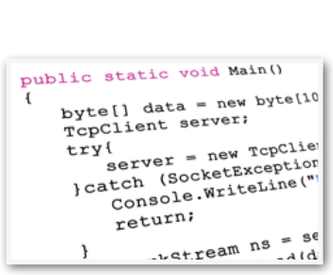
Daikon is freely available for download from http://plse.cs.washington.edu/daikon/download. The distribution includes both source code and documentation, and Daikon's license permits unrestricted use (see License). Many researchers and practitioners have used Daikon; those uses, and Daikon itself, are described in various publications.

[**Home** | FAQ | Download | Documentation | Publications | Mailing lists]

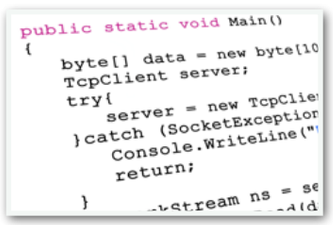
Daikon

- Es un detector de invariantes basado en patrones (como los vistos en la clase pasada)
- Es multi-lenguaje (front-ends para Java, C, C#, etc.)
- El front-end crea un archivo (dtrace) con los valores observados en la entrada y salida de cada método
- Ese archivo dtrace es luego analizado y se reportan los invariantes detectados

Daikon



Java



C#

Front-end Java

Front-end C#

.dtrace

Back-end Daikon

Invariantes

C,C++,Eiffel,Perl,VB

Daikon

- Open-source: https://github.com/codespecs/daikon
- Manual: https://plse.cs.washington.edu/daikon/download/doc/daikon.html
  - Contiene la lista de patterns disponibles
  - Explica como agregar nuevos patrones user-defined
- Instrumenta el bytecode de Java y escribe los valores en un archivo dtrace.gz

Ejemplo

```
class MyMainClass {

        private static int max(int x, int y) {
                if (x<y) {
                        return y;
                } else {
                        return x;
                }
        }

        public static void main(String[] args) {
                max(0,0);
        }

}
```

Ejecutar Daikon

- Invocar la ejecución de un programa y reportar invariantes MyMainClass
  - `$ java -cp [classpath] daikon.Chicory --daikon MyMainClass`
  - donde [classpath] debe incluir tanto el daikon.jar como las clases del sistema bajo test

Invariantes Daikon

```
================================================================
MyMainClass.main(java.lang.String[]):::ENTER
args has only one value
args.getClass().getName() == java.lang.String[].class
args[] == []
args[].toString == []
================================================================
MyMainClass.main(java.lang.String[]):::EXIT
args[] == orig(args[])
args[] == []
args[].toString == []
================================================================
MyMainClass.max(int, int):::ENTER
x == y
x == 0
================================================================
MyMainClass.max(int, int):::EXIT7
================================================================
MyMainClass.max(int, int):::EXIT
return == orig(x)
return == orig(y)
return == 0
Exiting Daikon.
```

Excepciones

- Se omiten en el monitoreo aquellos casos donde se produjo una excepción
- Por ejemplo, si StackAr(-5) produjo una excepción, el -5 no es analizado como un valor posible
- Los invariantes son invariantes de funcionamiento normal (no excepcional)

Inserción de Invariantes

- Daikon permite insertar **automáticamente** (un subconjunto) de los invariantes dinámicos como aserciones en el código

- La clase que permite la instrumentación es la clase Main del package daikon.tools.runtimechecker:

- Ejemplo:

  - `$ java daikon.tools.runtimechecker.Main instrument BoundedStack.inv.gz ubs/BoundedStack.java`

Daikon+JUnit4

- Para invocar un conjunto de test clases de JUnit se debe utilizar la clase org.junit.runner.JUnitCore de JUnit
  - `$ java -cp [classpath] daikon.Chicory --daikon org.junit.runner.JUnitCore TestClass1 TestClass2 …`
- donde [classpath] debe incluir
  - el jar-file de daikon
  - el jar-file de junit (versión >=4)
    - Opcionalmente el hamcrest de acuerdo a la versión de JUNIT4
  - las carpetas donde está la compilación del sistema bajo test y los test classes compilados

--ppt-select-pattern

- La opción ppt-select-pattern nos permite indicar un conjunto de program points de interés

- Por ejemplo, sólo los de la clase org.autotest.MyMainClass

- Ejemplo:

  - `$ java -cp [classpath] daikon.Chicory --daikon --ppt-select-pattern=MyMainClass MyMainClass`

Enunciado

Ejercicio #1

- Ejecutar Daikon sobre el proyecto stackar

    1. Descargar el proyecto stackar_project.zip y las librerías daikon.jar (versión 5.3.10), hamcrest-core-1.3.jar, junit-4.12.jar

    2. Configurar el script run_daikon.sh con la ruta de la carpeta donde se descargaron las librerías del punto anterior.

    3. Ejecutar el script run_daikon.sh y observar los invariantes que produjo Daikon.

Ejercicio #2

- Extender el test suite org.autotest.TestStackAr tal que Daikon reporte los mejores invariantes dinámicos posibles para los siguientes métodos de la clase org.autotest.StackAr:
  - El invariante de representación (.StackAr:::OBJECT)
  - El Constructor StackAr(int)
  - El método top()
  - El método toString()

Ejercicio #3

- Modificar el código de org.autotest.StackAr para que Daikon detecte invariantes de ciclo para el loop en el método toString()