Taller de Cobertura (Python)

Generación Automática de Casos de Tests - 2016

Slides from A.Zeller (Saarland U.)

Cobertura en Java

- En el Taller anterior aprendimos a usar una herramiento de cobertura (EclEmma) para medir la cobertura de un test suite
- Diseñamos nuestro test suite usando esta herramienta

Y ahora…

¡Vamos a construir nuestra propia herramienta de cobertura!

wat

Cobertura en Python

- **Objetivo:** Escribir una tool que mida la cobertura de branches y statements en Python

cgi_decode.py

```
def cgi_decode(s):
    t = ""
    i = 0
    while i < len(s):
        c = s[i]
        if c == '+':
            t = t + ' '
        elif c == '%':
            digit_high = s[i + 1]
            digit_low  = s[i + 2]
            i = i + 2
            if (hex_values.has_key(digit_high) and
                hex_values.has_key(digit_low)):
                v = (hex_values[digit_high] * 16 +
                     hex_values[digit_low])
                t = t + chr(v)
            else:
                raise Exception
        else:
            t = t + c

        i = i + 1

    return t
```

A B C D E F G H I L M

Tracing en Python

- En Python, tracear ejecuciones es mucho más simple que en los languajes compilados.
- La función sys.settrace(f) define f()
  como una función de tracing que es invocada
  (llamada) por cada línea ejecutada
- f() tiene acceso al estado completo del intérprete

Tracing en Python

frame actual (PC + variables)

“line”, “call”, “return”, …

```
import sys

def traceit(frame, event, arg):
    if event == "line":
        lineno = frame.f_lineno
        print "Line", lineno, frame.f_locals
    return traceit

sys.settrace(traceit)
```

tracer a ser usado
en este scope (este mismo)

https://docs.python.org/3/library/sys.html#sys.settrace

Tracing de Branches

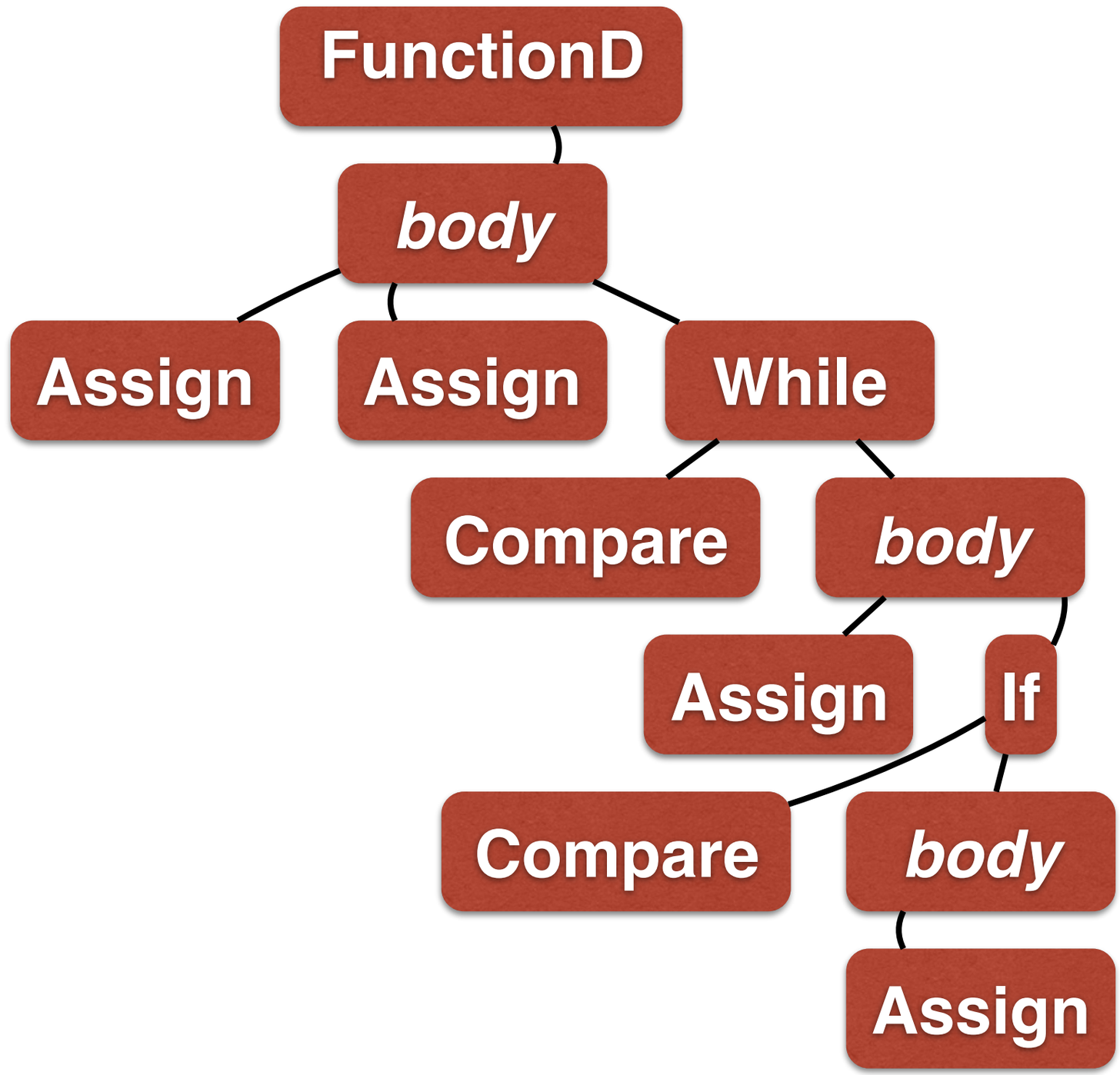
- Tracer los branches tomados es fácil – sólo tracea todos los pares de líneas ejecutados secuencialmente
- Pero cómo obtenemos todos los branches posibles?
- Se necesita analizar el programa estáticamente (sin ejecutarlo)

Abstract Syntax Trees

```
def cgi_decode(s):
    t = ""
    i = 0
    while i < len(s):
        c = s[i]
        if c == '+':
            t = t + ' '
        elif c == '%':
            ...
        else:
            t = t + c

        i = i + 1

    return t
```



Python AST

- El módulo de AST de Python convierte un archivo de programas Python en su abstract syntax tree (AST)
- El árbol puede ser visitado usando el design-pattern "visitor"

Python AST

root del AST

input de Python

```
import ast

root = ast.parse('x = 1')
print ast.dump(root)
```

AST como string
(para debugging)

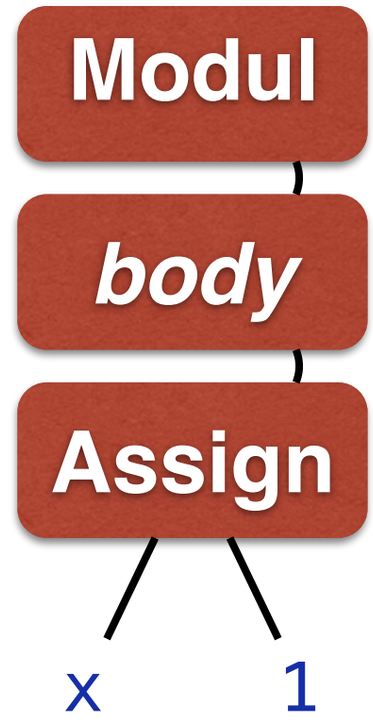
https://docs.python.org/2/library/ast.html#ast.AST

Python AST

```
import ast

root = ast.parse('x = 1')
print ast.dump(root)
```

```
→
Module(
   body = [
      Assign(
         targets = [
            Name(id = 'x', ctx = Store())
         ],
         value = Num(n=1)
      )
   ]
)
```



Design Pattern Visitor

- Desacopla la estructura de datos del código que efectua cambios/inspecciones sobre la estructura
- Ejemplo: árboles binarios
- Class BinaryTree: sólo construye el árbol
- BinaryTreeVisitor: recorre el árbol (y hace lo que tiene que hacer)

Ejemplo: Design Pattern

```
/* Arbol Binario */
class BinaryTree {
  private BinaryTree left;
  private BinaryTree right;
  private int value;
  public void accept(BinaryTreeVisitor v) {
    v.visit(this);
  }
}
```

Ejemplo: Design Pattern

```
/* Visitor abstracto para Binary Tree */
abstract class BinaryTreeVisitor {
 public void visit(BinaryTree b) {
     // override
  }
}
```

Ejemplo: Design Pattern

```
/* Visitor para contar nodos de un Binary Tree */

class NodeCounterVisitor extends BinaryTreeVisitor {

 private int nodes = 0;

  public void visit(BinaryTree b) {

      if (b.left!=null) { b.left.accept(this); }

      if (b.right!=null) {b.right.accept(this); }

      nodes+=1;

   }

   public int getNodeCount() { return nodes; }

}
```

AST Visitor

- La clase ast.NodeVisitor provee un método visit(n) que recorre todos los subnodos de n
- Debe ser subclaseada para ser extendida
- En cada nodo n de tipo TYPE, el método visit_TYPE(n) es llamado si este exists
- Si no existe visit_TYPE(n), el método generic_visit() recorre todos los hijos

AST Visitor

```
class IfVisitor(ast.NodeVisitor):

    def visit_If(self, node):
        print "if", node.lineno, ":"
        for n in node.body:
            print "    ", n.lineno
        print "else:"
        for n in node.orelse:
            print "    ", n.lineno

        self.generic_visit(node)
```

número de línea

muestra cuerpo y parte “else”

recorre los hijos

AST Visitor

Lee source Python

```
root = ast.parse(open('cgi_decode.py').read())

v = IfVisitor()
v.visit(root)
```

Visita todos los nodos IF

```
→
if 34 :
      35
else:
      36
if 36 :
      37
      38
      39
      40
else:
      47
if 40 :
      42
      43
else:
      45
if 81 :
      82
      83
else:
```

AST Visitor

```
def cgi_decode(s):
    t = ""
    i = 0
    while i < len(s):
        c = s[i]
34      if c == '+':
35          t = t + ' '
36      elif c == '%':
37          digit_high = s[i + 1]
38          digit_low  = s[i + 2]
39          i = i + 2
40          if (hex_values.has_key(digit_high) and
41              hex_values.has_key(digit_low)):
42              v = (hex_values[digit_high] * 16 +
43                   hex_values[digit_low])
44              t = t + chr(v)
45          else:
46              raise Exception
47      else:
            t = t + c

        i = i + 1

    return t
```

```
→
if 34 :
    35
else:
    36
if 36 :
    37
    38
    39
    40
else:
    47
if 40 :
    42
    43
else:
    45
if 81 :
    82
    83
else:
```

Enunciado (Preparación)

1. Ejecutar los tests de unidad del módulo test_cgi_decode.py para el módulo cgi_decode.py
   ¿Cuántos tests de unidad pasan y cuántos no pasan?
2. Crear un nuevo módulo nuevo que construya el AST para el módulo cgi_decode.py y lo imprima por consola usando el modulo "ast" de python.

Enunciado (Counter)

3. Crear una clase **LineCounter** que cuente la cantidad de líneas de código (statements) hay en el módulo cgi_decode.py.

¿Cuántas líneas encontró?

4. Crear una clase **BranchCounter** que cuente la cantidad de branches hay en el módulo cgi_decode.py.

¿Cuántas branches encontró?

Enunciado (Coverage)

5. Completar la clase **LineCoverage** para que reporte el recubrimiento de statements sobre cgi_decode

¿Cuál es el coverage de statements que reporta?

6. Completar la clase **BrachCoverage** para que reporte el recubrimiento de statements sobre el módulo cgi_decode.py.

¿Cuál es el coverage de statements que reporta?

Cheat Sheet

- hasattr(node, "lineno"): retorna true iff el objeto **node** tiene el atributo "**lineno**" (número de línea)?
- frame.f_lineno: retorna el número de línea del objeto frame
- frame.f_code.co_filename: nombre del archivo dónde se está ejecutando el código que pertenece a este frame
- https://docs.python.org/3/library/inspect.html