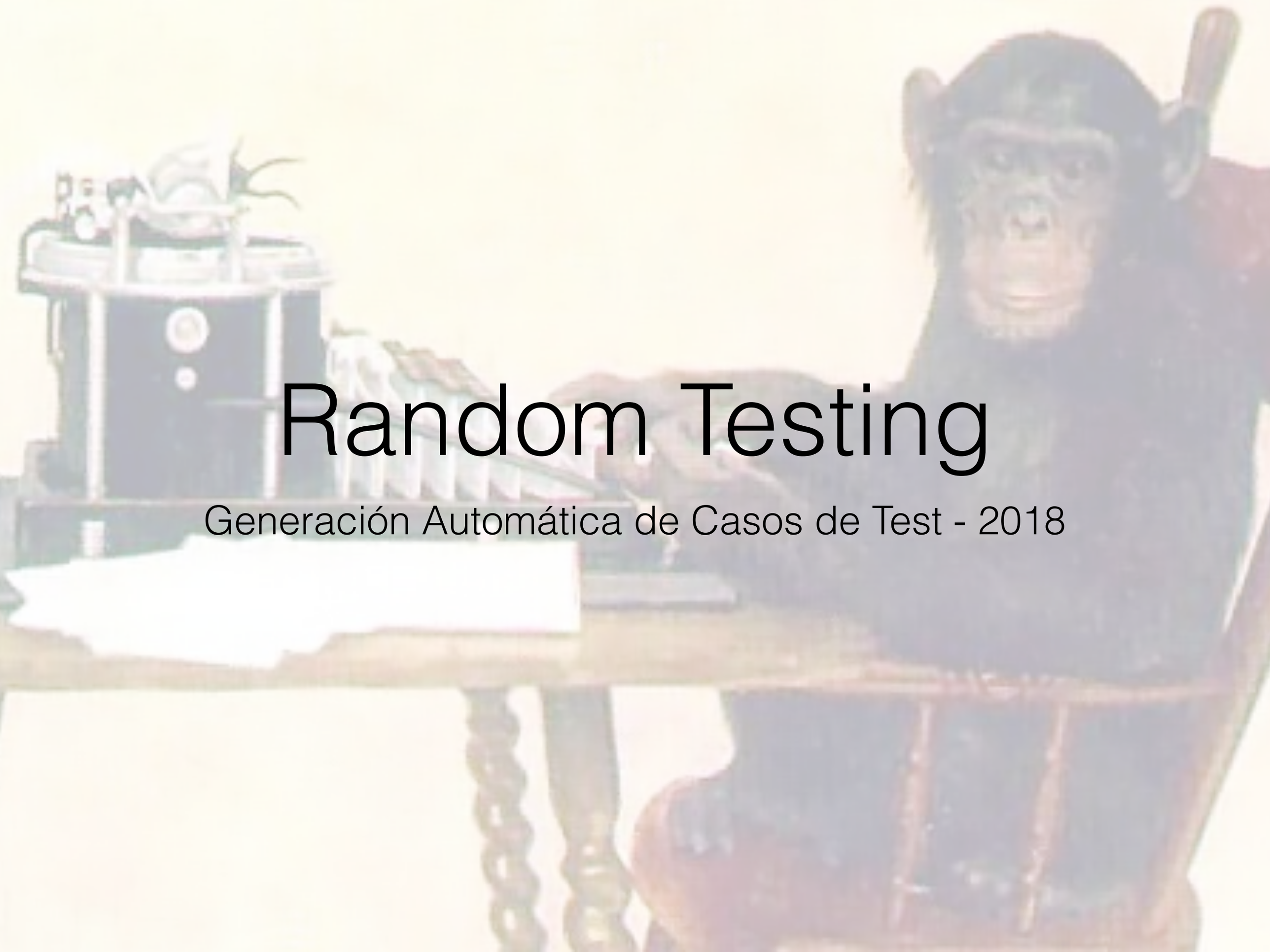


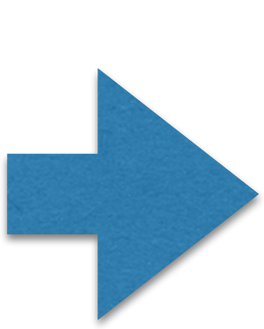
Random Testing

Generación Automática de Casos de Test - 2018

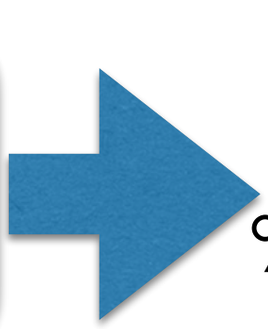
Repaso: Cobertura

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

JUnit



Herramienta de Cobertura



% Líneas Cubiertas
% Branches Cubiertos

ej: JaCoCo

Automáticamente podemos obtener información de cobertura estructural de un test suite

Repaso: Mutation Testing

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

JUnit

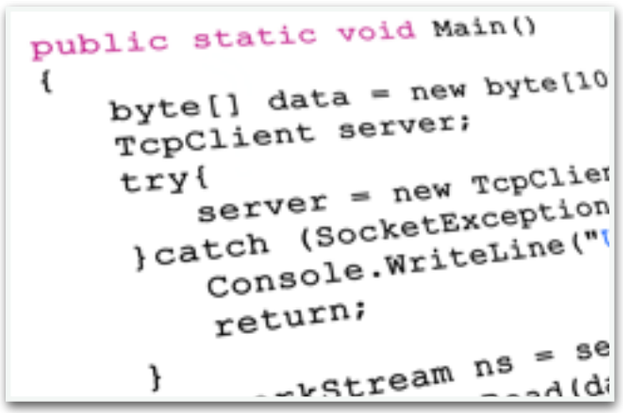
Herramienta de Mutation Testing Analysis

%Mutation Score (Mutantes muertos/ Mutantes totales)

ej: PiTest

Automáticamente podemos obtener el mutation score de un test suite

Generación Automática de Tests

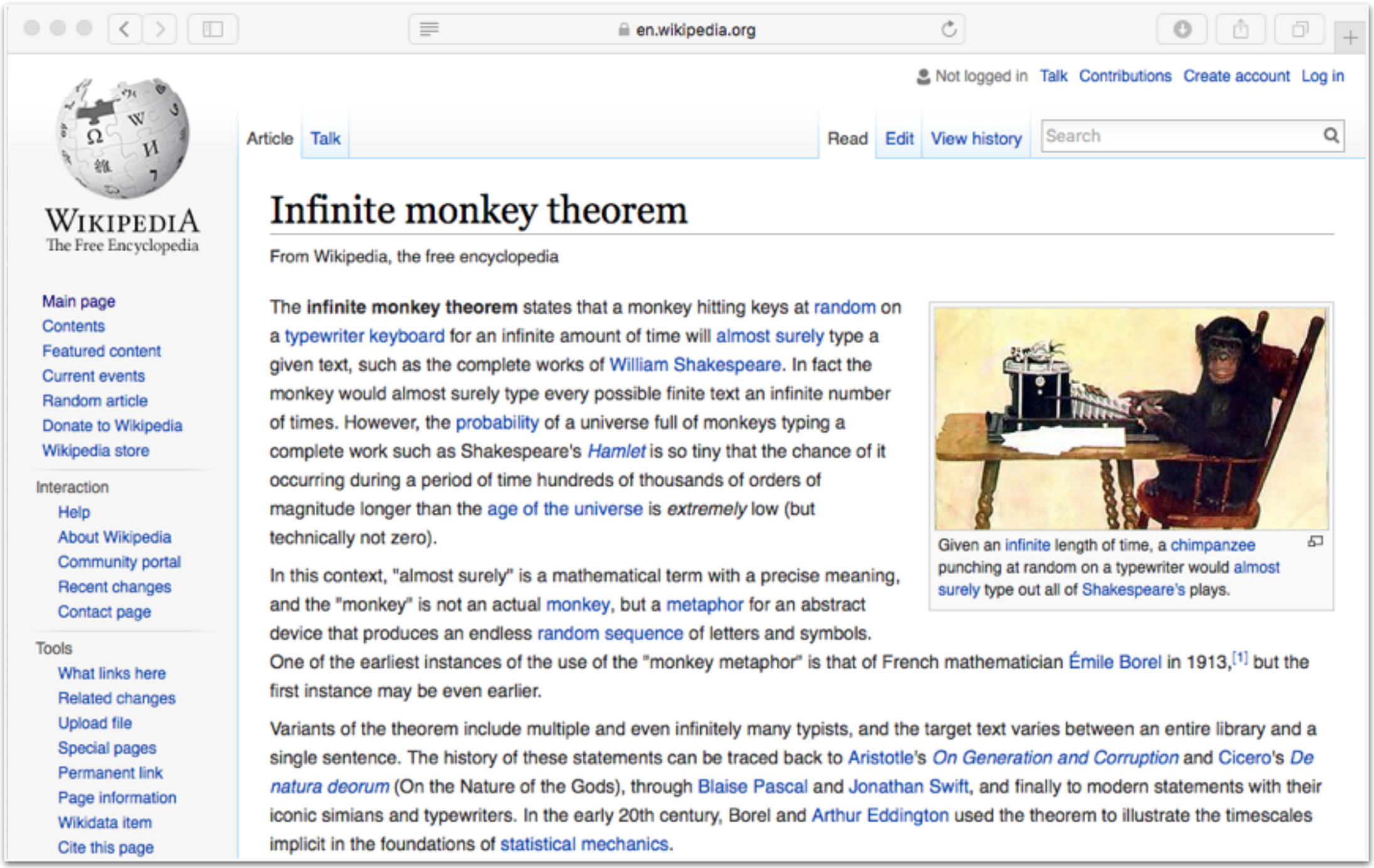


Programa

Test Suite

Teorema del Mono Infinito

Infinite monkey theorem



Infinite monkey theorem

- *"The **infinite monkey theorem** states that a monkey hitting keys at random on a typewriter keyboard for **an infinite amount of time** will almost surely type a given text, such as the complete works of William Shakespeare.*

- *In fact the monkey **would almost surely** type every possible finite text an infinite number of times."*

Random Testing

Programa bajo Test

Oráculo

- **Idea**: definamos nuestro “mono tipeador” y probemos nuestro programa usando sus inputs

- Necesitamos para ello de un Oráculo

Testing de Robustez

- A veces no disponemos de un oráculo para chequear automáticamente el programa

- Robustness Testing: ¿ocurrió algún crash?
  - ¿Qué es un crash?

Testing de Regresión

Programa Version 1

JUnit

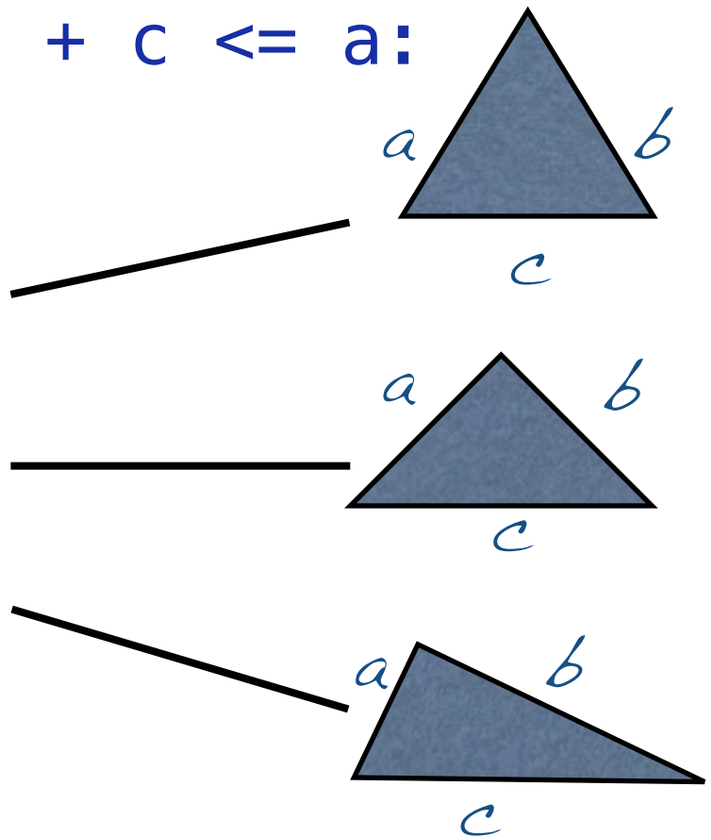
JUnit

Programa Version 2

- Regression Testing: ¿hay una diferencia?
  - ¿Es una diferencia esperable?

Ejemplo: Clasificador de Triángulos

```
def triang(a, b, c):
    if a + b <= c or a + c <= b or b + c <= a:
        return NOT_A_TRIANGLE
    elif a == b == c:
        return EQUILATERAL_TRIANGLE
    elif a == b or b == c or a == c:
        return ISOSCELES_TRIANGLE
    else:
        return SCALENE_TRIANGLE
```



Random Testing

- `triang()` recibe 3 números enteros
- ¿Cómo podemos probar nuestro programa 1000 veces usando valores entre 0 y 100?
- **Random Driver**: ejercita el programa usando valores aleatorios

Random Driver

```
from Triangle import triangle
import random

def random_tester():
    i = 0
    while i < 1000:
        a = random.randint(0, 100)
        b = random.randint(0, 100)
        c = random.randint(0, 100)
        test_passed = execute([a,b,c])
        i = i + 1
```

Ejecución de Tests

```
def execute(a,b,c):
        try:
            print a, b, c,
             v = triangle(a, b, c)
             print "PASS"
             return True

        except Exception as e:
            print "FAIL –", e
            return False
```

- Robustness Testing: ¿ocurrió algún crash?
  - crash = una excepción

Demo

$python randomtester.py

Random Testing

Casino Royale

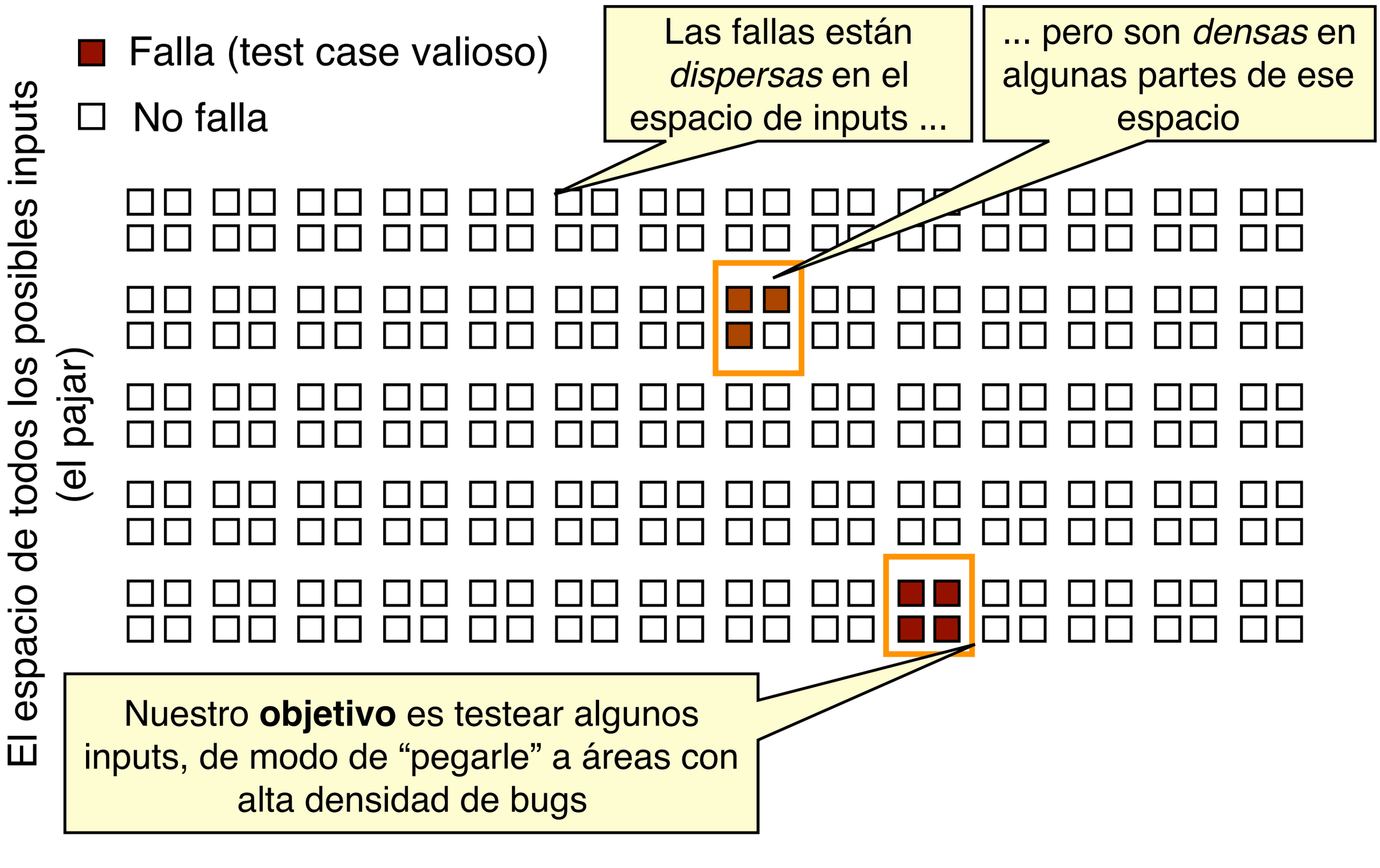
```
def triang(a, b, c):
    if a + b <= c or a + c <= b or b + c <= a:
        return NOT_A_TRIANGLE
    elif a == b == c:
        return EQUILATERAL_TRIANGLE
    elif a == b or b == c or a == c:
        return ISOSCELES_TRIANGLE
    else:
        return SCALENE_TRIANGLE
```

- ¿Cuáles serán nuestras chances de…
  - …obtener un triángulo isósceles?
  - …obtener un triángulo equilátero?

Agujas en el Pajar

- Para encontrar agujas, necesitamos buscar sistemáticamente

Patrones de Fallas



Random Testing **Adaptativo**

- *Adaptive random testing* (Chen et al., 2004)
  - Usa *feedback* de la ejecución de los Casos de test para *guiar* la generación de nuevos tests

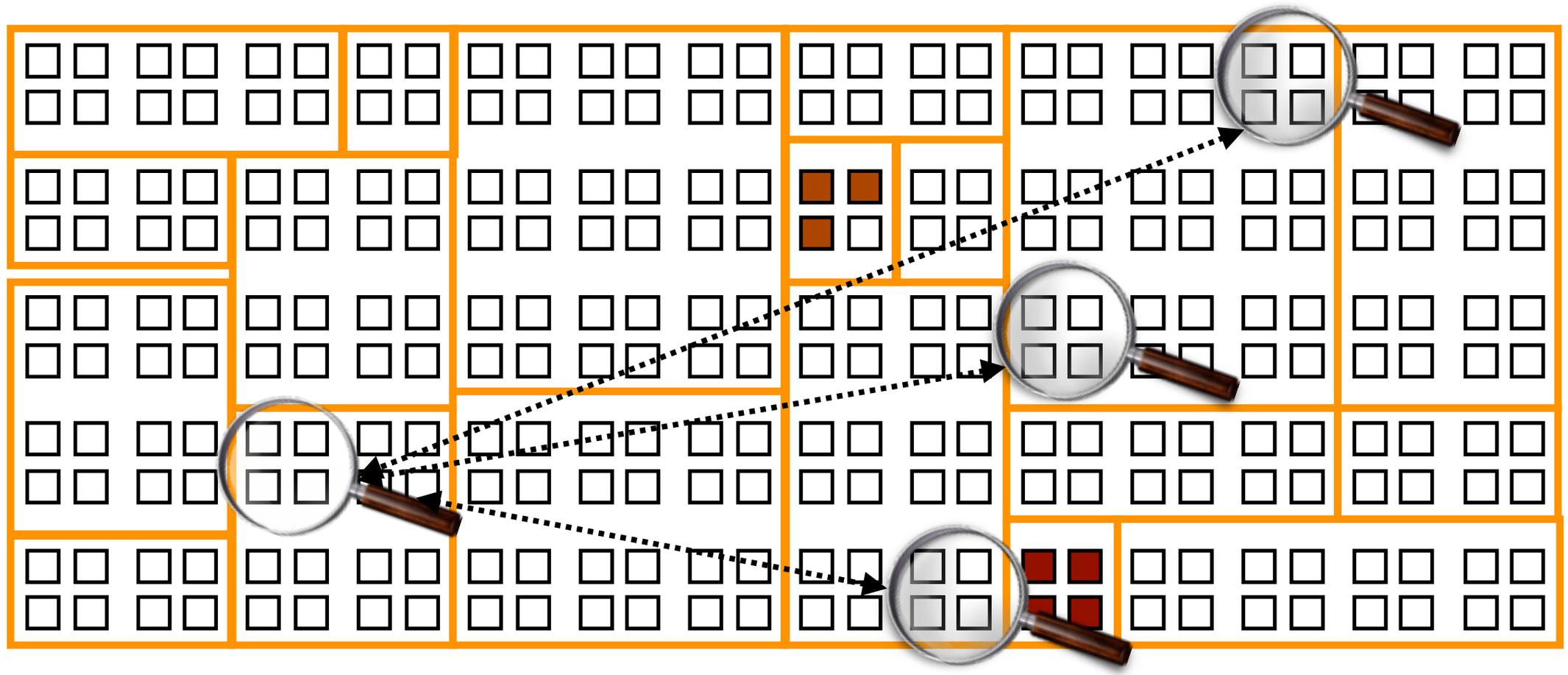
  - **Idea**: Generar casos de test que tenga una alta distancia a tests “no exitosos” (i.e. que no han detectado una falla)

Generación de Candidatos

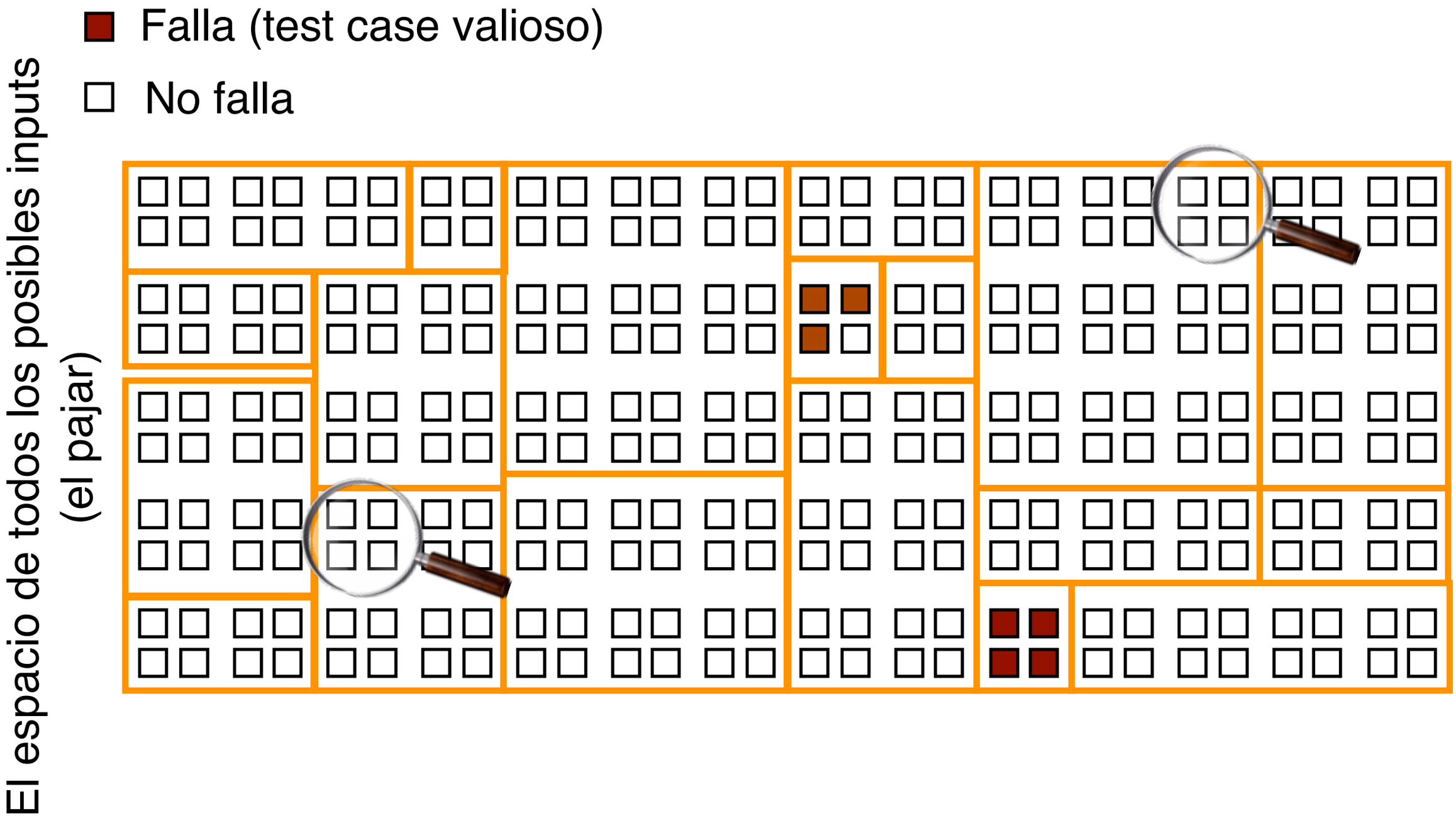
Falla (test case valioso)

No falla

El espacio de todos los posibles inputs
(el pajar)



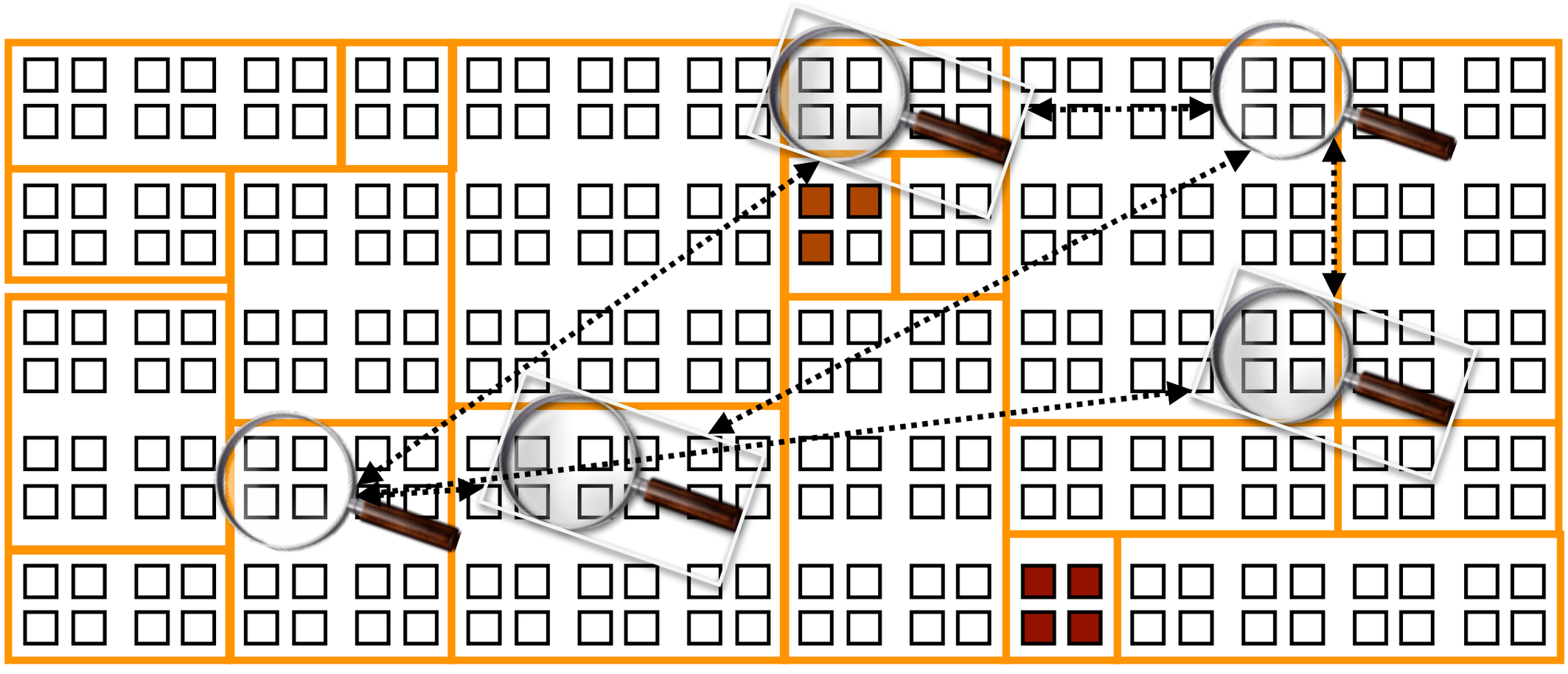
Selección del Candidato



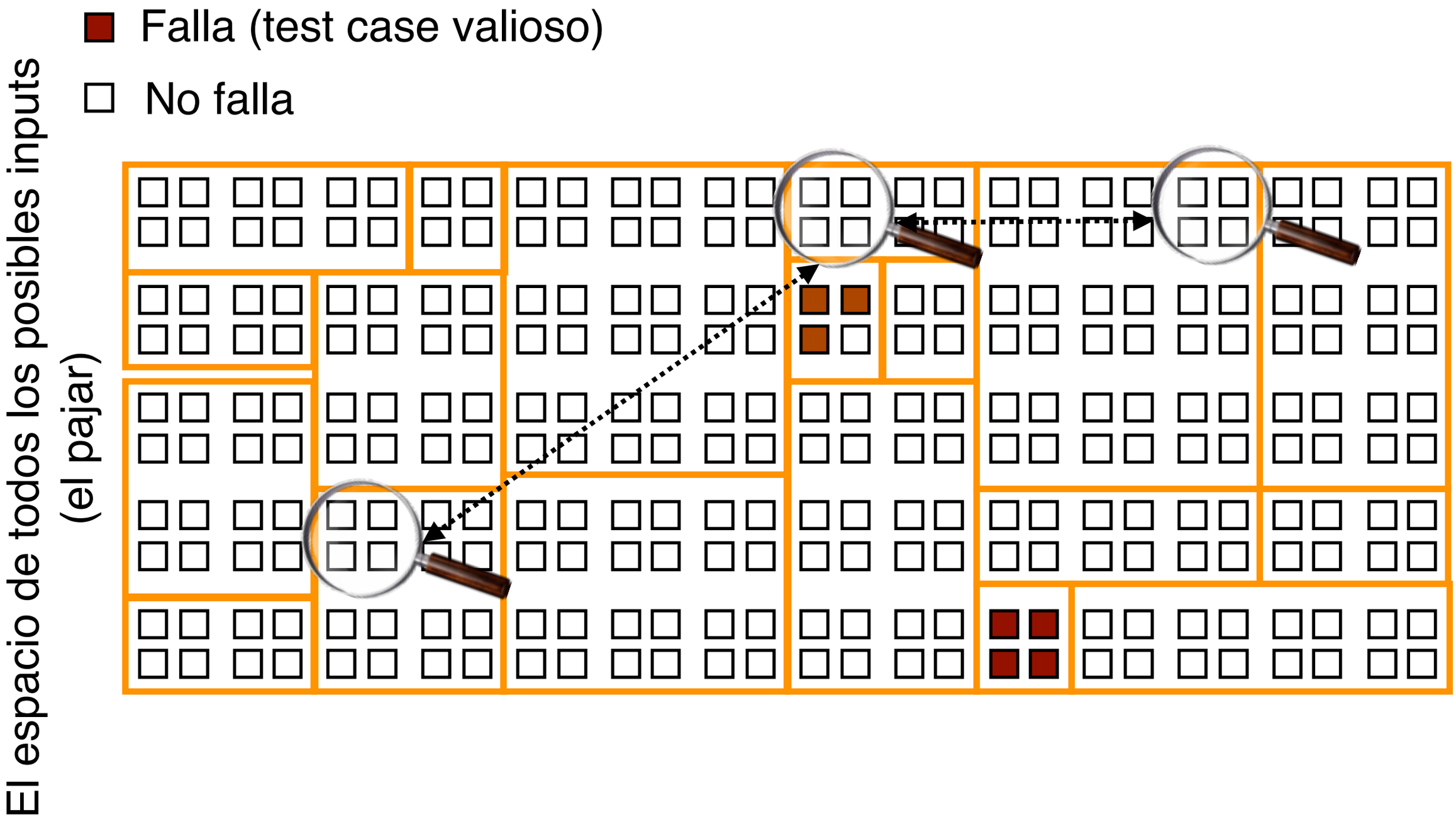
Generación de Candidatos

- Falla (test case valioso)
- No falla

El espacio de todos los posibles inputs
(el pajar)



Selección del Candidato



Métrica de Distancia

- Para poder hacer adaptive random testing, necesitamos una noción de distancia

- Cada dominio necesita su métrica de distancia

- Para puntos (p,q), podemos usar la distancia euclideana que mide la distancia espacial

$$d(p,q) = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2}$$

Métrica de Distancia

$$d(p,q) = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2}$$

```
from math import sqrt

# Compute euclidean distance
# between p = [x, y, z, ...] and q = [x, y, z, ...]
def euclidean_dist(p, q):
    assert len(p) == len(q)
    sum = 0
    for i in range(0, len(p)):
        dist = q[i] - p[i]
        sum = sum + dist * dist
    return sqrt(sum)
```

Generación de Candidatos

```
candidates = []
for i in range(0, CANDIDATES):
    a = random.randint(0, 100)
    b = random.randint(0, 100)
    c = random.randint(0, 100)
    candidates.append([a, b, c])
```

Selección de Candidatos

```
max_min_distance = 0
new_test = None
for candidate in candidates:
    min_distance = None
    for test in passing_tests:
        dist = euclidean_dist(candidate, test)
        if min_distance is None:
            min_distance = dist
        else:
            min_distance = min(min_distance, dist)

    if min_distance > max_min_distance:
        max_min_distance = min_distance
        new_test = candidate
```

Ejecución de Tests

```
try:
        [a, b, c] = new_test
        print a, b, c,
        v = triangle(a, b, c)
        print "PASS"

        # Add candidate to the set
        passing_tests.append(selection)

    except Exception as e:
        print "FAIL –", e
```

Demo

$ python adaptivetester.py

Adaptive Random Testing

¿Funciona?

- Random-Testing Adaptativo puede alcanzar **mejor** cobertura que Random-Testing Clásico

- Pero esta mejoría tiene un **costo**

¿Funciona?

- Generemos *n* tests
- Por cada test *i*, uno tiene que computar *m* candidatos…
- …los cuales deber ser comparados contra 1…(*i-1)* tests existentes (en peor caso)
- necesitamos $\frac{n^2}{2} \times m$ comparaciones!

Discusión

Adaptive Random Testing: An Illusion of Effectiveness?

Andrea Arcuri
Simula Research Laboratory
P.O. Box 134, 1325 Lysaker, Norway
arcuri@simula.no

Lionel Briand
Simula Research Laboratory and
University of Oslo
P.O. Box 134, 1325 Lysaker, Norway
briand@simula.no

ABSTRACT

Adaptive Random Testing (ART) has been proposed as an enhancement to random testing, based on assumptions on how failing test cases are distributed in the input domain. The main assumption is that failing test cases are usually grouped into contiguous regions. Many papers have been published in which ART has been described as an effective alternative to random testing when using the average number of test case executions needed to find a failure (F-measure). But all the work in the literature is based either on simulations or case studies with unreasonably high failure rates. In this paper, we report on the largest empirical analysis of ART in the literature, in which 3727 mutated programs and nearly ten **tr**illion test cases were used. Results show that ART is highly inefficient even on trivial problems when accounting for distance calculations among test cases, to an extent that probably prevents its practical use in most situations. For example, on the infamous Triangle Classification program, random testing finds failures in few milliseconds whereas ART execution time is prohibitive. Even when assuming a small, fixed size test set and looking at the probability of failure (P-measure), ART only fares slightly better than random testing, which is not sufficient to make it applicable in realistic conditions. We provide precise explanations of this phenomenon based on rigorous empirical analyses. For the simpler case of single-dimension input domains, we also perform formal analyses to support our claim that ART is of little use in most sit-

Keywords

Faulty region, random testing, F-measure, P-measure, shape, diversity, similarity, distance.

1. INTRODUCTION

Random testing does not exploit any information about the software under test (SUT). Therefore, historically it was considered a naive testing strategy that is less effective than partition testing. Myers in his 1979 book [34] states "probably the poorest [test case design] methodology is random input testing ...". But this statement was not agreed upon by everyone. Thayer *et al.* [41] for example were favorable to random testing, recommending it as a necessary final step of the testing activity.

In a seminal paper, Duran and Ntafos [17] carried out a series of experiments in which they showed that random testing could be more effective than the commonly used partition testing. That was a counterintuitive result that opened the doors to a large body of literature on the properties and benefits of random testing. Random testing has been shown to be a very useful tool in the hands of software testers and it is not a minor technique as it was originally thought [17, 5].

Usually, in random testing the test cases are sampled at random from the input domain D of the SUT with uniform probability. In other words, each test case is sampled with probability $p = 1/|D|$. The motivation is that, if we do not know anything about the SUT,

¿Funciona?

- El número de comparaciones se torna prohibitivo para **números grandes de tests** *(suele ser el caso en random testing)*

- ¡El tiempo utilizado en computar distancias y comparándolas puede ser usando en **ejecutar mas casos de tests**!

Demo

Random Testing vs. Adaptive Random Testing

Running "randomtester.py"...

Running "randomtester.py"...

Theme:

Python 2.7.5

```
7 3 8 PASS
0 3 9 PASS
7 1 9 PASS
6 2 0 PASS
4 3 3 PASS
1 3 8 PASS
7 3 10 PASS
7 10 8 PASS
5 4 9 PASS
9 8 6 PASS
3 2 9 PASS
4 4 9 PASS
0 6 10 PASS
0 10 10 PASS
6 6 0 PASS
0 10 6 PASS
3 5 3 PASS
2 5 10 PASS
0 5 4 PASS
1 8 7 PASS
3 8 9 PASS
5 3 9 PASS
9 8 5 PASS
0 6 6 PASS
4 5 9 PASS
4 4 0 PASS
5 3 10 PASS
10 1 9 PASS

8000 Tests
55 Failures
```

Program exited with code #0 after 9.34 seconds.

copy output

Running "a

```
8 2 0 PASS
0 0 3 PASS
10 1 8 PASS
9 8 6 PASS
2 2 3 PASS
0 8 3 PASS
4 7 6 PASS
5 0 7 PASS
2 7 1 PASS
4 8 9 PASS
2 6 0 PASS
3 0 8 PASS
6 2 8 PASS
5 3 5 PASS
7 6 3 PASS
8 2 2 PASS
2 0 4 PASS
9 5 8 PASS
9 9 5 PASS
3 6 7 PASS
9 6 4 PASS
2 7 5 PASS
9 8 7 PASS
8 0 8 PASS
3 2 5 PASS
9 8 8 PASS
4 2 2 PASS
3 7 7 PASS

1000 Tests
9 Failures
```

Program exited with code #0 after 8.93 seconds.

copy output

Random Testing para programas Orientados a Objetos

- **Hasta ahora**: los programas bajo tests sólo tienen entradas primitivas (integers)

- ¿Qué hacemos si el parámetro es de tipo *T* (no es un tipo primitivo)?

- Trivialmente: podemos elegir aleatoriamente entre:

  - El valor **null**

  - Invocar al constructor del tipo sin parámetros (si existe)

Random Testing para programas Orientados a Objetos

- Dado este método (Java)

```
public static Integer largest(LinkedList<Integer> list) {
  int index = 0;
  int max = Integer.MIN_VALUE;
  while (index <= list.size()-1) {
      if (list.get(index) > max) {
          max = list.get(index);
      }
      index++;
  }
  return max;
}
```

- La solución anterior puede usar `LinkedList()` o `null`
  - list=`null`
  - list=`[]` (la lista vacía)

Random Testing para programas Orientados a Objetos

- En los lenguajes orientados a objetos, necesitamos crear y modificar instancias de objetos

- Ejemplo:

```
// setup
LinkedList list0 = new LinkedList()
list0.add(null);
list0.add(null);
// exercise
list0.remove(0);
// check
assertEquals(1, list0.size());
```

Random Testing para programas Orientados a Objetos

- En los lenguajes orientados a objetos, una excepción no siempre es síntoma de falla

- Ejemplo:

```
LinkedList list0 = new LinkedList()
list0.get(-1); //throws IndexOutOfBoundsException
```

Que tira una excepción no significa un error, ya que se supone que get() de un índice negativo debe emitir un IndexOutOfBoundsException

Random Testing para programas Orientados a Objetos

- Elegir valores aleatorios de tipos primitivos (floats, strings, integers, booleans es fácil)
    - Existe una representación finita (bits)
- ¿Cómo elegimos aleatoriamente una instancia de una clase (ej:HashSet, LinkedList,File)?
- Necesitamos poder crear **secuencias de llamados a métodos** para construir instancias complejas (i.e. interesantes)

Catálogo de Métodos

- Necesitamos definir un **catálogo (menú)** de los métodos que podemos usar para poder crear instancias de objetos:
  - Por ejemplo:
    - java.util.*, java.lang.*
    - Todas las clases del proyecto
- Este **catálogo** puede ser calculado automáticamente

Generación de Secuencias de Invocaciones (Tests)

```
// generate a new sequence (that was not already in seqs)
// seqs is a set of already generated sequences
def generate_new_test_sequence(C,seqs):
  while (True):
    choose m(T1,…,Tk):Tr from catalog C
    for each input parameter of type Ti:
      choose Si from seqs s.t. builds an object of type Ti

    build new sequence Snew=S1;…;Sk;Tr vnew=m(v1,…,vk)
    if not Snew in seqs:
      return Snew
```

Generación de Secuencias de Invocaciones (Tests)

```
// Catalog of methods of interest
C = {…}
// seed basic sequences (e.g. “boolean b=false”, “int i=0”)
tests = { … }
```

Generación de Secuencias de Invocaciones (Tests)

```
// Catalog of methods of interest
C = {…}
// seed basic sequences (e.g. “boolean b=false”, “int i=0”)
tests = { … }

Do until time limit expires:

  new_test = generate_new_test_sequence(C,tests):
   // add new test to the output
  tests.add(new_test)

return tests
```

catálogo

```
class Foo
  static int max(int a, int b) {…}
class java.util.LinkedList
  LinkedList()
  boolean add(Object)
  boolean remove(int)
  Object get(int)
class Integer
  Integer(int)
  int intValue()
```

secuencias

```
int int0 =0;
boolean boolean0 = false;
Object object0 = new Object();
Integer integer0 = null;
float float0 = 0.0f;
LinkedList linkedList0 = null;
```

catálogo

```
class Foo
  static int max(int a, int b) {…}
class java.util.LinkedList
  LinkedList()
  boolean add(Object)
  boolean remove(int)
  Object get(int)
class Integer
  Integer(int)
  int intValue()
```

secuencias

```
int int0 =0;
boolean boolean0 = false;
Object object0 = new Object();
Integer integer0 = null;
float float0 = 0.0f;
LinkedList linkedList0 = null;
```

```
LinkedList linkedList0 = new LinkedList();
```

catálogo

```
class Foo
  static int max(int a, int b) {…}
class java.util.LinkedList
  LinkedList()
  boolean add(Object)
  boolean remove(int)
  Object get(int)
class Integer
  Integer(int)
  int intValue()
```

secuencias

```
nt int0 =0;
boolean boolean0 = false;
Object object0 = new Object();
Integer integer0 = null;
float float0 = 0.0f;
LinkedList linkedList0 = null;
inkedList linkedList0 = new LinkedList();
```

```
int int0 = 0;
LinkedList linkedList0 = new LinkedList();
boolean boolean0= linkedList0.remove(int0);
```

catálogo

```
class Foo
  static int max(int a, int b) {…}
class java.util.LinkedList
  LinkedList()
  boolean add(Object)
  boolean remove(int)
  Object get(int)
class Integer
  Integer(int)
  int intValue()
```

secuencias

```
int int0 =0;
boolean boolean0 = false;
bject object0 = new Object();
Integer integer0 = null;
float float0 = 0.0f;
LinkedList linkedList0 = null;
nkedList linkedList0 = new LinkedList();
```

```
int int0 = 0;
LinkedList linkedList0 = new LinkedList();
boolean b = linkedList0.remove(int0);
```

```
Object object0 = new Object();
LinkedList linkedList0 = new LinkedList();
boolean boolean0 = linkedList0.add(object0)
```

catálogo

```
class Foo
  static int max(int a, int b) {…}
class java.util.LinkedList
  LinkedList()
  boolean add(Object)
  boolean remove(int)
  Object get(int)
class Integer
  Integer(int)
  int intValue()
```

secuencias

```
int int0 =0;
```

```
boolean boolean0 = false;
```

```
Object object0 = new Object();
```

```
Integer integer0 = null;
```

```
float float0 = 0.0f;
```

```
LinkedList linkedList0 = null;
```

```
LinkedList linkedList0 = new LinkedList();
```

```
int int0 = 0;
LinkedList linkedList0 = new LinkedList();
boolean b = linkedList0.remove(int0);
```

```
Object object0 = new Object();
int int0 = 0;
LinkedList linkedList0 = new LinkedList();
linkedList0.remove(int0);
boolean boolean0 = linkedList0.add(object0)
```

. . .

Limitaciones (1)

- Algunas secuencias no consiguen construir la instancia esperada.
  - Ejemplo:

```
int int0 = -10;
ArrayList list0 = new ArrayList(int0)
```

- Esta provoca una excepción
- Es usada para generar otras secuencias!

Limitaciones (2)

- Hay secuencias distintas que producen el mismo objeto

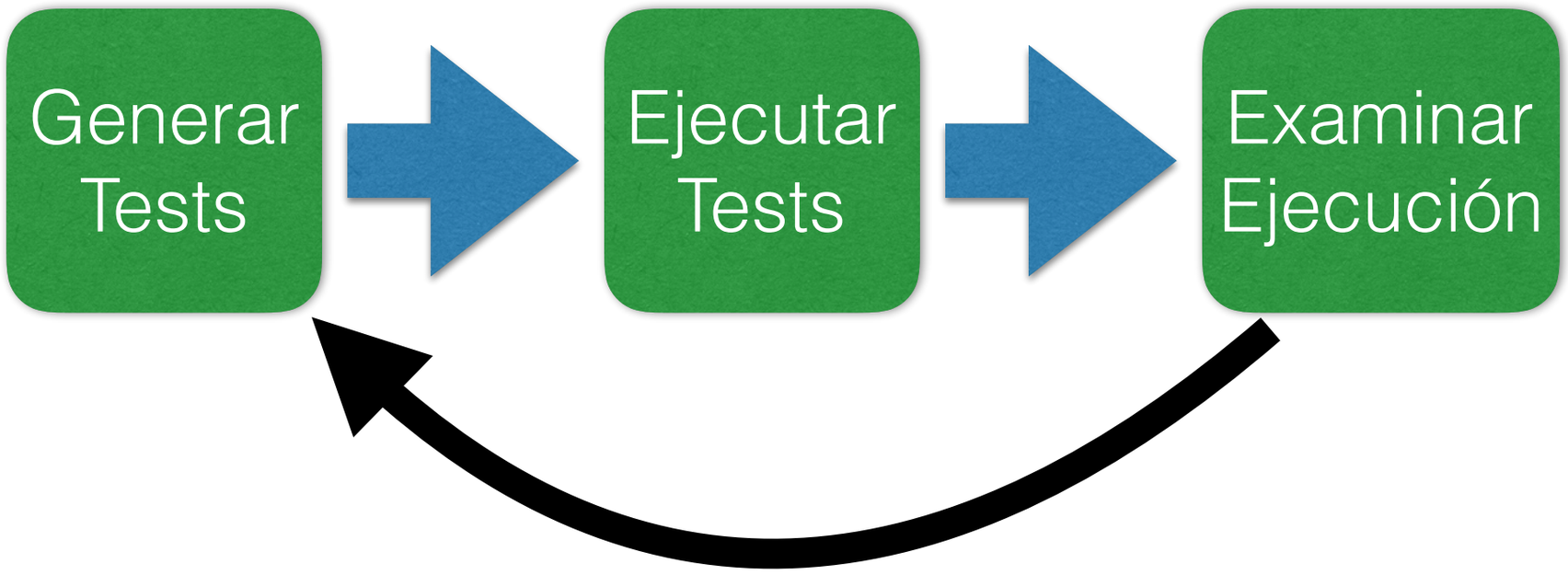
  - Ejemplo:

```
LinkedList list0 = new LinkedList()
list0.isEmpty()
```

y

```
LinkedList list0 = new LinkedList()
list0.add(null)
list0.remove(0)
list0.isEmpty()
```

Random Testing guiado por Feedback



```
// Catalog of methods of interest
C = {…}
// seed basic sequences (e.g. “boolean b=false”, “int i=0”)
tests = { … }
// valid instances
instances = {}
// exceptional tests
exception_tests = {}
```

```
// Catalog of methods of interest
C = {…}
// seed basic sequences (e.g. “boolean b=false”, “int i=0”)
tests = { … }
// valid instances
instances = {}
// exceptional tests
exception_tests = {}

Do until time limit expires:
  new_test = generate_new_test_sequence(C,tests):
  try:
    r = execute_test(new_test)
    if (r !in instances)
      // sequence creates a previously unseen instance
      tests.add(new_test)
      instances.add(r)
    else:
      // discard sequence
  catch Exception:
    // sequence signals an exception
    exception_tests.add(new_test)

return tests, exception_tests
```

Usamos el feedback de la ejecución

Oráculos

- Si tenemos un oráculo automático, podemos clasificar los tests entre ***passing*** y ***failing***

- Podemos usar Oráculos Implícitos (e.g. propiedades generalmente válidas)

- `o.equals(o)` returna verdadero

- `o.hashCode()` no tira una excepción

- `o.toString()` no tira una excepción

```
// Catalog of methods of interest
C = {…}
// seed basic sequences (e.g. “boolean b=false”, “int i=0”)
passing_tests = { … }
// valid instances
instances = {}
// exceptional tests
failing_tests = {}
```

```
// Catalog of methods of interest
C = {…}
// seed basic sequences (e.g. “boolean b=false”, “int i=0”)
passing_tests = { … }
// valid instances
instances = {}
// exceptional tests
failing_tests = {}

Do until time limit expires:
  new_test = generate_new_test_sequence(C,passing_tests):
  try:
    r = execute_test(new_test,oracles)
```

```
// Catalog of methods of interest
C = {…}
// seed basic sequences (e.g. “boolean b=false”, “int i=0”)
passing_tests = { … }
// valid instances
instances = {}
// exceptional tests
failing_tests = {}

Do until time limit expires:
  new_test = generate_new_test_sequence(C,passing_tests):
  try:
    r = execute_test(new_test,oracles)
    if (r !in instances)
      // sequence creates a previously unseen instance
      passing_tests.add(new_test)
      instances.add(r)
```

```
// Catalog of methods of interest
C = {…}
// seed basic sequences (e.g. “boolean b=false”, “int i=0”)
passing_tests = { … }
// valid instances
instances = {}
// exceptional tests
failing_tests = {}

Do until time limit expires:
  new_test = generate_new_test_sequence(C,passing_tests):
  try:
    r = execute_test(new_test,oracles)
    if (r !in instances)
      // sequence creates a previously unseen instance
      passing_tests.add(new_test)
      instances.add(r)
    else:
      // discard sequence
```

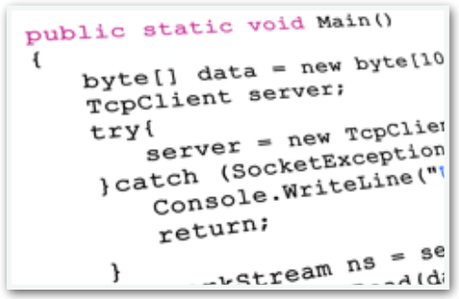
```
// Catalog of methods of interest
C = {…}
// seed basic sequences (e.g. “boolean b=false”, “int i=0”)
passing_tests = { … }
// valid instances
instances = {}
// exceptional tests
failing_tests = {}

Do until time limit expires:
  new_test = generate_new_test_sequence(C,passing_tests):
  try:
    r = execute_test(new_test,oracles)
    if (r !in instances)
      // sequence creates a previously unseen instance
      passing_tests.add(new_test)
      instances.add(r)
    else:
      // discard sequence
  catch OracleException:
      // failing test
      failing_tests.add(new_test)
```

```
// seed basic sequences (e.g. “boolean b=false”, “int i=0”)
passing_tests = { … }
// valid instances
instances = {}
// exceptional tests
failing_tests = {}

Do until time limit expires:
  new_test = generate_new_test_sequence(C,passing_tests):
  try:
    r = execute_test(new_test,oracles)
    if (r !in instances)
      // sequence creates a previously unseen instance
      passing_tests.add(new_test)
      instances.add(r)
    else:
      // discard sequence
  catch OracleException:
      // failing test
      failing_tests.add(new_test)
  catch Exception:
      passing_tests.add(new_test)

return passing_tests, failing_tests
```

Clasificamos en failing y passing tests

Generación Automática de Tests

Programa+Catálogo



Budget (tiempo,#tests)

Oráculos

Herramienta de Generación de Tests

JUnit

Test Suite

Random Testing para lenguajes Orientados a Objetos

- Enfoque más popular: *feedback-guided*
- Evitar secuencias redundantes
- Evitar secuencias que no producen instancias
- Utilizar oráculos implícitos para clasificar los tests generados
- Necesitamos una condición de parada (tiempo, tests, etc.)

Random Testing

- **Idea**: definamos nuestro “mono tipeador” y probemos nuestro programa usando sus inputs
- Necesitamos para ello de un Oráculo

Random Testing **Adaptativo**

- *Adaptive random testing* (Chen et al., 2004)
- Usa *feedback* de la ejecución de los Casos de test para *guiar* la generación de nuevos tests
- **Idea**: Generar casos de test que tenga una alta distancia a tests “no exitosos” (i.e. que no han detectado una falla)

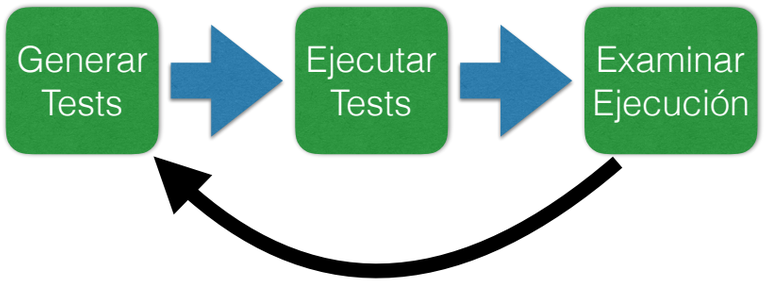
Random Testing para programas Orientados a Objetos

- Dado este método (Java)

```
public static Integer largest(LinkedList<Integer> list) {
  int index = 0;
  int max = Integer.MIN_VALUE;
  while (index <= list.size()-1) {
     if (list.get(index) > max) {
        max = list.get(index);
     }
     index++;
  }
  return max;
}
```

- La solución anterior puede usar `LinkedList()` o `null`
  - list=`null`
  - list=`[]` (la lista vacía)

Random Testing guiado por Feedback



Feedback de la ejecución del Test