软件分析与测试
第四次课

严俊

中国科学院大学
University of Chinese Academy of Sciences
ISCAS 中国科学院软件研究所
Institute of Software, Chinese Academy of Sciences

测试的要素

- 测试的要素
  - 对被测软件行为的描述
  - 被测软件功能的正确性的描述（Oracle）
  - 测试充分性
    - Coverage Criteria
- 如果缺乏这些要素
  - 随机测试
  - 蜕变测试

本次课内容

- 随机测试方法
- 蜕变测试技术
- 测试案例
  - 编译器测试
  - 浏览器测试

测试用例

- 测试的目标是为了找错
  - 大部分黑盒测试，我们不知道哪里会出错
- 充分性（多样性）
  - 两头下注
  - 覆盖面尽可能广→覆盖准则
- 区分性
  - 能够区分正确和错误的程序行为

测试用例的区分性

- SUT $f(x)$以及一个错误的版本 $f'(x)$

- 一个有区分度的测试集 $T = \{t_1, t_2, \ldots, t_n\}$
- 存在测试用例 $t \in T$，使得 $f(t) \neq f'(t)$

将条件表达式
x <= 1
写成了
x < 1
如何设计有区分度的测试用例？

Test case	x <= 1	x < 1
x = 0	T	T
x = 1	T	F
x = 2	F	F

例：公平性测试

- **决策系统中可能有不少不公平的行为**
  - 性别、种族、年龄、地域
  - 个体公平性：Such a discrimination exists when two individuals who differ only in the values of their protected attributes (such as, gender/race) while the values of their non-protected ones are exactly the same, get different decisions.
- **公平性测试**
  - 对于系统**z = f(x, y)**（**y**为保护属性），寻找两个测试用例 **(x1, y1)**、**(x2, y2)**, 其中 **x1 = x2, y1 ≠ y2**，使得输出 **z1 ≠ z2**
  - 例（机器翻译）

    The man is a doctor. 这个男人是医生。

    The woman is a doctor. 这个女人是护士。

例：满足MC/DC准则的测试集

- $S = a \wedge b$
- 测试集$t_1=(1, 1)$, $t_2=(1,0)$, $t_3=(0,1)$

故障类型	SUT	t_1	t_2	t_3
正确表达式	$a \wedge b$	1	0	0
单布尔运算符故障	$a \vee b$	1	1	1
	$a \wedge \neg b$	0	1	0
	$\neg a \wedge b$	0	0	1
多布尔运算符故障	$a \vee \neg b$	1	1	0
	$\neg a \vee b$	1	0	1
	$\neg a \wedge \neg b$	0	0	0
	$\neg a \vee \neg b$	0	1	1

随机测试

黑盒测试

- 等价类划分
  例：求平方根（负数，非负数）
- 边界值测试
  例：求平方根（最小的负数，绝对值很小的负数，0，绝对值很小的正数，最大的正数）
- 实际测试
  等价类：不知道
  边界值：-∞，+∞

随机测试

- 随机测试是通过在输入域中通过随机选择的方式来产生测试数据。
- 测试人员按自己的想法随手产生的测试用例也可以认为是随机测试。

```
int myAbs(int x) {
  if ( x > 0 ) {
    return x;
  }
  else {
    return x;
// bug: should be '-x'
  }
}
```

```
void testAbs(int n) {
  for ( int i = 0; i < n; i++ ) {
    int x = getRandomInput();
    int result = myAbs(x);
    assert( result >= 0);
  }
}
```

随机测试的优点

- 简单
  - 不需要过多的信息
  - 不需要开发复杂的程序
- 适应性广，几乎可以用于所有的测试方案
  - 因为不需要过多的信息
- 开销低，可以作为其他测试方法的补充
  - 算法复杂度已经到了最低的极限
- 较好的测试数据分布（测试多样性）

随机测试的用途

- 在没有足够的信息支持其他测试方法时
- 如果能够有办法不使用预期结果进行测试的验证
  - 如： 冒烟测试、蜕变测试等
- 作为其他测试方法的支持
  - 驱动SUT执行，用于收集信息进行动态分析
  - 先使用随机测试生成一批数据，再使用其他方法弥补随机测试遗漏的部分。
- 在刚开始进行测试时，初步发现错误

随机测试用例

- 对于简单的参数类型，如整数、 实数等，可以直接产生随机数据。

```
void testAbs(int n) {
  for ( int i = 0; i < n; i++ ) {
    int x = getRandomInput();
    int result = myAbs(x);
    assert( result >= 0);
  }
}
```

随机测试用例

- **复杂、高维度数据类型**
  - 简单的随机产生数据效率不高
  - **e.g.** 线性方程求解器 $a_1x_1 + a_2x_2 + \cdots + a_nx_n = b$
    科学计算器
- **复杂数据类型的随机数据生成**
  - 对被测软件的输入空间进行建模，为测试数据的随机生成创造条件。
  - 产生随机数，映射到输入空间的某个点，将该点转换为一个测试数据。

输入空间建模

- 每一个可能的输入在空间中都有一个对应的点；空间中每一个点都对应一个输入。
- 注意各种类型的数据在空间中分布的情况。例如，避免过多地生成失效数据。
- 注意计算机产生的随机数是实数，需要建立一种简单、方便的映射算法，把随机生成的数据（可能是多个数据）映射为输入数据。
- 所建立的空间模型，和程序需要的数据可能有差异，如忽略了某些重要的约束条件，需要对生成的测试数据进行再加工。

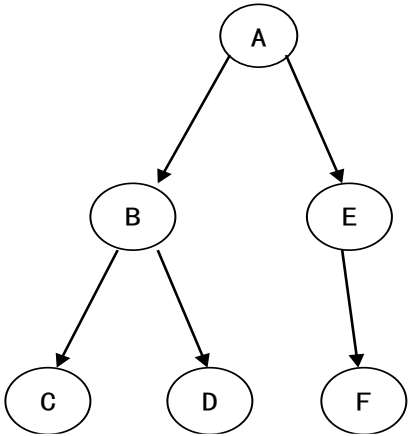
随机测试案例

- 为某个输入为二叉树的程序构造测试用例

  - 二叉树的中根序列和先根序列将唯一确定一棵二叉树；因此可以将二叉树的空间结构映射为一个 $\{1, ..., n\}$的随机序列。这个随机序列对应一棵二叉树的中根序列，而这棵二叉树的先根序列是$1, ..., n$ 。
  - 生成过程分为三个步骤，
    首先随机生成一个整 数n ，代表树的结点数目；
    生成长度为n的随机序列;
    根据随机序列构造一个二叉树。

    存在的问题：序列可能无法构造二叉树
    如先根ABC，中根CAB

中根：CBDAFE
先根：ABCDEF



案例-JUSTIN工具

- 自动生成Java单元测试数据的工具
  - 以Java字节码为输入
  - 以public方法为测试单元
  - 生成单元测试源码
  - 生成结果可在JUnit框架下编译执行
- 所需环境
  - JDK 1.8
  - Junit
  - 命令行
- 测试数据生成方法
  - 随机测试，模糊测试，……

- 方法参数的自动生成
  - 基本类型
    - 包含 int | short | byte | long | float | double | char | Boolean
    - String 类型的常量生成
  - 自定义类
    - 支持复杂对象参数的生成
  - 数组
    - 支持多维数组
    - 支持元素为基本类型和自定义类
    - 数组中自动添加元素
- 方法调用

Justin产生的测试用例

- 源代码

```
package cn.ios.test;
public class GasStation {
    public int canCompleteCircuit(int[] gas, int[] cost) {
        int [] rest = new int [gas.length];

        int res = 0;

        for(int i = 0; i < gas.length; ++i){
            rest[i] = gas[i] - cost[i];
            res += rest[i];
        }

        if(res < 0)
            return -1;
        int len = rest.length;
        for(int i = 0; i < rest.length; ++i){
            int start = (i+1)%len;
            int end = i;
            int total = rest[i];
            if(total < 0)
                continue;
            while(start != end){
                total += rest[start];
                if(total < 0)
                    break;
                start = (start + 1)%len;
            }
            if(total >= 0)
                return i;
            else
                continue;
        }
        return -1;

    }
}
```

- 生成的测试代码

```
package cn.ios.test;

import org.junit.Test;

public class GasStation_Test {

  @Test
  public void test_GasStation_canCompleteCircuit_0(){

      int[] intArray1 = {-1852899294, -27147850, -1423120474};
      int[] intArray2 = {1513584829};
      cn.ios.test.GasStation gasStation0 = new cn.ios.test.GasStation();
      gasStation0.canCompleteCircuit(intArray1, intArray2);

  }


}
```

Effectiveness Metrics

- P-measure - probability of detecting at least one failure
- E-measure - expected number of failures
- F-measure - expected number of test cases to detect the first failure

随机测试的缺点

- 测试数据发现程序失效的效率较低
- 难以确定测试的效果和停止测试的时机（区分性差）
- 大量测试用例
  - 大量的测试数据的预期输出很难获得（Oracle问题）

随机测试的效率

- 以下代码，采用随机测试触发错误的概率是多大？
- int32 $[-2^{31}, 2^{31}-1]$

```
int32 gcd( int32 m, int32 n ){
        int32 x = abs(m), y = abs(n);
        while( x != y ) {
                if ( x > y ) x -= y;
                else if ( x < y ) y -= x;
        }
        return x;
}
```

随机测试的效率问题

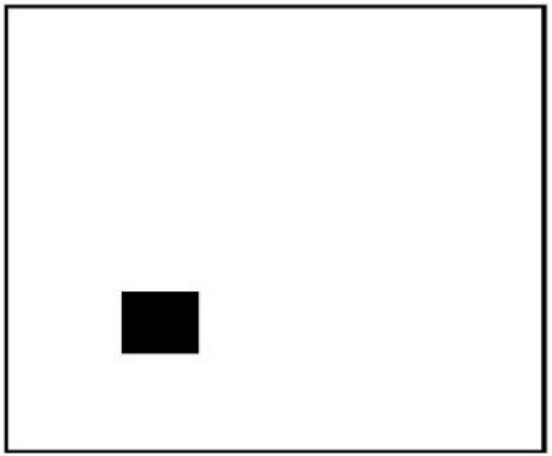
- 有人(Myers 1979)认为随机测试是效率最低的方法，因为它没有使用任何信息。
- 不恰当的分割测试（Partition testing）比随机测试的效果还差。
  - Dividing the input domain into partitions which are mutually exclusive; Selecting test cases from each partition
- 也有人（ P.J. Schroeder 2004等）比较了组合测试和随机测试，发现二者在某些案例上有类似的错误发现率

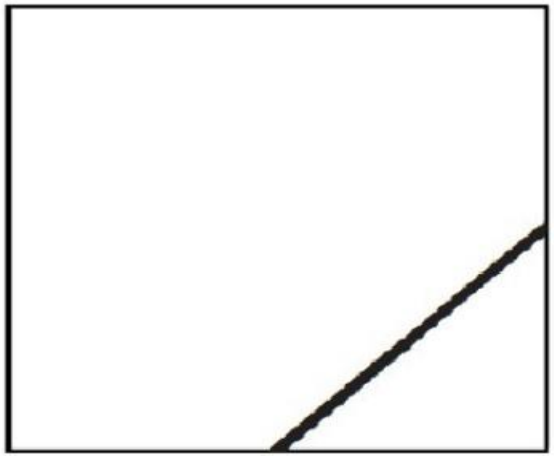
对随机测试的改进

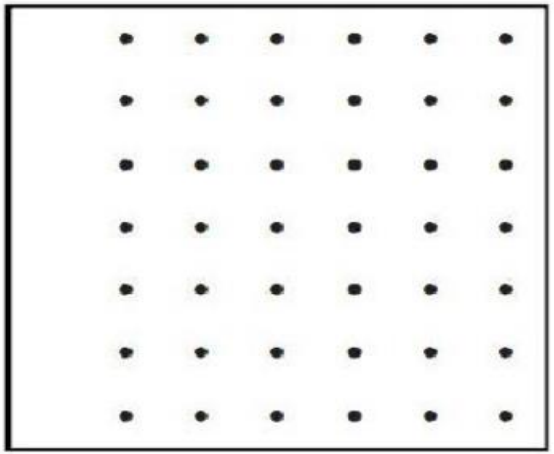
- 随机测试所生成的测试数据在发现错误方面是低效率的
- 改进主要的思想是： 根据某些信息，为随机测试提供生成的指导
  - 自适应随机测试(Adaptive Random Testing)：根据失效输入一般会集中出现在输入空间的某个局部区域的特征，提出将测试用例尽量均匀地分布在整个输入空间，从而提高测试用例发现错误的效率。
  - 概率测试：根据用户的需求，为对输入空间进行划分，不同部分用不同的概率产生测试用例。
  - 统计测试：先使用随机测试，当进行到一定程度时停止，再使用其他测试方法产生（补充）测试用例。

自适应随机测试

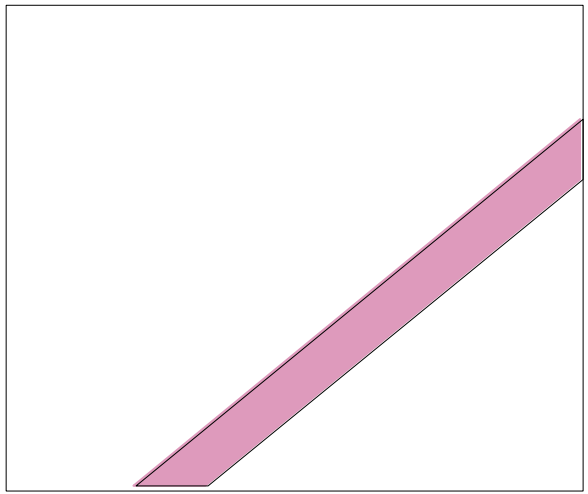
- 在被测程序中，失效输入通常会集中在输入空间的某几个地方，被称为失效区域。失效区域形状被归 类为3种。





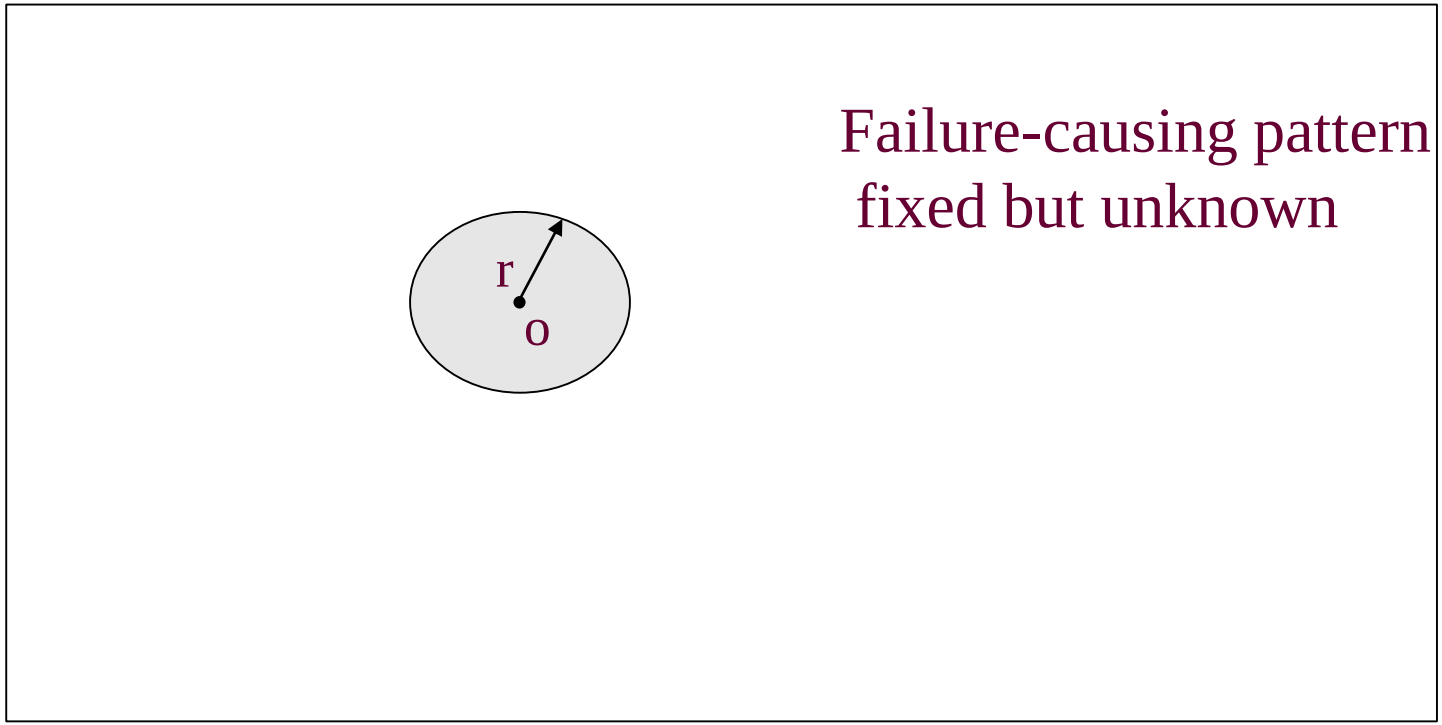


Contiguous Failure Pattern

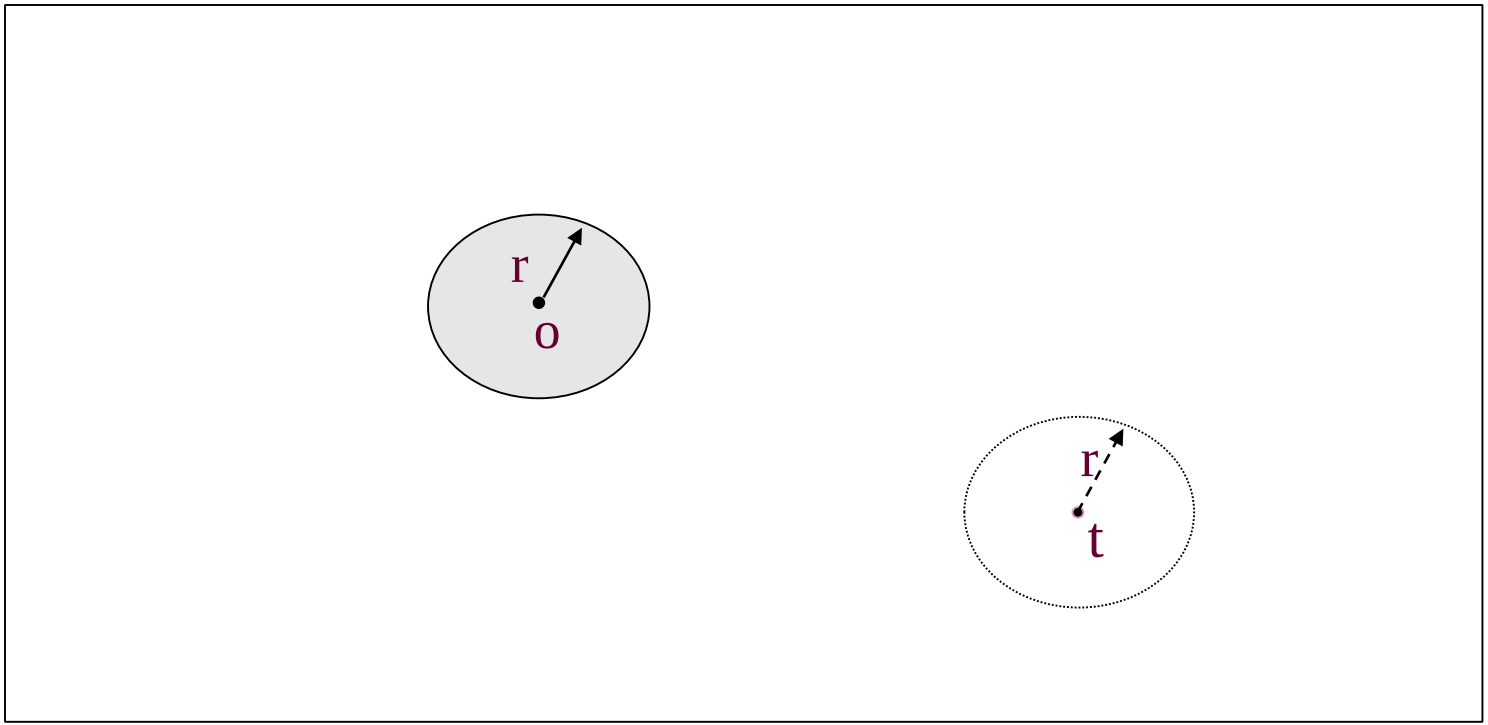


```
if (2*x – y > 10)
/* the correct statement is
 if (2*x – y > 20) */
    z = x/2 *y;
else
    z := x*y;
```

Intuition of ART



Intuition of ART



Intuition of ART

One corollary of the existence of contiguous failure regions is that "non-failure regions", that is, regions of the input domain where the software produces outputs according to specification, will also be contiguous. Therefore, given a set of previously executed test cases that have not revealed any failures, new test cases located away from these old ones are more likely to reveal failures — in other words, test cases should be more evenly spread throughout the input domain. Based on this intuition, Adaptive Random Testing (ART) was developed to improve the failure-detection effectiveness of random testing.

T. Y. Chen, F.-C. Kuo, R. G. Merkel and T. H. Tse. Adaptive Random Testing: the ART of Test Case Diversity. Journal of Systems and Software, Vol. 83(1), 60-66, 2010.

自适应随机测试

- **自适应随机测试的基本思想：每当生成一个新的测试用例的时候，尽量远离原有的测试用例。**
- **一些启发式策略：**
  - 基于距离的，FSCS (Fixed size of candidate set)。先随机生成几个测试用例（候选集），计算这些测试用例 和已经生成的测试用例的距离，选择距离最大的那个作为 新的测试用例
  - 基于排除域的，RRT (Rejected Region testing)。在每一个已生成的测试用例的附近指定一块排除域，随机产 生一个测试用例，如果这个测试用例不在任何一个排除域 中，那么它就成为新的测试用例
  - 基于分割的。利用已生成的测试用例把整个 输入空间分割成若干个部分，从最大的那个部 分中选择测试用例
  - 基于概率函数的。通过设定概率函数，将距离现有的测试用例较远的输入空间的点的概率设大，越近的设小

自适应随机测试的优点

- **能够大幅度提高所生成随机测试用例的效率**
  RT is easily implemented. However, depending on its implementation, ART can improve upon RT. CT does as well as ART whether or not t’ = t, but provides a valuable improvement in the cases when t’ = t. (C. Nie, IST 2015)
- **不需要额外增加信息。因为自适应随机测试运用的是测试中的一般规律**

自适应随机测试的缺点

- 计算复杂度大幅度提高
- 在实践应用中效果不佳

一些改进

- 由于自适应随机测试的计算复杂度太高，有一些方法改进生成效率。
  - 基于遗忘的方法（ forgetting）。在生成过程中，仅考虑最近生成的若干个测试用例（如100个）
  - 基于镜像的方法（mirroring）。在生成过程中，将输入 空间划分为对称的几个部分，仅对某个空间进行测试用例生成。当生成新的用例后，就映射到其他几个空间

常用的随机测试-模糊测试

- 用大量的畸形数据，触发系统的漏洞
- 构造随机或者不期望的数据作为测试输入
- Fuzzing, Fuzz Testing,
- Fuzzer, Fuzzing Tool

与随机测试的关系

模糊测试来自于随机测试

但有其特点：

- 输入格式固定
- 错误类型容易判定
- 对测试人员要求比较高
- 通常不是进行功能性测试，而是检查系统处理错误的能力，比如“入侵，破坏，崩溃”

history

- **Barton Miller (http://www.cs.wisc.edu/people/bart)**

  In 1988, Miller founded the field of Fuzz random software testing, …

- **Barton P. Miller, Lars Fredriksen, Bryan So. An Empirical Study of the Reliability of UNIX Utilities. Commun. ACM 33(12): 32-44 (1990)**

模糊测试用例生成策略

- 简单随机生成

  例：`While [ 1 ]; do cat /dev/urandom | nc -vv target port; done`

- 基于生成的 (generation-based fuzzer)
  - 采用模板描述程序的输入
- 基于变异的 (mutation-based fuzzer)
  - 从一个有效的输入开始，不断改变其中的内容
  - 依赖于已有的经过测试的数据包和文件

关于Fuzz testing的最老网站是
http://www.cs.wisc.edu/~bart/fuzz/fuzz.html 1990

对于windows NT的Fuzz test report中，

- 在random键盘或者鼠标输入的情况下，NT4.0有21%的程序crash
- 在random键盘或者鼠标输入的情况下，NT4.0有另外24%的程序会hung掉
- 在完全random的键盘或者鼠标的输入情况，几乎100%的程序会crash或者hung

- 蜕变测试

Testing the Non-Testable Programs

- A program is said to be testable if the output of any input can be verified
- To compute $41^{1/7}$
  - Suppose the computed output is 1.7
  - How can we know whether this output is correct or not?

Testable Programs

- 求解线性方程组

  $3x + 2y - z = 4$

  $x - 2y - 2z = -9$

  $2x + y + z = 7$

  Suppose the solutions x=1, y=2 and z=3

- To find the values of x such that

  x**67 + 3*(x**46) – x**37 +4.5 =0

  Suppose the solutions for x are: 2.17, 6.5, ..

Non-Testable Programs

- A program is said to be non-testable if the output of any input cannot be verified (or *cannot be verified in practice*)
- Example
  - A weather forecasting system which reports the amount of rain for a specific date
  - A clinical x-ray system
  - Compute the average for 10 million real numbers

Sine function

- *sin* function
  - *sin*(0^o)=0
  - *sin*(30^o)=0.5
- Suppose the program returns:

  sin(29.8^o)=0.49876 correct?

Shortest path program

- **Shortest path program *SP*(*G*, *a*, *b*)**
  - **where *G* is a graph, *a* is the starting node and *b* is the destination node**
  - ***SP*(*G*, *a*, *b*) returns a path like: *a* – *x* – *y* - …. - *s* – *t* – *b***
- **验证方法?**
  - **Find all possible paths from node *a* to node *b***
  - **Check against all these possible path to see whether *SP*(*G*, *a*, *b*): *a* – *x* – *y* - …. - *s* – *t* – *b* is the shortest**

Test Oracle

- A mechanism or procedure against which the computed outputs could be verified
- Test oracles are available but too expensive to be applied

Metamorphic Testing （MT）

- A simple but effective method to alleviate the test oracle problem
- Though we do not know the correctness of the output of any individual input, we may know the relation between some related inputs and their outputs

Sine function

- Suppose *sin*(29.8^o) returns 0.49876
- *sin* function has the following properties
  - *sin*(x)= *sin*(x+360)
- Compute 29.8^o + 360^o = 389.8^o
- Execute the program with 389.8^o as input
- Check whether *sin*(29.8^o) = *sin*(389.8^o)

Shortest path problem *SP*(*G*, *a*, *b*)

- **Suppose the program returns:**
  - **|*SP*(*G*, *a*, *b*)| = 1,000,001 correct or incorrect?**
- **Shortest Path Problem has the following properties:**
  - **|*SP*(*G*, *a*, *b*)| = |*SP*(*G*, *a*, *c*)| +|*SP*(*G*, *c*, *b*)| where *c* is a node in *SP*(*G*, *a*, *b*)**
  - **|*SP*(*G*, *a*, *b*)| = |*SP*(*G*, *b*, *a*)|**
- **Execute *SP*(*G*, *a*, *c*), *SP*(*G*, *c*, *b*) and *SP*(*G*, *b*, *a*)**

Metamorphic Testing (A Simplified Form)

- Define source (initial) test cases using some test case selection strategies
- Identify some properties of the problem (referred to as the metamorphic relations)
- Construct follow-up test cases from the source test cases with reference to the identified metamorphic relations
- Verify the metamorphic relations using the computed outputs

Applications of MT

- Bioinformatics programs
- Embedded systems
- Machine learning software
- Optimization systems
- Compilers
- ……
- …….

Interesting Results

- **Siemens suite**
  - **print_token, schedule, and schedule_2**
- **Compiler**
  - **Compiler Validation via Equivalence Modulo Inputs，PLDI 2014. Best Paper Award**
  - ***If programs P and P' are equivalent with respect to input I, then their object codes are equivalent with respect to I.***
- **Machine learning tool – Weka**
- **…….**

Metamorphic Relations (MR)

- Select an input
- Modify it, hopefully that the relevant change of output will be somehow predictable.

If yes, any generalisation?

If yes, then identify an MR

Example 1

To find the average of a series of integers

Input is: {3, 7, 12, 6, 8, 6, 3, 5, 15, 4}

What are the possible MRs?

Example 2

Shortest Path – SP (*G*, *a*, *b*)

Output is: *a* – *i* – *j* – *k* – ….. – *p* – *q* – *r* – *b*

What are the MRs

Example 3

Greedy Algorithm on Set Covering Problem

- To find the smallest set of keys that can open all doors

$$M_{KL} = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & k_1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & k_2 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & k_3 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & k_4 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & k_5 \\ l_1 & l_2 & l_3 & l_4 & l_5 & l_6 & l_7 & l_8 & l_9 & \end{pmatrix}$$

- Greedy algorithm find O = [k4, k2, k5, k3]
- global minimum solution is [k2, k3, k5]

蜕变关系

- MR1 (Interchanging columns related to the key-lock relationship)
- MR2 (Adding a useless key row)
- MR3 (Adding an insecure lock column)
- MR4 (Rearranging rows corresponding to the selected keys on top while preserving their order).
- MR5 (Adding a combined key of two consecutively selected keys).
- MR6 (Excluding a selected key other than the first selected key while preserving the order of the remaining selected keys).
- MR7 (Deleting a selected key while preserving the order of the remaining selected keys).
- MR8 (Adding an exclusive lock to an unselected key)
- MR9 (Adding an exclusive lock to an unselected key while preserving the order of the previously selected keys)

A. C. Barus *et al.*,Testing of Heuristic Methods: A Case Study of Greedy Algorithm

MR5, MR6

Adding a combined key of two consecutively selected keys

k6=k2+k5

Excluding a selected key other than the first selected key while preserving the order of the remaining selected keys

excluding k5

$$M'(\mathrm{MR5}) = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & k_1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & k_2 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & k_3 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & k_4 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & k_5 \\ 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & k_6 \\ l_1 & l_2 & l_3 & l_4 & l_5 & l_6 & l_7 & l_8 & l_9 & l_{10} & l_{11} & \end{pmatrix}$$

$$M'(\mathrm{MR6}) = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & k_1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & k_2 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & k_3 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & k_4 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & k_5 \\ l_1 & l_2 & l_3 & l_4 & l_5 & l_6 & l_7 & l_8 & l_9 & \end{pmatrix}$$

关键：选择蜕变关系

- 可以根据初始测试用例的输出设计后续测试用例
- 优先选择输入关系较为复杂的蜕变关系
- 优先选择使得输入中对应的变元数量较多的蜕变关系
- 组合蜕变关系的检测错误能力较单个蜕变关系强
- 包含的检测程序语义丰富
- 与典型程序或者算法使用的策略越相似，检测效果越有限

案例：公平性测试

- 决策系统中可能有不少不公平的行为
  - 性别、种族、年龄、地域
- 公平性测试
  - 对于系统z = f(x, y) （y为保护属性），寻找两个测试用例 (x1, y1)、 (x2, y2), 其中 x1 = x2, y1 ≠ y2，使得 输出 z1 ≠ z2
  - 蜕变关系：
    x1 = x2 -> z1 = z2

案例：自然场景下的文本定位系统

- 功能：定位文字位置，输出一张图片上所有包围框的位置，使用四个顶点的坐标表示。

（a）

700, 98, 659, 95, 660, 80, 701, 83
695, 157, 638, 154, 639, 135, 696, 139
723, 156, 695, 154, 697, 138, 724, 141
635, 169, 635, 154, 662, 154, 662, 169
693, 171, 665, 170, 665, 154, 693, 155
819, 299, 786, 297, 788, 281, 820, 283
820, 302, 819, 284, 873, 283, 873, 301
902, 303, 873, 300, 874, 283, 903, 285
903, 339, 786, 336, 786, 299, 904, 301
789, 366, 789, 335, 865, 335, 865, 366
853, 467, 853, 453, 902, 453, 902, 467

（b）

怎么测试？

- 如何设计蜕变关系？

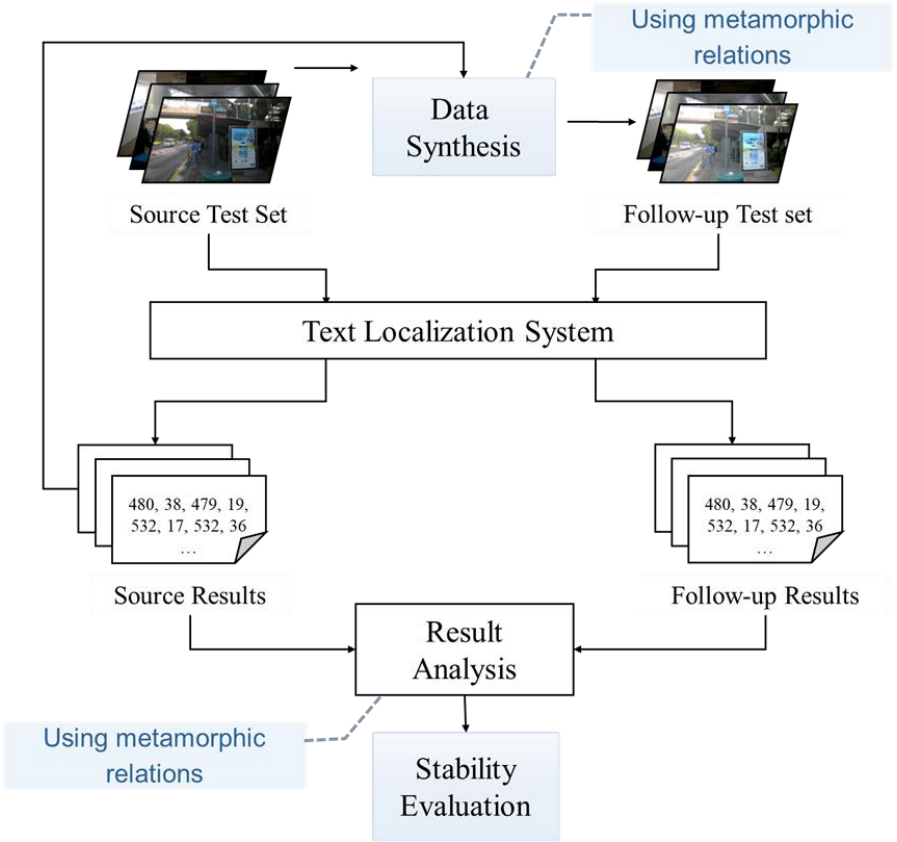
文本定位系统测试（JSS 2021）

保留语义的蜕变关系

- Increasing brightness (MR_{ib})
- Decreasing brightness (MR_{db})
- Channel switch (MR_{cs})

非保留语义的蜕变关系

- Perspective transformation (MR_{pt})
- Watermarking (MR_{wm})
- Masking (MR_{ma})



R. Yan, S. Wang, et al., Stability evaluation for text localization systems via metamorphic testing, Journal of Systems and Software, Vol 181, 2021

实验设计

- 集合相似度

  全部蜕变样本与原样本结果的平均匹配率

- 检测率（水印蜕变）

  成功检测到水印的样本占总数的比例

- 未检测率（遮盖蜕变）

  遮盖样本重新检测出样本的数量占总数的比例

The performance of three academic systems with the dataset of ICDAR 2015 (ICDAR2015, 2015).

Name	Recall	Precision	F-Score	Method
PSENet	85.22%	89.30%	87.21%	segmentation
PixelLink	83.77%	86.65%	85.19%	segmentation
EAST	77.32%	84.66%	80.83%	regression

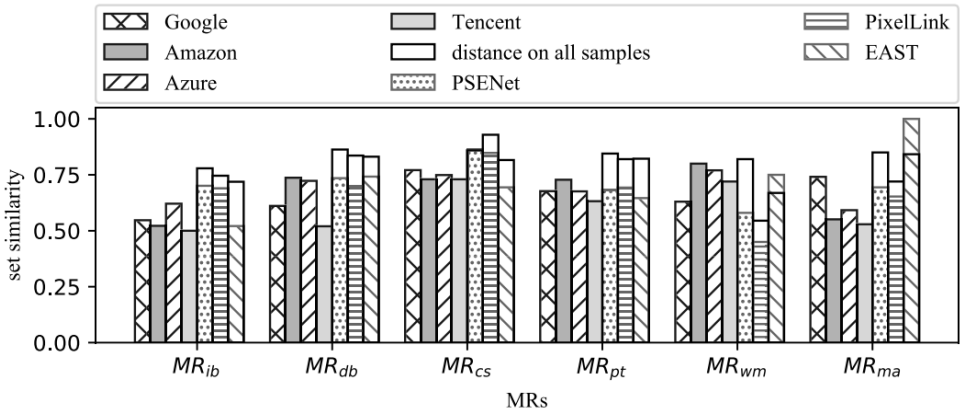
Investigated commercial systems.

Platform	API Name	Version/Last used
Google Cloud Platform	Vision AI	Apr-20
Amazon Web Services	Amazon Rekognition	Apr-20
Azure	Cognitive services	Apr-20
Tencent	GeneralAccurateOCR	v2018-11-19

测试结果

- 学术系统的实验结果
  - 三个系统均对亮度调节更敏感。
  - PixelLink在识别多样化颜色样本方面更具有优势，处理微小变换较其他两个更弱。
  - EAST同样对水印蜕变很敏感，但在六种不同蜕变关系上结果差异不大。
- 商用系统的实验结果
  - GCP在通道蜕变和遮盖蜕变方面表现更为出色。
  - AWS在暗度蜕变和透视蜕变方面表现出色，表明在任意形状的文本上具有优势。
  - Azure 总体上做得很好。

MRs	PSENet	PixelLink	EAST
MR_{ib}	0.779	0.746	0.719
MR_{db}	0.863	0.836	0.831
MR_{cs}	0.859	0.929	0.816
MR_{pt}	0.845	0.820	0.822
MR_{wm}	0.820	0.545	0.669
MR_{ma}	0.850	0.720	0.842



Testing Compilers

GraphicsFuzz

MOBILE

Google acquires GraphicsFuzz for its mobile graphics card benchmarking

KYLE WIGGERS @KYLE_L_WIGGERS AUGUST 6, 2018 9:51 AM

- **Metamorphic Testing of Drivers**
- GraphicsFuzz's three-person team — Alastair Donaldson, Hugues Evrard and Paul Thomson — will join Google's Android Graphics Team to "integrate their graphics driver testing technology within the Android ecosystem"

国科火 ISCAS

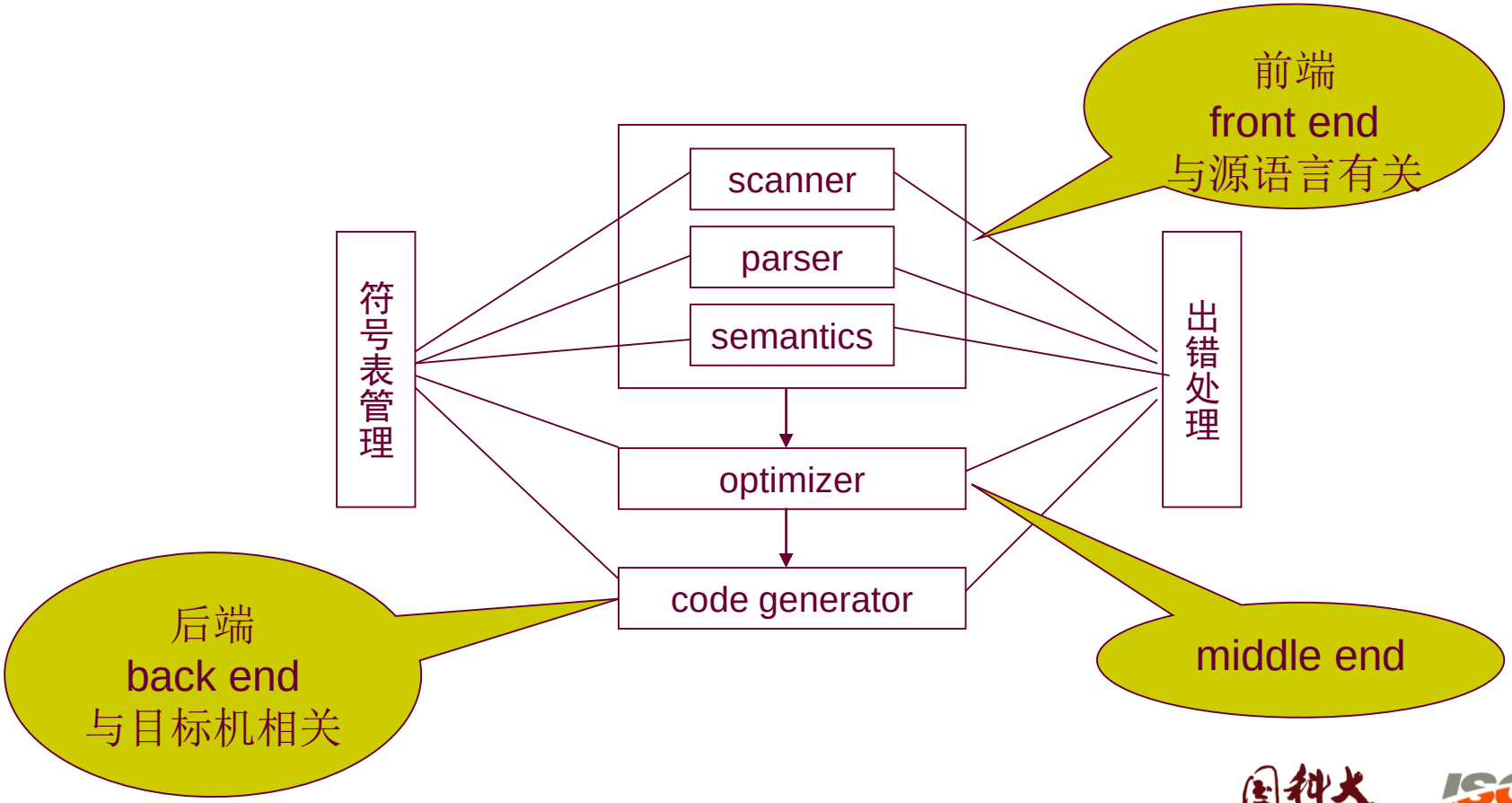
ShaderTest GLES

- Its reference shaders produce an image, after which ShaderTest GLES applies semantics-preserving transformations to the source code and compares the output. If the two results don't match, …
- It's been used to uncover vulnerabilities in phones like the Samsung Galaxy S6 and S9. In the Galaxy S9's case, GraphicsFuzz discovered an exploit in Qualcomm's Adreno 630 graphics driver.

编译器 (compiler)

- 编译器是将用一种编程语言（源语言）编写的计算机代码转换成另一种计算机语言（目标语言）的计算机软件。
- source code in a high-level programming language (e.g., Java, C, C++) to target code in a lower level language
- Ex. `bubble_bad.c` → `a.out`

编译程序的组成结构


前端
front end
与源语言有关
scanner
parser
semantics
符号表管理
出错处理
optimizer
code generator
后端
back end
与目标机相关
middle end

Testing Compilers

- 编译器还会有错？
- The CompCert project investigates the formal verification of realistic compilers usable for critical embedded software. (http://compcert.inria.fr/)
- How can we test compilers?
  - GCC's torture test suite

Random Testing

- **Randomly generate a set of programs and compile them using the compiler.**
  - **Prog_1 → exec_1**
  - **Prog_2 → exec_2**
  - **Prog_3 → exec_3**

- **Challenges:**
  - **generating what kind of programs ?**
  - **How can we check the results ? (Oracle)**

差分测试 (Differential Testing)

- 同样的测试输入，对于不同的系统，应该有一致的结果
  - 同一系统的不同版本（纵向）
  - 不同的系统（横向）
- 测试应用
  - 编译器测试
  - 浏览器测试
  - ……

Csmith

- a randomized test-case generator that supports compiler bug-hunting using *differential testing*. [Yang et al. PLDI2011]
- Csmith generates a C program; a test harness then compiles the program using several compilers, runs the executables, and compares the outputs.
- It generates random programs that are expressive — containing complex code using many C language features.

Csmith

- It generates tests that explore atypical combinations of C language features. Atypical code is simply underrepresented in fixed compiler test suites.

- ... we have found and reported more than 325 bugs in mainstream C compilers including GCC, LLVM, and commercial tools. … Twenty-five of our reported GCC bugs have been classified as P1.

Ex. 1

```
int foo (void) {
  signed char x = 1;
  unsigned char y = 255;
  return x > y;
}
```

- `$ gcc ex1.c -O2`
- `$ ./a.out`
- `0`
- gcc v 4.4.1 (Ubuntu 4.4.1-4ubuntu9)

Figure 1. We found a bug in the version of GCC that shipped with Ubuntu Linux 8.04.1 for x86. At all optimization levels it compiles this function to return 1; the correct result is 0.

T.Y.Chen -- Testing Compilers

- **Compiler Validation via Equivalence Modulo Inputs, V. Le, M. Afshari and Z. Su, PLDI 2014, 216–226, 2014. Best Paper Award**

Their testing method is basically a MT method

Its MR is:

If programs P and P' are equivalent with respect to input I,

then their object codes are equivalent with respect to I.

http://blog.regehr.org.archives/1161

Metamorphic testing

- **bad ideas: adding layers of parens, rewriting (x+y) to be (y+x), rewriting x to be (x+0), etc.**
- **The reason that these are bad ideas is that the changes will be trivially undone by the optimizer, resulting in poor testing of the optimizer logic**

https://blog.regehr.org/archives/1161 (John Regehr)

Equivalence modulo inputs (EMI)

- "profile and mutate" strategy
- pruning unexecuted code

- Our extensive testing in eleven months has led to 147 confirmed, unique bug reports for GCC and LLVM alone.

[Le, Afshari, Su PLDI 2014]

国科大 ISCAS

EMI

- **the program transformation is removal of dead code -- dynamically (code that is dead in some particular run).**
  1. 在任意输入上运行C程序，使用编译器插装检查执行哪些行
  2. 创建一个新的C程序，缺少在步骤1中没有执行的代码片段
  3. 在相同的输入上运行新程序。如果编译器的输出发生了变化，则报告编译器的错误。

John Regehr (https://blog.regehr.org/archives/1161)

```c
struct tiny { char c; char d; char e; };
f(int n, struct tiny x, struct tiny y, struct tiny z,
      long l) {
  if (x.c != 10) abort();
  if (x.d != 20) abort();
  if (x.e != 30) abort();
  if (y.c != 11) abort();
  if (y.d != 21) abort();
  if (y.e != 31) abort();
  if (z.c != 12) abort();
  if (z.d != 22) abort();
  if (z.e != 32) abort();
  if (l != 123) abort();
}
main() {
  struct tiny x[3];
  x[0].c = 10;
  x[1].c = 11;
  x[2].c = 12;
  x[0].d = 20;
  x[1].d = 21;
  x[2].d = 22;
  x[0].e = 30;
  x[1].e = 31;
  x[2].e = 32;
  f(3, x[0], x[1], x[2], (long)123);
  exit(0);
}
```

(a) Test `931004-11.c` from the GCC test suite; it compiles correctly by all compilers tested.

```c
struct tiny { char c; char d; char e; };
f(int n, struct tiny x, struct tiny y, struct tiny z,
      long l) {
  if (x.c != 10) /* deleted */;
  if (x.d != 20) abort();
  if (x.e != 30) /* deleted */;
  if (y.c != 11) abort();
  if (y.d != 21) abort();
  if (y.e != 31) /* deleted */;
  if (z.c != 12) abort();
  if (z.d != 22) /* deleted */;
  if (z.e != 32) abort();
  if (l != 123) /* deleted */;
}
main() {
  struct tiny x[3];
  x[0].c = 10;
  x[1].c = 11;
  x[2].c = 12;
  x[0].d = 20;
  x[1].d = 21;
  x[2].d = 22;
  x[0].e = 30;
  x[1].e = 31;
  x[2].e = 32;
  f(3, x[0], x[1], x[2], (long)123);
  exit(0);
}
```

(b) Test case produced by Orion by transforming the program in Figure 1a, triggering a bug in Clang.

- 搜索引擎的测试

搜索引擎的测试

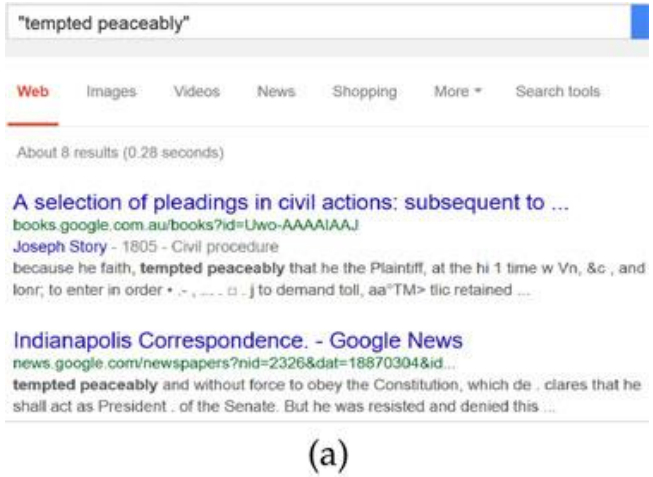
- 输入：字符串 （with some options）
- 输出：一系列的 websites (URLs)

- 输出结果对不对？

Zhiquan Zhou et al. Metamorphic Testing for Software Quality Assessment: A Study of Search Engines. TSE 42(3): 264-284 (2016)

Metamorphic Relation: MPSite

- an additional criterion: all results be retrieved from domain
  - “side effect of antibiotics in babies”
  - “side effect of antibiotics in babies” site:uk



(a)

(b)

Metamorphic Relation: MPTitle

- Synonyms
- title extracted from the search engine's results screen
  - "tempted peaceably"
  - "tempted peaceably" A selection of pleadings in civil actions subsequent to
- follow-up query

MPTitle

- another follow-up query
  - [“tempted peaceably” Indianapolis Correspondence Google News]
- For this query, however, Google did not find any result, as shown in the screenshot below:

No results found for **"tempted peaceably" Indianapolis Correspondence Google News**.

Metamorphic Relation: MPReverseJD

- reverse the order
  - [“Vincent Van Gogh” AND “Elvis Presley” AND “Albert Einstein” AND “Plato”]
  - [“Plato” AND “Albert Einstein” AND “Elvis Presley” AND “Vincent Van Gogh.”]

- **Stability**: two result sets should have a large intersection.

- Use the metric **Jaccard similarity coefficient**

Metamorphic Relation: SwapJD

- The source query A contains only two words (without quotation marks) and the follow-up query B is constructed by swapping the two words.
- The similarity can be measured by calculating the Jaccard coefficient of the top x results in the two result lists

SwapJD

- **The results:**

Bing found 25 results for [Seoul traffic] (left),
but 0 results for [traffic Seoul] (right).

Metamorphic Relation: Top1Absent

- extending the idea of MPSite
  - “stanford”
  - ”stanford” site:edu

总体结论

List of the Best and Worst Performers

MR and usage pattern	The best performer	The worst performer
MPSite (English)	Google	Baidu
MPSite (Chinese)	Google and Baidu	Bing
MPTitle (English)	Bing	Baidu
MPTitle (Chinese)	Baidu	CBing
MPReverseJD (people)	Google	CBing and Baidu
MPReverseJD (companies)	Google	Baidu
MPReverseJD (drugs)	Google	Baidu
SwapJD (Universal)	Google	Bing and Baidu
Top1Absent	Google	Bing

蜕变测试的本质

- 对于被测程序p(x), 假定我们有蜕变关系

  $C_1(x_1, x_2, \ldots, x_n) \rightarrow C_2(p(x_1), p(x_2), \ldots, p(x_n))$;

- 构造一个test_main函数

```
test_main(x1, x2, …, xn)
{
    y1 = p(x1);
    y2 = p(x2);
    ...
    yn = p(xn);

    assert( C1(x1, x2, …, xn) → C2(y1, y2, …, yn) );
}
```

断言 Assertions

- 例：assert(x != 0)；assert(p!=NULL);

- *Tony Hoare, COMPSAC 2002*

  Using Assertions

  - as a means of documentation
  - to avoid system crash
  - as test oracle
  - for modular verification
  - ...

References

- T. Y. Chen, S. C, Cheung, and S. M. Yiu. 1998. Metamorphic testing: A New Approach for Generating Next Test Cases. Technical Report HKUSTCS98-01. Department of Computer Science, Hong Kong University of Science and Technology.
- Segura, S., Fraser, G., Sanchez, A. B., & Ruiz-Cortés, A. (2016). A survey on metamorphic testing. *IEEE Transactions on software engineering*, *42*(9), 805-824.

小结

- 随机测试方法
  - 缺乏对被测系统描述
  - 避免复杂的分析
- 蜕变测试技术
  - 缓解oracle问题
- 两种测试方法结合使用的案例
  - 编译器测试
  - 浏览器测试