بنام خدا

دانشکده مهندسی کامپیوتر

درس آزمون نرم‌افزار

تمرین سوم

اعضاء گروه:

امیرحسین کارگران خوزانی (۹۹۲۰۱۱۱۹)

سیدسجاد میرزابابایی (۹۹۲۱۰۱۴۲)

رامتین باقری (۹۹۳۰۱۹۳۸)

تیر ۱۴۰۰

سوال ۱)

قواعد RIP را با توجه به موتانت داده شده می‌نویسیم:

Reachability: $\neg(\text{numbers} = \emptyset \vee \text{val}\ not\ in\ \text{numbers})$

اگر آرایه خالی باشد یا مقدار val در آرایه نباشد، دسترسی به موتانت نخواهیم داشت.

Infection: $\text{val} \neq \text{numbers}.indexOf(\text{val})$

اگر مقداری که می‌خواهیم پیدا کنیم با شناسه قرارگیری آن در آرایه برابر باشد، آلودگی اتفاق نمی‌افتد.

Propgation: $True$

زیرا تحت هر شرایطی خطا منتشر می‌شود.

حال به سراغ زیر بخش‌های سوال می‌رویم:

زیربخش الف)

ورودی‌ای که باعث نقض Reachability شود، می‌بایست با معکوس رابطه آن سازگار باشد، یعنی:

$$\text{numbers} = \emptyset \vee \text{val}\ not\ in\ \text{numbers}$$

یک ورودی با این شرایط می‌تواند numbers=[] و val=3 باشد.

زیربخش ب)

ورودی درخواستی باید با رابطه Reachability ∧ ¬Infection سازگار باشد. رابطه باز شده، برابر خواهد بود با:

$$(\text{numbers} \neq \emptyset \wedge \text{val}\ in\ \text{numbers}) \wedge \text{val} = \text{numbers}.indexOf(\text{val})$$

یک ورودی با این شرایط می‌تواند numbers=[6,1] و val=1 باشد.

زیربخش ج)

ورودی درخواستی باید با رابطه Reachability ∧ Infection ∧ ¬Propagation سازگار باشد. یعنی:

$$(\text{numbers} \neq \emptyset \wedge \text{val}\ in\ \text{numbers}) \wedge \text{val} \neq \text{numbers}.indexOf(\text{val}) \wedge \text{False}$$

از آنجایی که عبارت بالا معادل با False است، پس نمی‌توان ورودی‌ای برای آن متصور شود. به عبارت دیگر، این حالت هیچگاه برقرار نخواهد شد.

زیربخش د)

ورودی‌ای که باعث kill شدن موتانت شود، باید به گونه‌ای باشد که اگر آن را به دو نسخه عادی و موتانت شده برنامه بدهیم، خروجی‌های متفاوتی حاصل شود. برای اینکار، کافی است تا شرایط زیر برقرار باشد:

$$\text{Reachability} \wedge \text{Infection} \wedge \text{Propagation} = \text{Reachability} \wedge \text{Infection}$$

به عبارت دیگر:

$$(\text{numbers} \neq \emptyset \wedge \text{val}\ in\ \text{numbers}) \wedge \text{val} \neq \text{numbers}.indexOf(\text{val})$$

یک ورودی با شرایط بالا می‌تواند numbers=[5,4,1] و val=4 باشد.

سوال ۲)

بخش الف)

```
public static int power(int left, int right)
{
    //*********************************************
    // Raises left to the power of right
    // precondition : right >= 0
    // postcondition : Return left**right
    //*********************************************

    int rslt;
    rslt = left;
 M1 rslt = failOnZero(left);
 M2 rslt = right;
    if (right == 0)
 M3 if (right trueOp 0)
 M4 if (right falseOp 0)
    {
        rslt = 1;
 M5     rslt -= 1;
 M6     Bomb();
    }
    else
    {
        for (int i=2; i<=right; i++)
 M7     for (int i=0; i<=right; i++)
 M8     for (int i=2; i>right; i++)
            rslt = rslt * left;
 M9         rslt = rslt * right;
 M10        rslt = rslt + left;
 M11        rslt = rslt * -left;
 M12        Bomb();
    }
    return rslt;
}
```

بخش ب)

در قطعه کد مورد سوال، ۵ عبارت (خطوط ۱۰، ۱۱، ۱۳، ۱۷ و ۱۸) وجود دارند که می‌توان آنها را به عنوان خطای برنامه تغییر داد. برای شمارش تعداد موتانت‌ها به صورت تقریبی، می‌توانیم از عملگرهای موتانت داخل اسلایدهای درس رجوع کنیم و تعداد خطوطی که می‌توان از این عملگرها استفاده کرد را به ازای هر عملگر حساب کنیم:

دسته عملگر	نام عملگر	موتانت‌های احتمالی	تعداد تقریبی موتانت‌ها
Absolute Value Insertion	abs()	rslt=abs(left) rslt=abs(rslt * left)	۵

	rslt=abs(rslt) * left rslt=rslt * abs(left) return abs(rslt)		
۵	rslt=negAbs(1) rslt=negAbs (rslt * left) rslt=negAbs (rslt) * left rslt=rslt * negAbs(left) return negAbs(rslt)	negAbs()	
۳	rslt=failOnZero(left) rslt=failOnZero(rslt * left) return failOnZero(rslt)	failOnZero()	
۱	rslt = rslt leftOp left	leftOp	**Arithmetic Operator Replacement**
۱	rslt = rslt rightOp left	rightOp	
۵	rslt = rslt + left rslt = rslt % left rslt = rslt ** left rslt = rslt – left rslt = rslt / left	+,-,*,**,/,%	
۲	if (right falseOp 0) for (… i falseOp right…)	falseOp	**Relational Operator Replacement**
۲	if (right trueOp 0) for (… i trueOp right…)	trueOp	

۱۰	if (right < 0)
	if (right <= 0)
	if (right > 0)
	if (right >= 0)
	if (right ≠ 0)
	for (… i < right …)
	for (… i ≠ right …)
	for (… i >= right …)
	for (… i > right …)
	for (… i == right …)
		<,<=,>,>=,≠	

۰	-	falseOp	**Conditional Operator Replacement**
۰	-	trueOp	
۰	-	leftOp	
۰	-	rightOp	
۰	-	leftOp	**Shift Operator Replacement**

٥	-	leftOp	**Logical Operator Replacement**
٥	-	rightOp	
١٥	rslt += left rslt -= left rslt *= left rslt /= left rslt %= left rslt += 1 rslt -= 1 rslt *= 1 rslt /= 1 rslt %= 1 rslt += rslt * left rslt -= rslt * left rslt *= rslt * left rslt /= rslt * left rslt %= rslt * left	+=, -=, *=, /=, %=	**Assignment Operator Replacement**
٧	rslt = -left rslt = -1 for (int i=-2…) rslt = -(rslt * left) rslt = (-rslt) * left rslt = rslt * (-left) return -rslt	-	**Unary Operator Insertion**
٥	-	-	**Unary Operator Deletion**
١٠	rslt = right rslt = right * left rslt = rslt * rslt right = rslt * left right = right * left right = right * right left = rslt * left left = right * left left = right * rslt left = left * left	-	**Scalar Variable Replacement**
٥	-	-	**Bomb Statement Replacement**

مجموع	۷۱

با محاسبه موتانت‌های قابل ساخت برای شبه کد مورد سوال، به تعداد تقریبی ۷۱ رسیدیم. البته مجددا تاکید می‌کنیم که این تعداد به صورت تقریبی است و موتانت‌هایی در شمارش در نظر گرفته شده که از نظر ما معنادار باشند.

بخش ج)

#Mutant	Input		Mutated Output	Original Output	Mutant Killed?
	left	right			
1	0	0	ERROR	1	✓
2	3	2	6	9	✓
3	3	2	1	9	✓
4	0	0	0	1	✓
5	0	0	-1	1	✓
6	0	0	EXCEPTION	1	✓
7	3	2	81	9	✓
8	3	2	3	9	✓
9	3	2	6	9	✓
10	3	2	6	8	✓
11	3	2	-9	9	✓
12	3	2	EXCEPTION	9	✓

همانطور که در جدول بالا مشخص است، می‌توان با دو ورودی تمام ۱۲ موتانت تولید شده را کشت. بدیهی است که تعداد کم ورودی‌های مورد نیاز به علت حجم کم برنامه است.

سوال ۳)

#	Rule
1	Input ::= statement
2	statement ::= expr
3 – 4	expr ::= LHS RHS AO | AO LHS RHS
5	LHS ::= digit
6	RHS ::= digit
7 – 10	AO ::= “Div” | “Mod” | “GCD” | “LCM”
11 – 20	digit ::= “0” | “1” | “2” | “3” | “4” | “5” | “6” | “7” | “8” | “9”

بخش الف)

TR={"Div", "Mod", "GCD", "LCM", "0", "1", "2", "3", "4", "5", "6", "7", "8", "9"}

TC = {0 1 Div, 2 3 Mod, GCD 4 5, LCM 6 7, Div 8 9}

بخش ب)

#	Rule
1	Input ::= statement
2	statement ::= expr
3 – 4	expr ::= LHS RHS AO | AO LHS RHS
5	LHS ::= ~~digit~~ AO (Non-terminal Replacement)
6	RHS ::= digit
7 – 10	AO ::= “Div” | “Mod” | “GCD” | “LCM”
11 – 20	digit ::= “0” | “1” | “2” | “3” | “4” | “5” | “6” | “7” | “8” | “9”

بخش ج)

TR = {P1, P2, P3, …, P20}

TC = {Div 0 Mod, GDC LCM 1, Div 2 LCM, Mod LCM 3, Div Mod 4, Mod LCM 5, Mod LCM 6, Mod LCM 7, Mod LCM 8, Mod LCM 9}

سوال چهارم)

Introduction to Software Testing

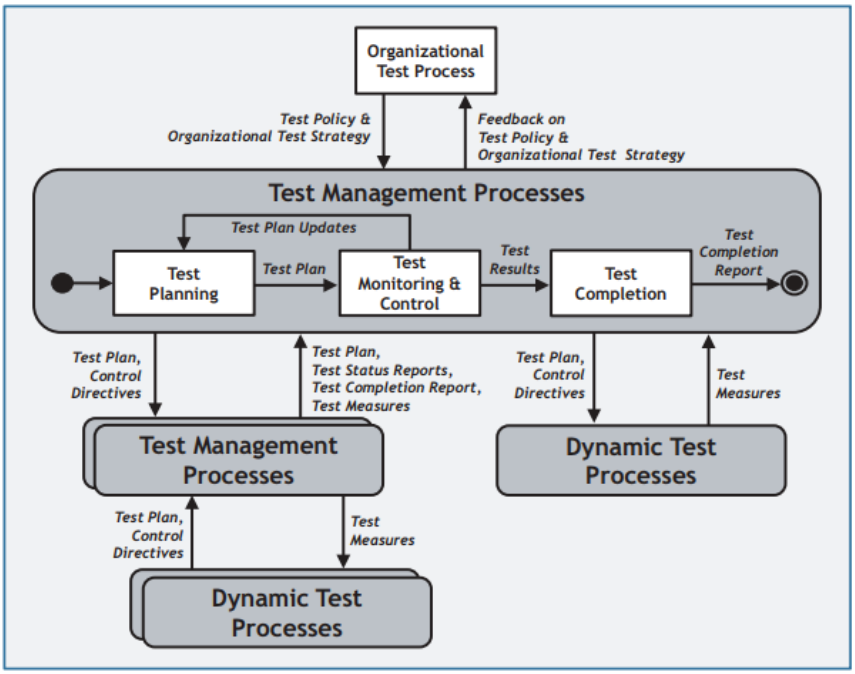
Requirements Analysis and Specification
Choose testing criteria
Obtain or build support software
Define testing plans at each level
Build test prototypes
Clarify requirement items and test criteria
Develop project test plan
System and Software Design
Validate design and interface
Design system tests
Develop coverage criteria
Design acceptance test plan
Design usability test
Intermediate Design
Specify systems test cases
Develop integration and unit test plans
Build or collect test support tools
Suggest ordering of class integration
Detailed Design
Create test cases
Build test specifications
Implementation
Create test case values
Conduct unit testing
Report problems properly
Integration
Perform integration testing
System Deployment
Perform system testing
Perform acceptance testing
Perform usability testing

ISO/IEC/IEEE 29119-2

Organizational Test
Develop organizational test specificaton
Monitor and control of organizational test specification
Update organizational test specification
Test Planning
Understand context
Organize test plan development
Identify and analyse risks
Identify risk mitigation approaches
Design test strategy
Determine staffing and scheduling
Record test plan
Gain consensus on the test plan
Make the test plan available and communicate
Test Monitoring and Control
Setup
Monitor
Control
Report
Test Completion
Archive test assets
Clean up test environment
Identify lessons learned
Report test completion
Test Design and Implementation
Feature sets identification
Derive test conditions
Derivetest coverage items
Derive test cases
Assemble test sets
Derive test procedures
Test Environment Setup and Maintenance
Establish a test environment
Maintain test environment
Execute test procedures
Compare test results
Record test execution plan
Test Reporting
Analyze test results
Create/update incident report

فرآیند آزمون موجود در بخش دوم استاندارد ISO/IEC/IEEE 29119 به همراه فرآیند آزمون بیان شده در کتاب مرجع درس، در بالا قرار گرفته‌اند. در ادامه، لیستی از موارد متفاوت میان این دو فرآیند را برمی‌شماریم:

- فرآیند آزمون در استاندارد ISO 29119 به صورت لایه‌ای طراحی شده است که در بالاترین لایه فرآیند آزمون در سطح سازمان واقع شده است. در لایه پایین‌تر، فرآیند آزمون در سطح مدیریتی و پروژه است و در پایین‌ترین لایه، با متدهای تست و ابزارها در ارتباط هستیم. در حالی که فرآیند مطرح شده داخل کتاب، لایه‌ای نیست.
- به واسطه لایه‌ای بودن فرآیند آزمون در استاندارد ISO 29119، لایه‌ها می‌توانند با یکدیگر در ارتباط باشند. مثلا، در لایه Test Management Process در تصویر زیر، مشاهده می‌کنیم که با لایه Dynamic Test Process هم به صورت تعاملی در ارتباط است. این مورد باعث افزایش انعطاف‌پذیری در جریان اجرای فرآیند آزمون می‌شود. با این حال، در فرآیند کتاب مرجع تعامل میان بخش‌های مختلف فرآیند آزمون دیده نمی‌شود؛ تنها ارتباط میان بخش‌ها این است که خروجی بخش قبل، به عنوان ورودی بخش بعدی در نظر گرفته می‌شود.



- با توجه به وجود لایه مخصوص سازمانی و وظایفی که در آن مشخص شده است، به نظر می‌رسد استاندارد ISO 29119 نسبت به فرآیند آزمون کتاب مرجع، توجه بیشتری به فرآیندهای سازمانی دارد. البته این مورد نیز به دلیل اینکه می‌بایست یک استاندارد فراگیر باشد، قابل توجیه است.