SmartUnit: Empirical Evaluations for Automated Unit Testing of Embedded Software in Industry

Chengyu Zhang, Yichen Yan, Hanru Zhou, Yinbo Yao, Ke Wu,
Ting Su, Weikai Miao, Geguang Pu

East China Normal University, China
Nanyang Technological University, Singapore
National Trusted Embedded Software Engineering Technology Research Center, China

2018.5.30

Outline

- Background
- Approach
- Implementation
- Evaluation
- Conclusion

Motivation

RTCA DO-178B/C

IEC 61508

ISO26262

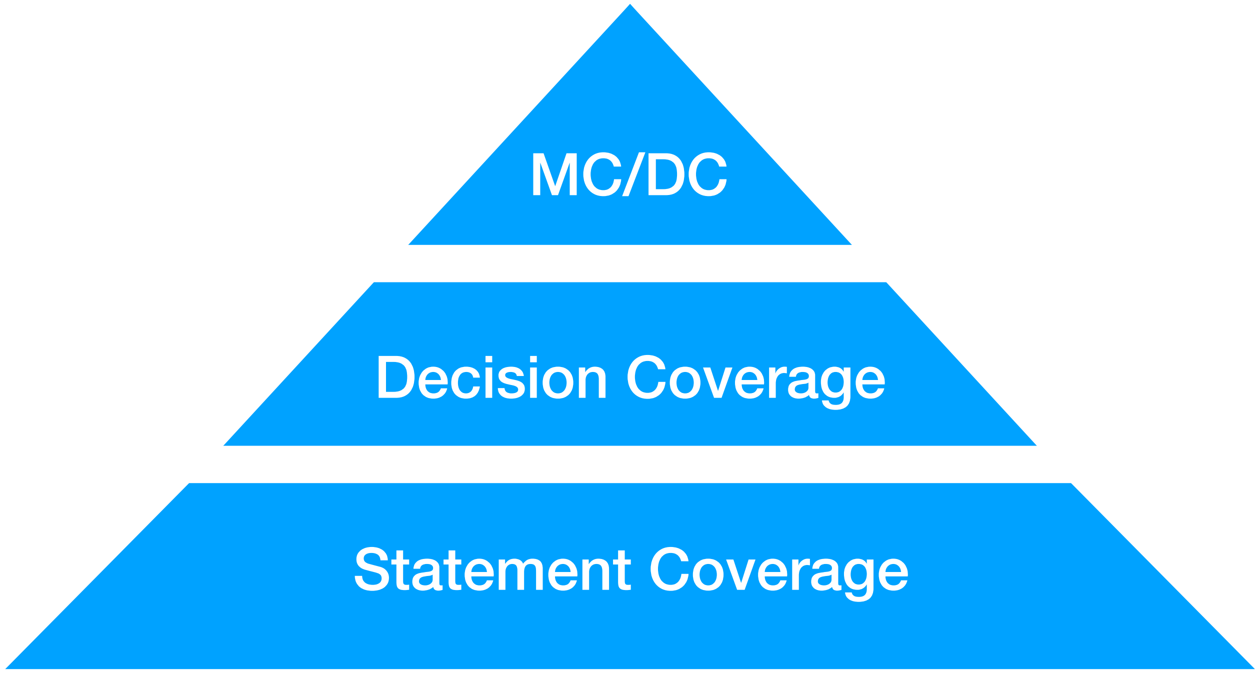
Unit Testing
Code Coverage Criterion

RTCA DO-178B/C

Level A

Level B

Level C



MC/DC

Decision Coverage

Statement Coverage

Condition Coverage

```
If ( A || B ) && C ) {
   /*Instructions*/
}
else{
   /*Instructions*/
}
```

```
A = True B = True C = True

A = False B = False C = False
```

Decision Coverage

```
If ( A || B ) && C ) {
   /*Instructions*/
}
else{
   /*Instructions*/
}
```

```
A = True B = True C = True
=> True

A = False B = False C = False
=> False
```

Modified Condition/Decision Coverage (MC/DC)

```
If ( A || B ) && C ) {
   /*Instructions*/
}
else{
   /*Instructions*/
}
```

```
A = False B = True C = True

A = False B = True C = False

A = False B = False C = True

A = True B = False C = True
```

Modified Condition/Decision Coverage (MC/DC)

```
If ( A || B ) && C ) {
   /*Instructions*/
}
else{
   /*Instructions*/
}
```

```
A = False B = True C = True

A = False B = True C = False

A = False B = False C = True
=> False
A = True B = False C = True
=> True
```

Modified Condition/Decision Coverage (MC/DC)

```
If ( A || B ) && C ) {
   /*Instructions*/
}
else{
   /*Instructions*/
}
```

```
A = False B = True C = True
=>True
A = False B = True C = False
=>False
A = False B = False C = True

A = True B = False C = True
```

Modified Condition/Decision Coverage (MC/DC)

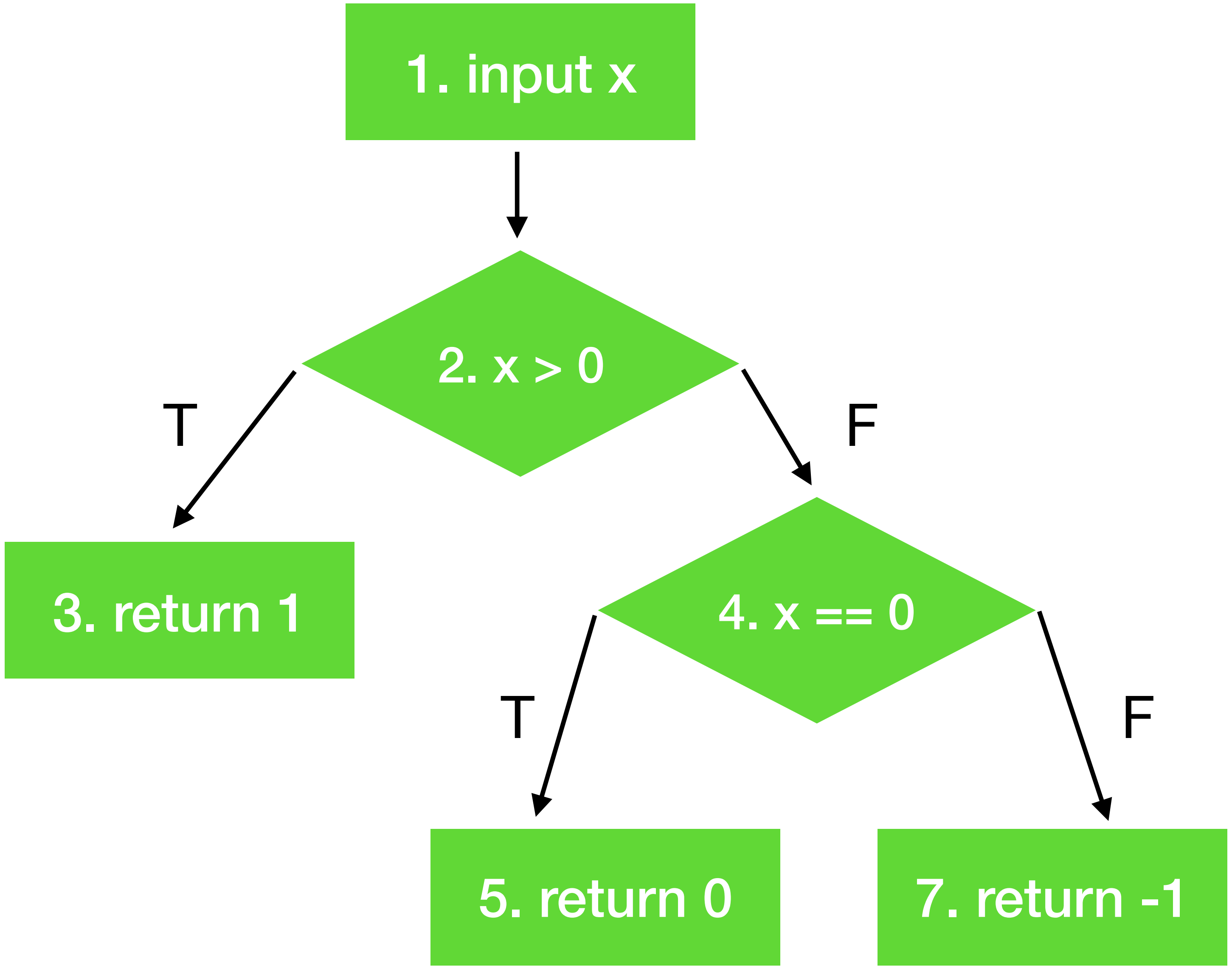
```
If ( A || B ) && C ) {
  /*Instructions*/
}
else{
  /*Instructions*/
}
```

```
A = False B = True C = True
=> True
A = False B = True C = False

A = False B = False C = True
=> False
A = True B = False C = True
```

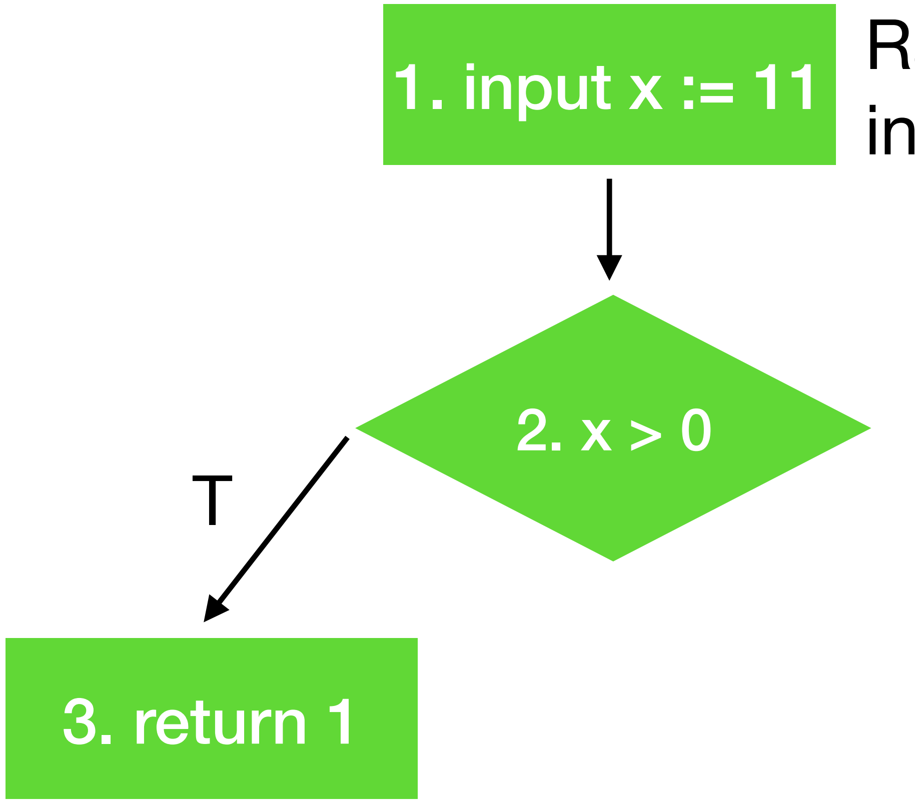
Dynamic Symbolic Execution (Concolic Testing)

```
int checkSign (int x){
  if (x > 0)
    return 1;
  else if (x == 0)
    return 0;
  else
    return -1;
}
```



Dynamic Symbolic Execution (Concolic Testing)

```
1 int checkSign (int x){
2   if (x > 0)
3     return 1;
4   else if (x == 0)
5     return 0;
6   else
7     return -1;
8 }
```

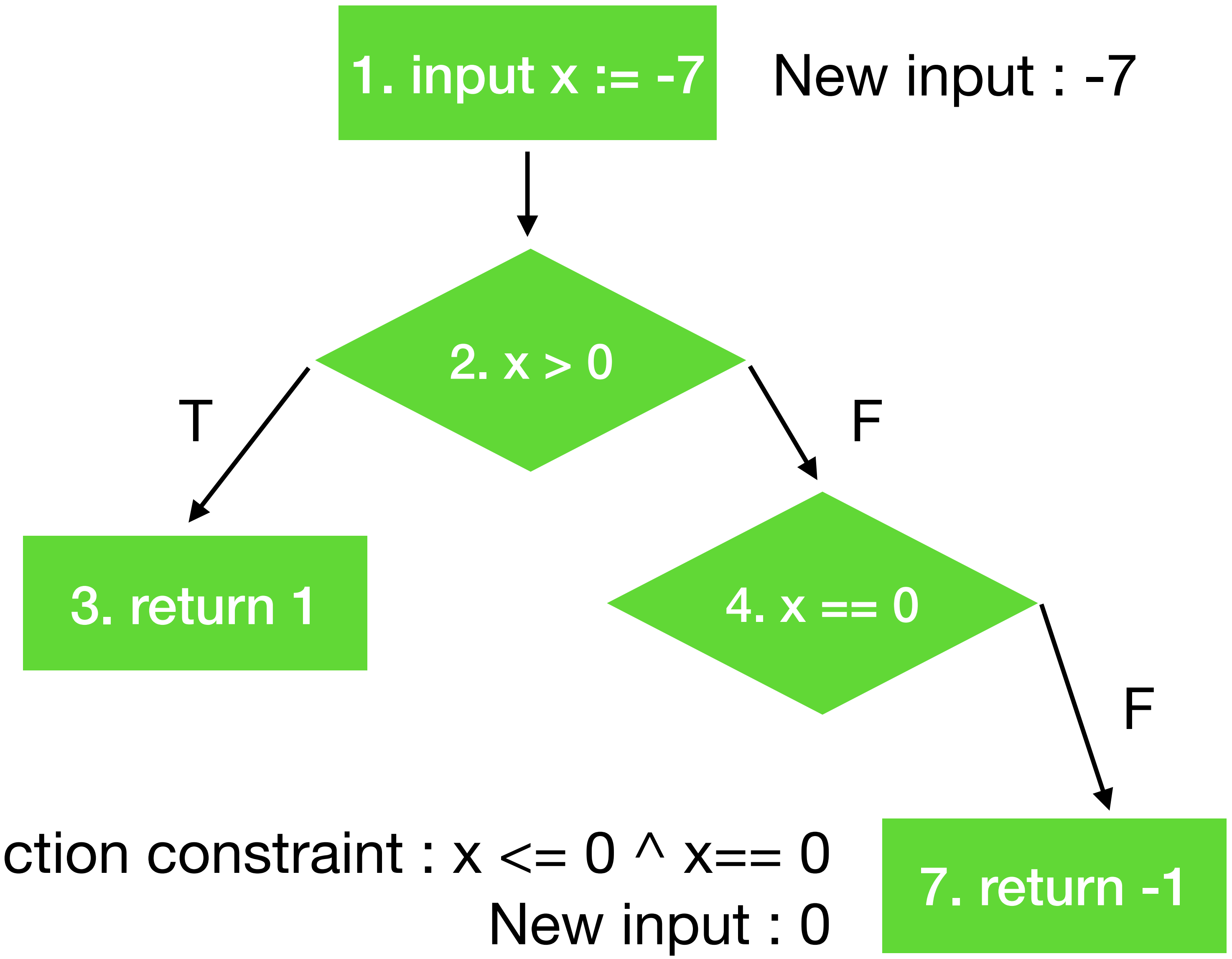


Random generated input : 11

Conjunction constraint : x <= 0
New input : -7

Dynamic Symbolic Execution (Concolic Testing)

```
1 int checkSign (int x){
2   if (x > 0)
3     return 1;
4   else if (x == 0)
5     return 0;
6   else
7     return -1;
8 }
```



1. input x := -7

New input : -7

2. x > 0

T

F

3. return 1

4. x == 0

F

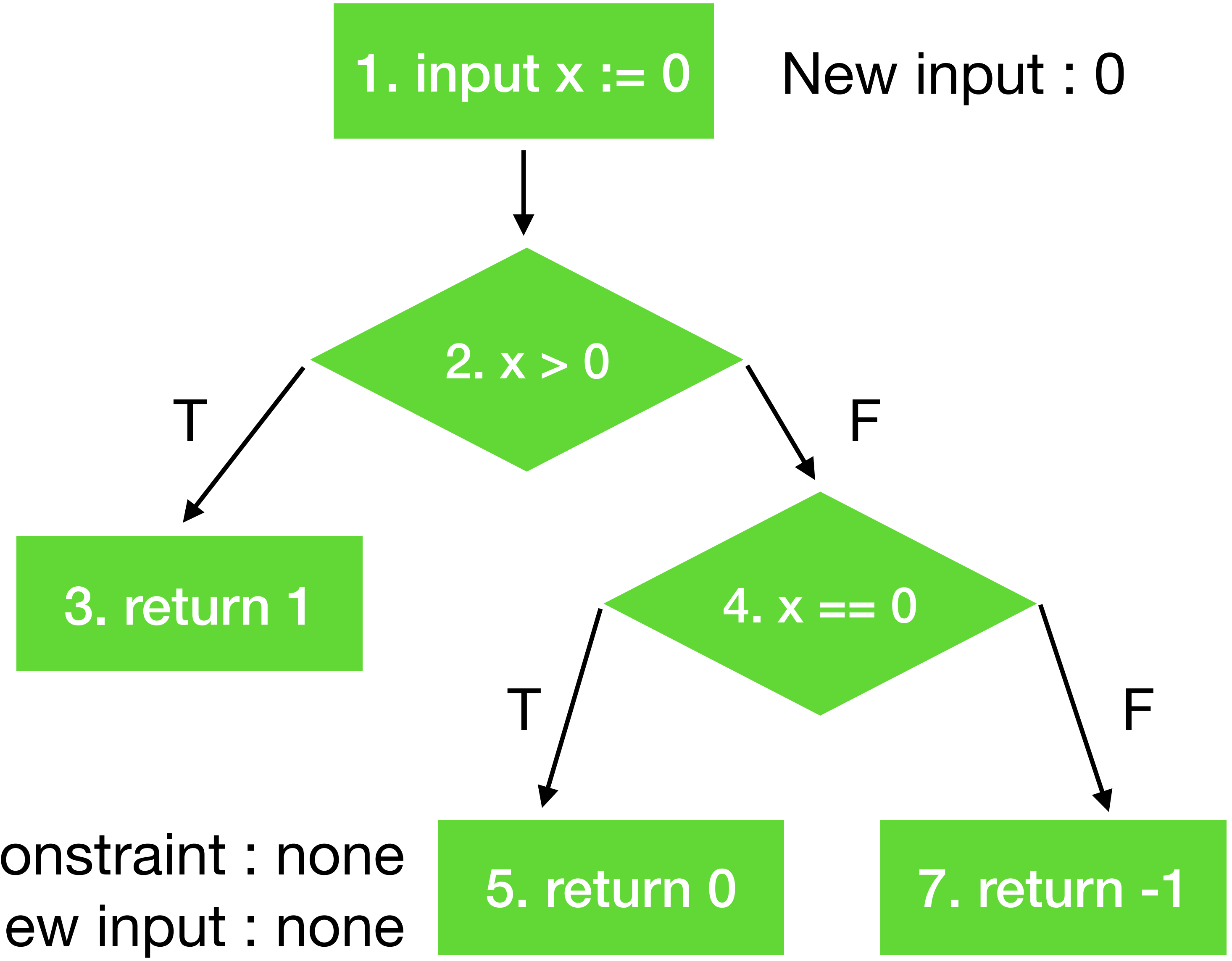
7. return -1

Conjunction constraint : x <= 0 ^ x== 0

New input : 0

Dynamic Symbolic Execution (Concolic Testing)

```
1 int checkSign (int x){
2   if (x > 0)
3     return 1;
4   else if (x == 0)
5     return 0;
6   else
7     return -1;
8 }
```



New input : 0

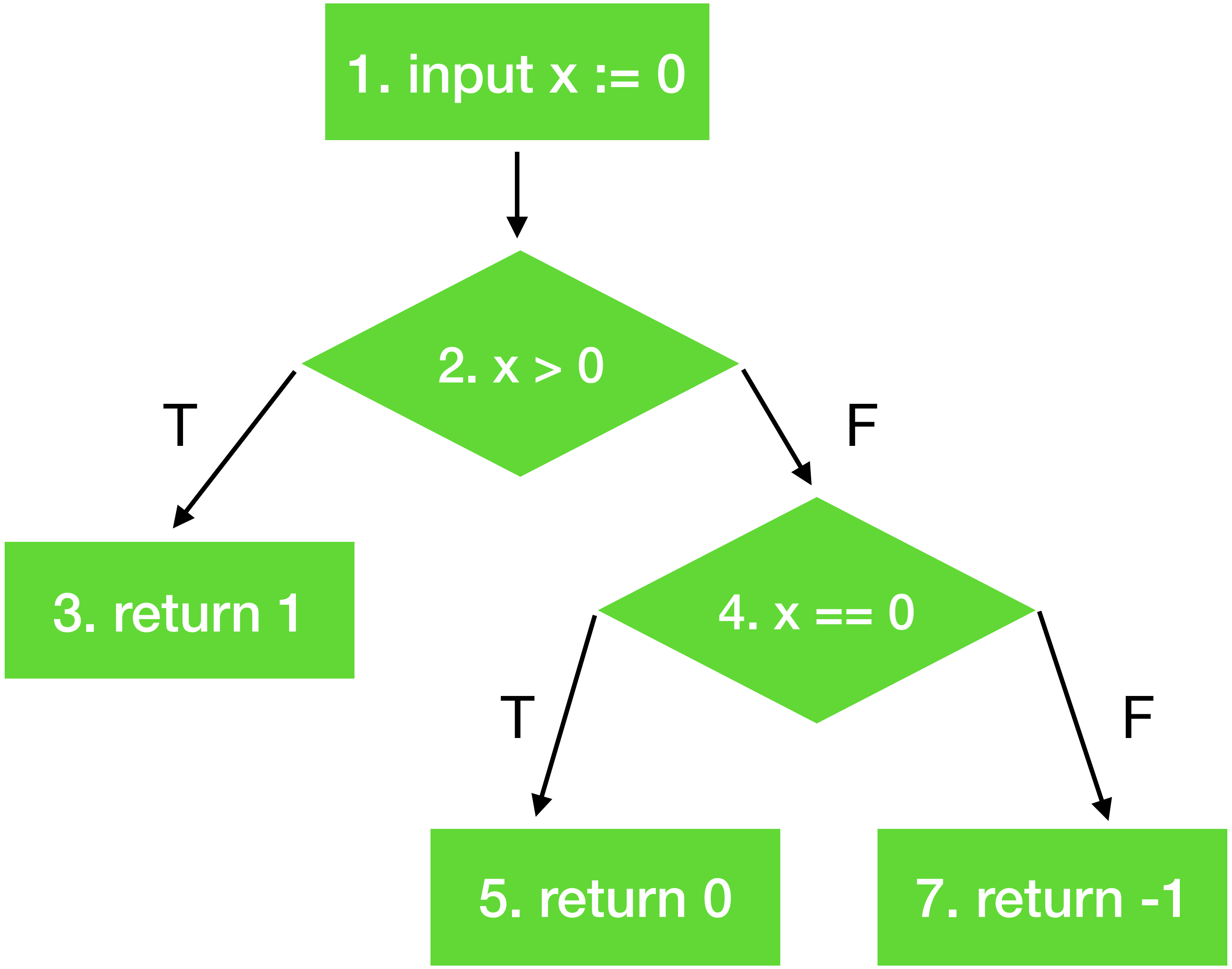
Conjunction constraint : none
New input : none

Dynamic Symbolic Execution (Concolic Testing)

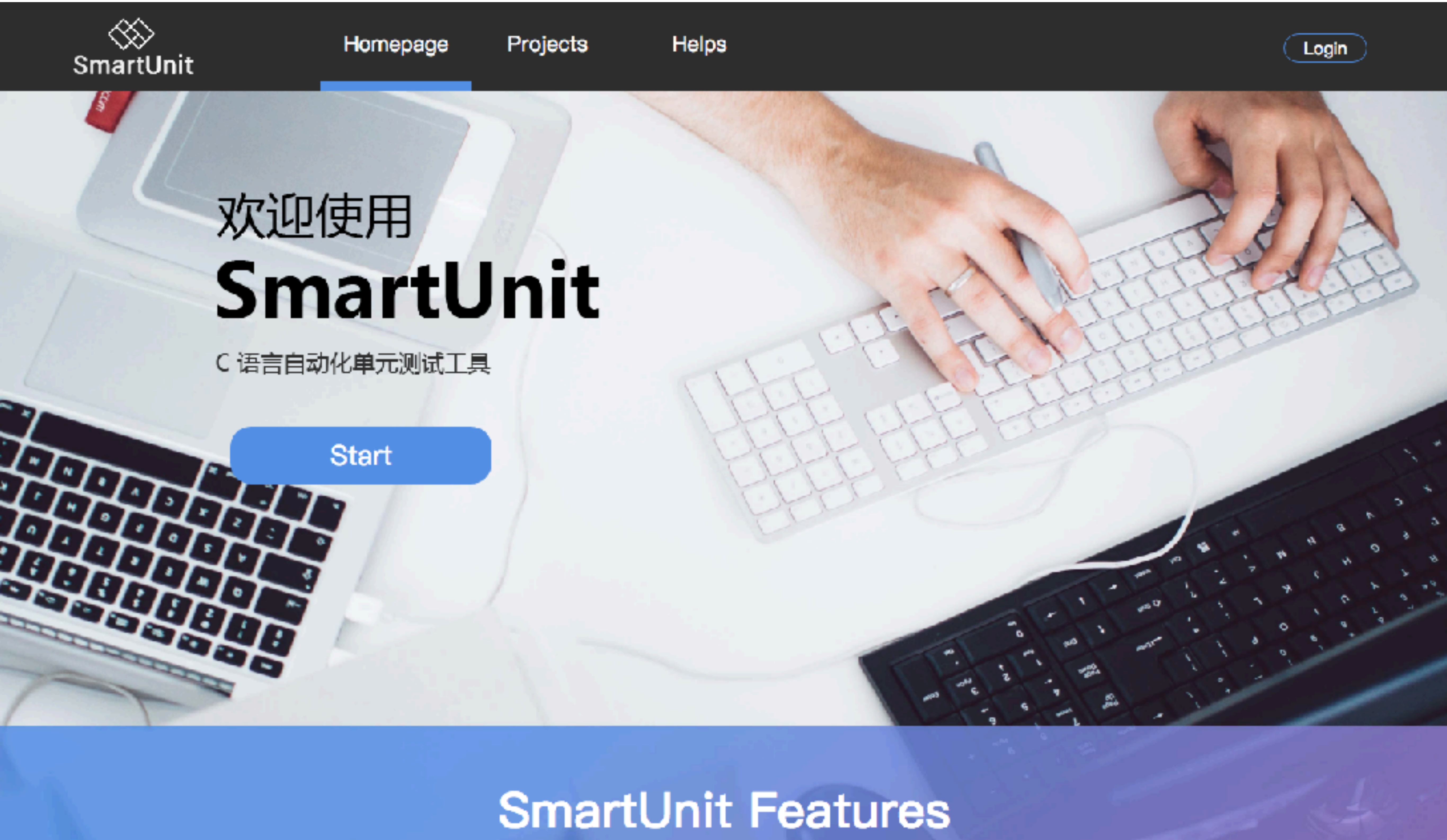
```
1 int checkSign (int x){
2   if (x > 0)
3     return 1;
4   else if (x == 0)
5     return 0;
6   else
7     return -1;
8 }
```

Test suite :

{ 11, -7, 0 }



Cloud-based Platform



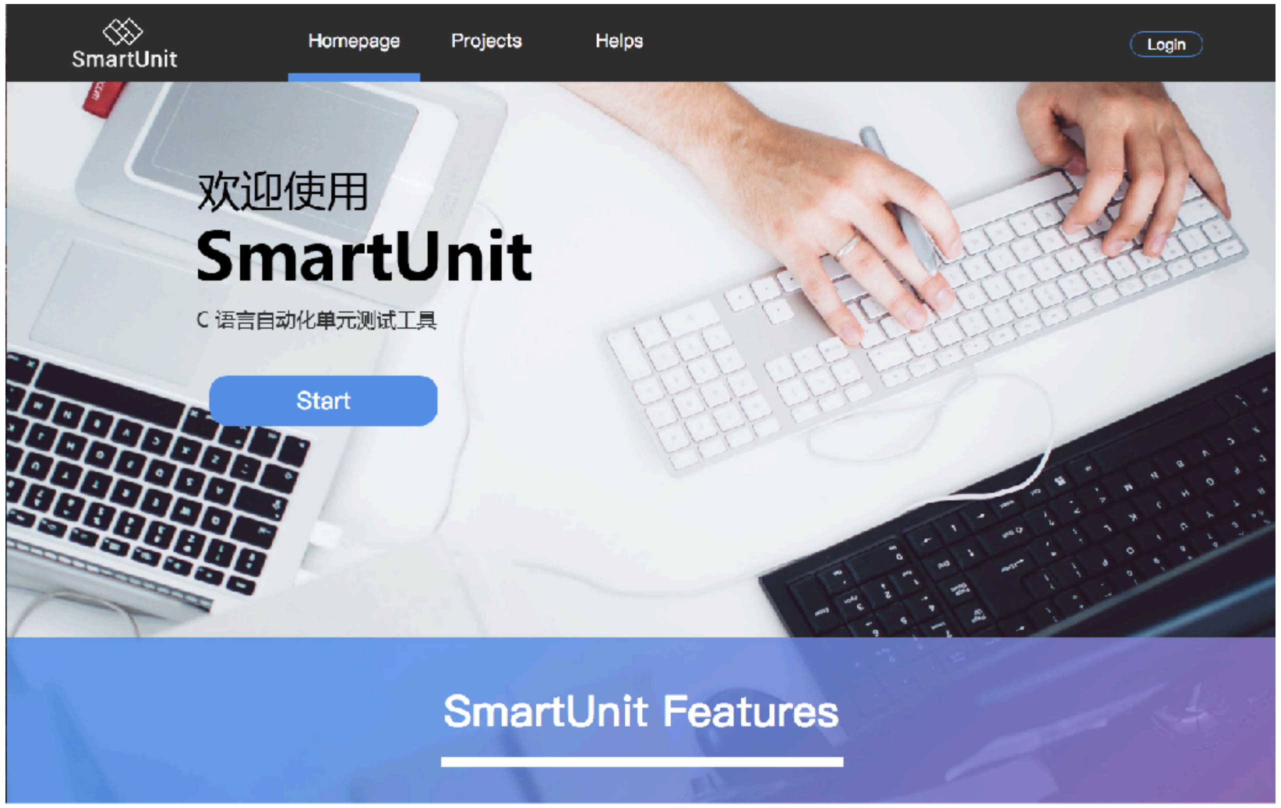
欢迎使用

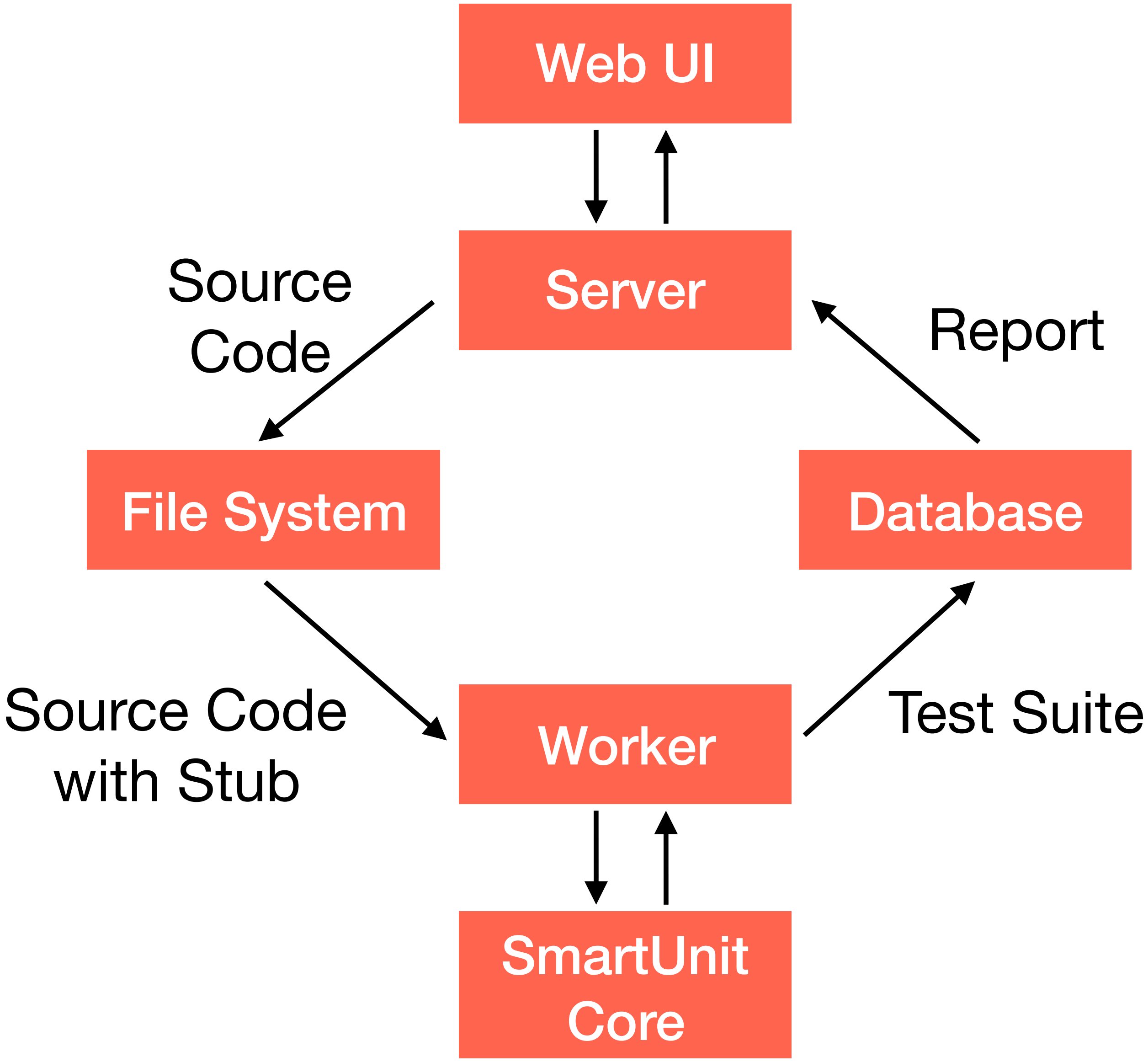
SmartUnit

C 语言自动化单元测试工具

SmartUnit Features

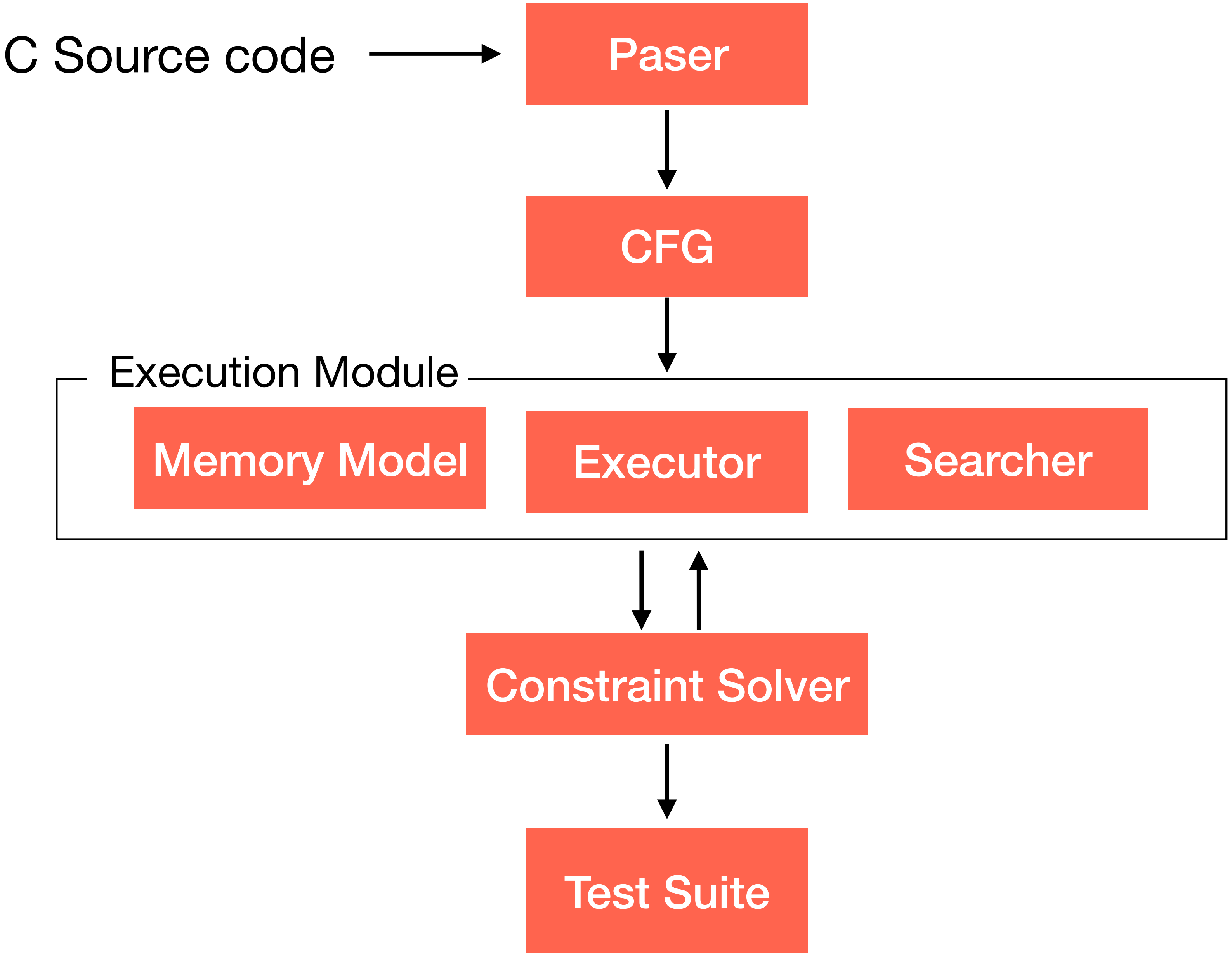
Cloud-based Platform Workflow





Implementation

SmartUnit Execution Core



Features

Automatically

insert stubs for function calls & variables
(Global variables, function inputs, etc.)

generate test suite.
(For LDRA Testbed, Tessy, etc.)

generate test report.
(Statement, branch, MC/DC coverage)

Research Questions

RQ1: How about the performance of SmartUnit on both commercial embedded software and open-source database software?

RQ2: What factors make dynamic symbolic execution get low coverage?

RQ3: Can SmartUnit find the potential runtime exceptions in real-world software?

RQ4: What are the differences in terms of time, cost and quality between automatically generated test cases and manually written test cases?

RQ1：

How about the performance of SmartUnit ?

Subjects	# Files	# Functions	# LOC
Aerospace Software	8	54	3,769
Automotive Software	4	330	31,760
Subway Signal Software	108	874	37,506
SQLite	2	2,046	126,691
PostgreSQL	906	6,105	279,809
Total	1,028	9,409	479,535

RQ1：

How about the performance of SmartUnit ?

Subjects	Statement Coverage*				Branch Coverage*				MC/DC Coverage*			
	N/A	0-50%	50-99%	100%	N/A	0-50%	50-99%	100%	N/A	0-50%	50-99%	100%
Aerospace Software	1	3	10	41	1	5	8	41	45	2	-	8
Automotive Software	1	3	11	315	1	6	8	315	274	5	1	50
Subway Signal Software	6	1	50	817	6	2	55	811	558	11	11	294
SQLite	86	86	206	1668	86	119	205	1636	1426	118	149	351
PostgreSQL	687	732	1044	3642	687	1102	804	3512	4083	1308	249	465

*** : Statistic with the number of functions for corresponding coverage range**

RQ2：

What factors make dynamic symbolic execution get low coverage ?

- Environment variables and Environment functions
- Complex operations
- Limitation of SMT solver

RQ3：

Can SmartUnit find the potential runtime exceptions in real-world software?

- Array index out of bounds
- Fixed memory address
- Divided by zero

RQ3:

Can SmartUnit find the potential runtime exceptions in real-world software?

- Array index out of bounds

```
static char *cmdline_option_value(int argc, char **argv, int i) {
  if (i == argc) {
    utf8_printf(stderr, "Error: missing argument to %s\n",
                 argv[0], argv[argc - 1]);
    exit(1);
  }
  return argv[i];
}
```

RQ3：

Can SmartUnit find the potential runtime exceptions in real-world software?

- Fixed memory address

(*0X00000052) or (*(symbolic_variable+12)

RQ3:

Can SmartUnit find the potential runtime exceptions in real-world software?

- Divided by zero

```
static void getLocalPayload(int nUsable, u8 flags, int nTotal, int *pnLocal){
  int nLocal, nMinLocal, nMaxLocal;
  if( flags==0x0D ){
    nMinLocal = (nUsable - 12) * 32 / 255 - 23;
    nMaxLocal = nUsable - 35;
  }else{
    nMinLocal = (nUsable - 12) * 32 / 255 - 23;
    nMaxLocal = (nUsable - 12) * 64 / 255 - 23;
  }
  nLocal = nMinLocal + (nTotal - nMinLocal) % (nUsable - 4);
}
```

RQ4：

What are the differences between automatically generated and manually written test cases?

Subjects	# Functions	Time (s)	Average (s/func)
Aerospace Software	54	318	6
Automotive Software	330	329	1
Subway Signal Software	874	2,476	3
SQLite	2,046	13,482	6
PostgreSQL	6,105	18,857	3
Total	**9,409**	**35,462**	**3.77**

A trained test engineer can only product test case for 5-8 functions per day.

Conclusion

SmartUnit

- Dynamic symbolic execution
  (High coverage unit testing)
- Potential runtime exceptions
  (Out of bounds, divided by zero, etc.)
- Industry application
  (Insert stubs, test report, test suite)

CASC China Academy of Space Technology

卡斯柯 CASCO

GAC MOTOR