Automatic Specification-Based Program Testing

Shaoying Liu
Department of Computer Science
Faculty of Computer and Information Sciences
Hosei University, Tokyo, Japan
Email: sliu@hosei.ac.jp
HP: http://cis.k.hosei.ac.jp/~sliu/

Overview

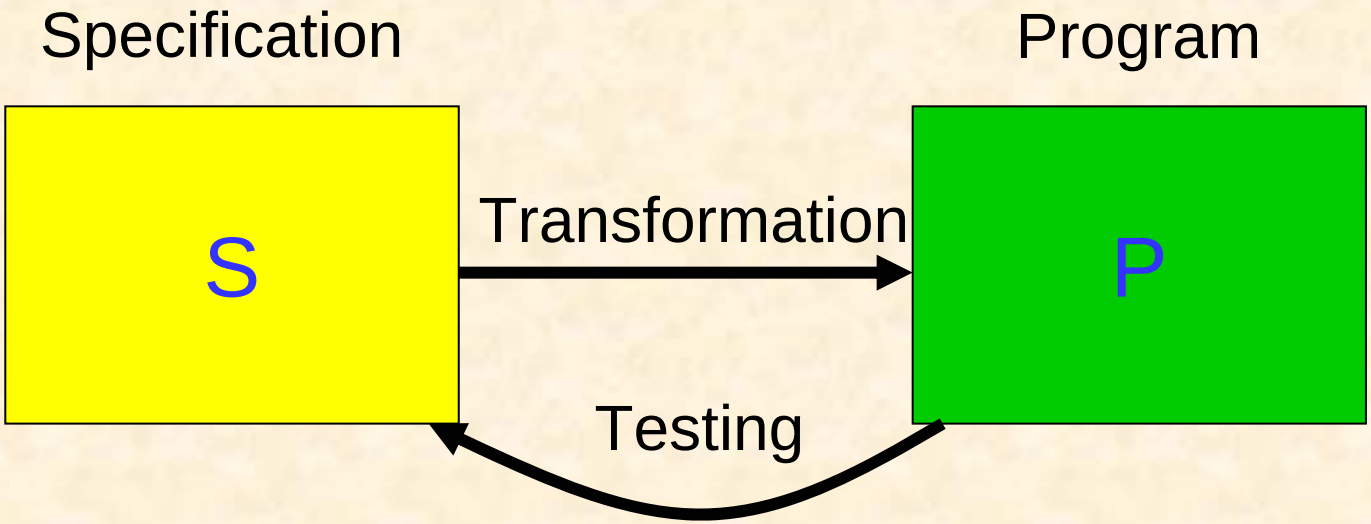
1. **Goals of Automatic Specification-Based Testing**

2. **A decompositional approach to automatic test case generation based on formal specifications**

3. **Test oracle for test result analysis**

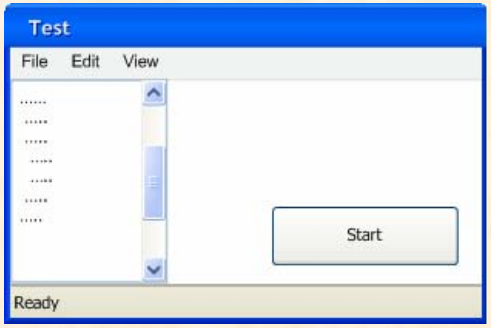
4. **Conclusion and future research**

1. Goals of Automatic Specification-Based Testing



Ideal goal of automatic specification-based testing (ASBT)

Press a Button



Adequate test cases

	x	y	z
case1	3	5	2
case2	0	4	9
case3	9	3	35
……			
……			

```
Method(int x, int y, int z){
int w;
if(x < y)
{
  w = y/x;
  while(w < z)
  {
    …
  }
} else
  …
}
}
```

Next

Practical goals of ASBT

1. Every independent function defined in the specification is tested (at least once) (User's view).

2. Every representative program path is traversed or some required coverage criteria (e.g., MCDC - Modified Condition/Decision Coverage) are satisfied. (Program's view)

2. A decompositional approach to automatic test case generation based on formal specifications

(1) Strategy and criteria for test case generation

(2) Algorithms for test set generation

(3) “Vibration” method (V-Method) for test set generation from atomic predicates

(1) Strategy and Criteria for Test Case Generation

Let

denote an operation specification. A set of functional scenarios can be derived from the specification, each defining an independent function in terms of input-output relation.

Scenario-based testing: a strategy for ``divide and conquer''

Specification

```
process A(x: int) y: int
pre   x > 0
post (x > 10 => y = x + 1) and
       (x <= 10 => y = x – 1)
end_process
```

Program

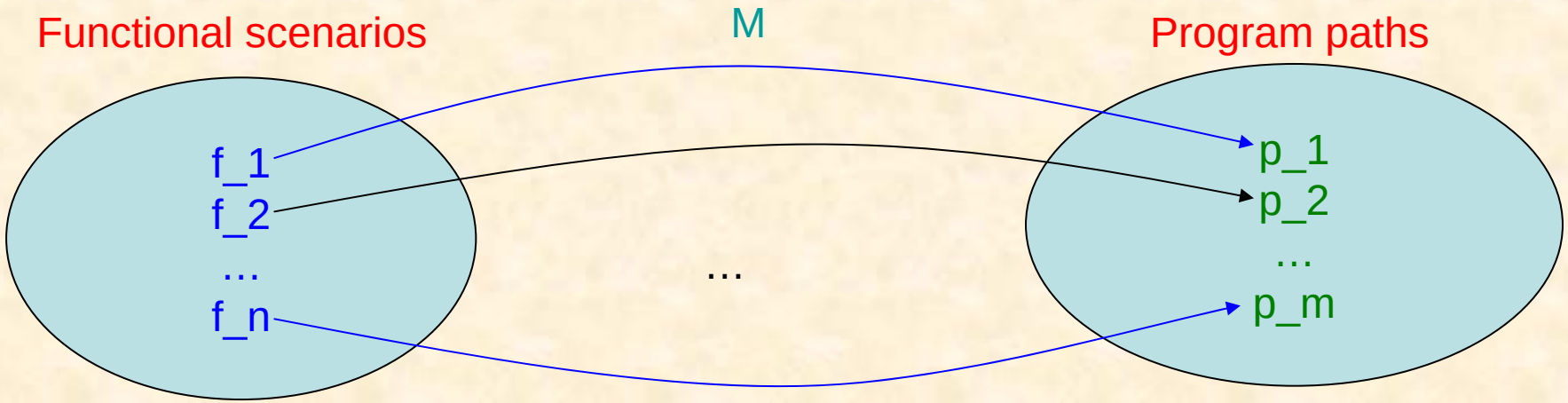
```
int A(int x) {
If (x > 0) {
  if (x > 10)  y := x * 1;
  else  y := x – 1;
  return y; }
else System.out.println("the
       pre is violated")  }
```

Satisfy?

Functional scenario:

~Apre ∧Ci ∧Di

(i=1,…,n)



Definition 1.1 (FSF) Let

$S_{post} \equiv (C_1 \wedge D_1) \vee (C_2 \wedge D_2) \vee \cdots \vee (C_n \wedge D_n)$,

where C_i is a guard condition and

D_i is a defining condition, $i = 1,\ldots,n$.

Then, a functional scenario form (FSF) of S is:

$(\sim S_{pre} \wedge C_1 \wedge D_1) \vee (\sim S_{pre} \wedge C_2 \wedge D_2) \vee \cdots \vee$

$(\sim S_{pre} \wedge C_n \wedge D_n)$

where

$f_i = \sim S_{pre} \wedge C_i \wedge D_i$ is called a functional scenario) and

$\sim S_{pre} \wedge C_i$ is called a test condition.

Example:

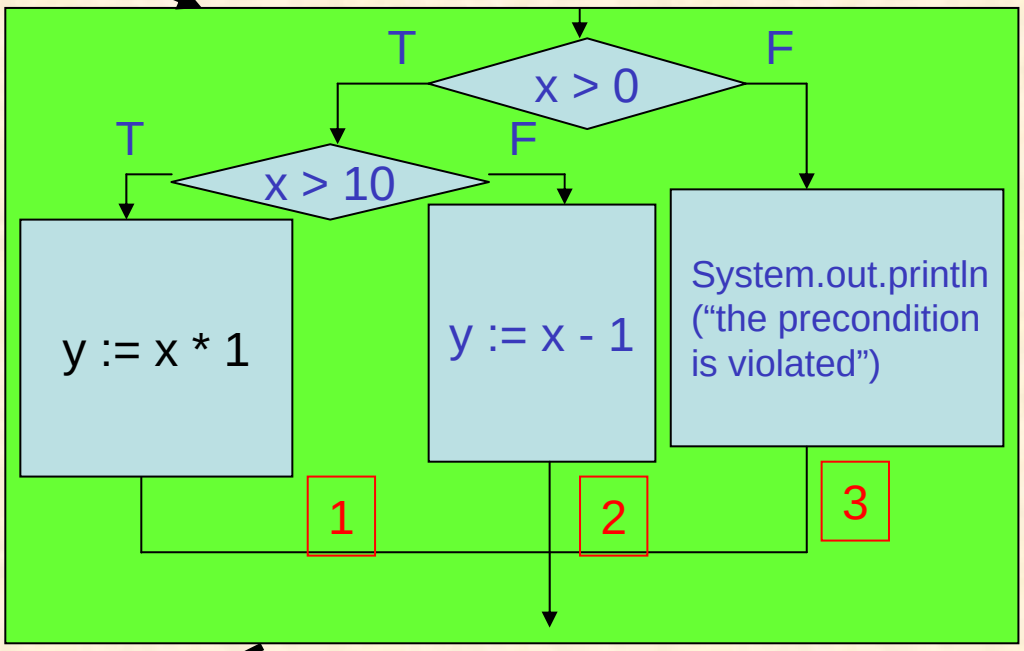
Test case generation

Specification

process A(x: int) y: int
pre x > 0
post (x > 10 => y = x + 1) and
(x <= 10 => y = x – 1)

Functional scenarios:
(1) x > 0 ∧ x > 10 ∧ y = x + 1
(2) x > 0 ∧ x <= 10 ∧ y = x – 1

Program



Test result analysis

Test strategy:

Let operation S have an FSF $(\sim S_{pre} \wedge C_1 \wedge D_1) \vee (\sim S_{pre} \wedge C_2 \wedge D_2) \vee \cdots \vee (\sim S_{pre} \wedge C_n \wedge D_n)$, where $(n \geq 1)$.

Let T be a test set for S. Then, T must satisfy the condition

$$(\forall i \in \{1,...,n\} \exists t \in T \cdot \sim S_{pre}(t) \wedge C_i(t))) \text{ and } \exists t \in T \cdot \neg \sim S_{pre}(t)$$

where $\neg \sim Spre(t)$ describes an exceptional situation.

Notation: $G: L_E \rightarrow T_S$

where

L_E is the set of logical expressions involved.

T_S is a set of test sets

Criterion 1:

$$G((\sim Spre \wedge C_1 \wedge D_1) \vee (\sim Spre \wedge C_2 \wedge D_2) \vee \cdots \vee (\sim Spre \wedge Cn \wedge Dn)) =$$

$$G(\sim Spre \wedge C_1 \wedge D_1) \, U \, G(\sim Spre \wedge C_2 \wedge D_2) \, U \cdots U \, G(\sim Spre \wedge Cn \wedge Dn).$$

Criterion 2:

Let $\sim S_{pre} \wedge C_i \wedge D_i$ $(i = 1,...,n)$ be a functional scenario of specification S. Then,

$G(\sim S_{pre} \wedge C_i \wedge D_i) = G(\sim S_{pre} \wedge C_i)$

Criterion 3:

Let $P_1 \vee P_2 \vee \cdots \vee P_m$ be a DNF of the test condition $\sim S_{pre} \wedge C_i$. Then, we define

$G(\sim S_{pre} \wedge C_i) =$

$G(P_1 \vee P_2 \vee \cdots \vee P_m) =$

$G(P_1) \cup G(P_2) \cup \cdots \cup G(P_m)$

Criterion 4:

Let $S_{iv} = \{x_1, x_2, \ldots, x_r\}$ and $Q(x_1, x_2, \ldots, x_q)$ $(q \leq r)$ be a relation involving variables $x_1, x_2, \ldots, x_q$.

Then,

$$G(Q(x_1, x_2, \ldots, x_q)) = \{T_c \mid (\forall x \in \{x_1, x_2, \ldots, x_q\} \cdot Q(T_c(x_1), T_c(x_2), \ldots, T_c(x_q))) \wedge (\forall x \in (S_{iv} \setminus \{x_1, x_2, \ldots, x_q\}) \cdot T_c(x) = any)\}$$

where T_c: S_{iv} -> Values

Example: $G(x > y) = \{ \{(x, 5), (y, 3), (z, 8)\}, \{(x, 8), (y, 2), (z, 300)\} \}$

Criterion 5:

Let $Q_1 \wedge Q_2 \wedge \cdots \wedge Q_w$ be a conjunction of w atomic predicates in the test condition of a functional scenario of S. Then, we have

$$G(Q_1 \wedge Q_2 \wedge \cdots \wedge Q_w) =$$
$$G(Q_1) \cap G(Q_2) \cap \ldots \cap G(Q_w)$$

(2) Algorithms for Test Set Generation

(2.1) For Atomic Predicates

Let $Q(x_1, x_2, ..., xm)$ be an atomic predicate. It may have three formats:

(1) $x_1 \ominus E$, where $\ominus \in \{=, >, <, >=, <=, <>\}$, x_1 is a single variable, E a constant.

(2) $E_1 \ominus E_2$, where E_1 and E_2 are both arithmetic expressions that involve only variable x_1.

(3) $E_1 \ominus E_2$, where E_1 and E_2 are both arithmetic expressions that may involve variables $x_1, x_2, ..., xm$.

An algorithm of test case generation for $x_1,x_2,\ldots,x_m$ in format (3) $E_1 \ominus E_2$:

Step1 Randomly choose values $v_2,v_3,\ldots,v_m$ from the corresponding types of variables in $E_1 \ominus E_2$;

Step2 Substitute $v_2,v_3,\ldots,v_m$ for variables $x_2,x_3,\ldots,x_m$ in $E_1 \ominus E_2$;

Step3 Convert $E_1 \ominus E_2$ to the format $x_1 \ominus E$ by applying appropriate algorithms depending on $\ominus$.

(2.2) For conjunctions

Let $Q_1 \wedge Q_2 \wedge \cdots \wedge Q_w$ be a conjunction of atomic predicates.

Let $x_1, x_2, \ldots, x_r$ be all input variables of the operation specification. Then, an algorithm for generating a test case from the conjunction is:

First apply the corresponding algorithm to generate a test case for Q_1, and then use it to evaluate Q_2, …, Q_w. If all of them are true, a test case is generated; otherwise, repeat the same procedure until a test case is generated or a stopping generation condition is met.

(2.3) For disjunctions

A simple algorithm for test case generation from the disjunction $P_1 \vee P_2 \vee \cdots \vee P_m$:

The essential part of the algorithm is a while-loop, which produces one test case from each disjunctive clause until all the disjunctive clauses are covered, and then form a test set that contains all of the test cases produced.

(3) “Vibration” method (V-Method) for test set generation from atomic predicates

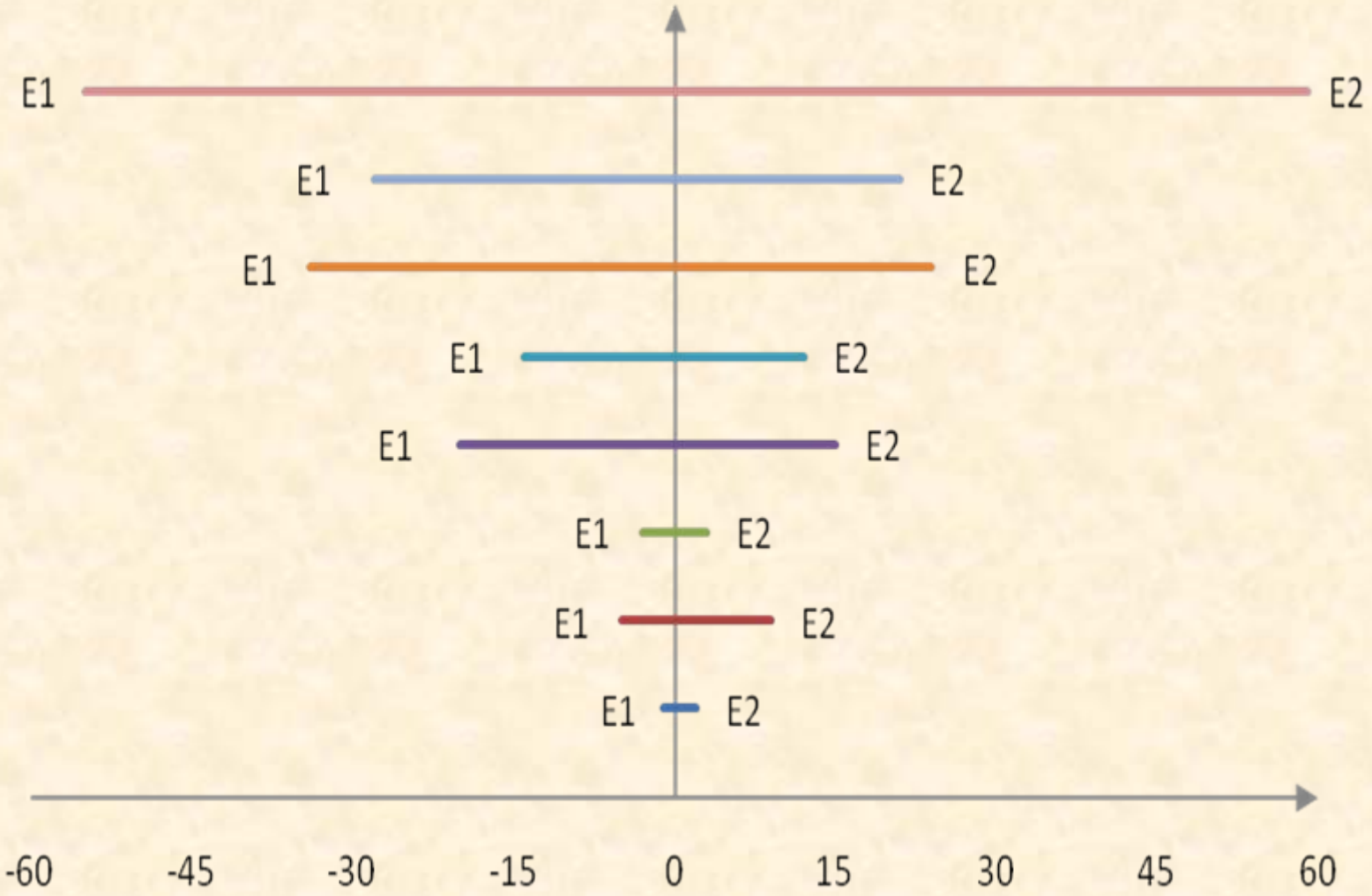
Let $E_1(x_1,x_2,\ldots,x_n)$ R $E_2(x_1,x_2,\ldots,x_n)$ denote that expressions E_1 and E_2 have relation R, where $x_1,x_2,\ldots,x_n$ are all input variables involved in these expressions.

Question: If the test case generated from this relation is not adequate to meet the required coverage criterion or standard, how to generate other test cases so that the goal of satisfying the required criterion can be quickly approached?

V-Method:

We first produce values for $x_1,x_2,\ldots,x_n$ such that the relation $E_1(x_1,x_2,\ldots,x_n)\ R\ E_2(x_1,x_2,\ldots,x_n)$ holds with any "distance" between E_1 and E_2, and then repeatedly create more values for the variables such that the relation still holds but the “distance” between E_1 and E_2 “vibrates” (changes repeatedly) between the initial “distance” and the “maximum” “distance”.

Example: E1 > E2



Examples of Distance definition

Distance function definition for nat0, nat, int, real types:

Distance(E_1, E_2, “R") ≡ abs(E_1 - E_2)

where R inset {>, >=, <, <=, <>, =}

Distance function definition for set types:

$Distance(E_1, E_2, \text{"subset"}) \equiv card(E_2) - card(E_1)$

$Distance(E_1, E_2, \text{"psubset"}) \equiv card(E_2) - card(E_1)$

$Distance(E_1, E_2, \text{"inset"}) \equiv card(E_2) - index(E_1,E_2)$

$Distance(E_1, E_2, \text{"notin"}) \equiv card(E_2)$

$Distance(E_1, E_2, \text{"="}) \equiv 0$

$Distance(E_1, E_2, \text{"<>"}) \equiv abs(card(E_1)-card(E_2))$

3. Test oracle for test result analysis

Definition 3.1: Let $\sim S_{pre} \wedge C \wedge D$ be a functional scenario and **T** be a test set generated from its test condition $\sim S_{pre} \wedge C$. If the condition

$$\exists t \in T \cdot \sim S_{pre}(t) \wedge C(t) \wedge \neg D(t, P(t))$$

holds, it indicates that a bug in program P is found by t (also by T).

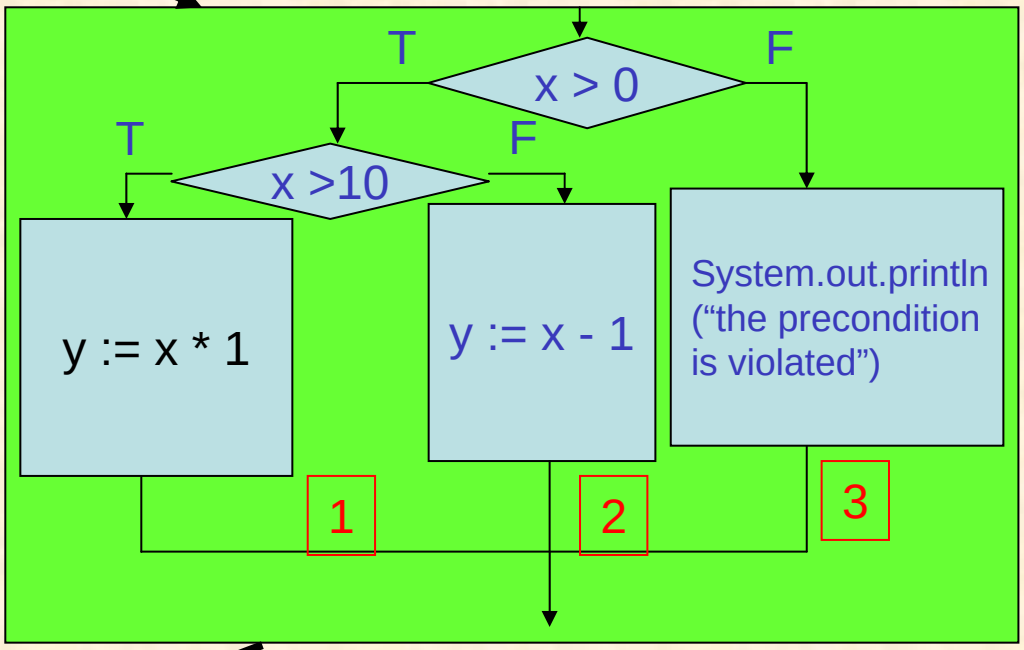
Test case generation

Specification

process A(x: int) y: int
pre x > 0
post (x > 10 => y = x + 1) and
(x <= 10 => y = x – 1)

Functional scenarios:
(1) x > 0 ∧ x > 10 ∧ y = x + 1
(2) x > 0 ∧ x <= 10 ∧ y = x – 1

Program



Test result analysis

x	y	~Apre ∧ C1 (C2)	D1 (D2)	~Apre ∧ C1 ∧ ¬ D1 (¬D2)
15	15	true	false	true
8	7	true	true	false

4. Conclusion and future research

4.1 Conclusion

Automatic specification-based testing can significantly benefit from formal specifications, but still face many challenges.

Test result analysis based on formal specifications can be automatically performed, but it would be extremely difficult for informal specification-based testing.

4.2 Future research

- Establish a theory to explain the relation among specification, test case generation method, coverage criteria, and bug detection effectiveness.
- Develop more efficient algorithms for test case generation from a conjunction.
- Explore automatic debugging techniques to deal with realistic software systems.

Selection of our recent related publications

(1) **Shaoying Liu**, Yuting Chen, Fumiko Nagoya, John McDermid, "Formal Specification-Based Inspection for Verification of Programs", **IEEE Transactions on Software Engineering**, 30 Sept., 2011.

(2) **Shaoying Liu**, John McDermid, Yuting Chen, "A Rigorous Method for Inspection of Model-Based Formal Specifications", **IEEE Transactions on Reliability**, Vol. 59, No. 4, December, 2010, pp. 667-684.

(3) **Shaoying Liu**, Tetsuo Tamai, Shin Nakajima, "A Framework for Integrating Formal Specification, Review, and Testing to Enhance Software Reliability", **International Journal of Software Engineering and Knowledge Engineering**, 21(2), 2011, pp. 259-288.

(4) **Shaoying Liu**, ``Integrating Top-Down and Scenario-Based Methods for Constructing Software Specifications", **Journal of Information and Software Technology**, Elsevier, Vol. 54, No. 11, Nov. 2009, pp. 1565-1572.

(5) **Shaoying Liu** and Hao Wang ``An Automated Approach to Specification Animation for Validation", **Journal of Systems and Software**, Elsevier Science Inc., No. 80, 2007, pp. 1271-1285.

(6) **Shaoying Liu** and Yuting Chen, ``A Relation-Based Method Combining Functional and Structural Testing for Test Case Generation", **Journal of Systems and Software**, Elsevier Science Inc.,Vol. 81, No. 2, February 2008, pp. 234-248.

(8) **Shaoying Liu**, Shin Nakajima, "A Decompositional Approach to Automatic Test Case Generation Based on Formal Specifications", 4th IEEE International Conference on Secure Software Integration and Reliability Improvement (SSIRI 2010), Singapore, June 9-11, 2010, pp. 147-155.

(9) **Shaoying Liu**, T.Hayashi, K. Takahashi, K. Kimura, T. Nakayama, Shin Nakajima, "Automatic Transformation from Formal Specifications to Functional Scenario Forms for Automatic Test Case Generation", 9th International Conference on Software Methodologies, Tools, and Techniques (SoMet 2010), IOS Press, Yokohama, Japan, Sept. 29-Oct.1, 2010, pp. 383-397.

(10) Xi Wang, **Shaoying Liu**, Huaikou Miao, "A Pattern System to Support Refining Informal Ideas into Formal Expressions", 12th International Conference on Formal Engineering Methods (ICFEM 2010), LNCS, Springer-Verlag, 17-19 Nov. 2010, Shanghai, China, pp. 662-677

(11) Fauziah Zainuddin and **Shaoying Liu**, "Integrating Prototyping into the SOFL Three-Step Modeling Approach", **13th International Conference on Formal Engineering Methods (ICFEM 2011),** LNCS, Springer-Verlag, Durham, UK, Oct. 25-28, 2011, pp. 163-178.

(12) **Shaoying Liu** and Shin Nakajima, "A "Vibration" Method for Automatically Generating Test Cases Based on Formal Specifications", **18th Asia Pacific Software Engineering Conference (APSEC 2011),** IEEE CS Press, HCM City, Vietnam, Dec. 5-8, 2011, pp. 73-80.

Thank You !

課題

(1) Briefly describe what is the ideal goal of automatic specification-based testing.

(2) Briefly explain what is specification-based program testing and what activities are involved for such a testing.

(3) Describe the strategy for formal specification-based program testing?