Sumar
Execuție normală. Execuție simbolică
Execuția simbolică cu ramificații
Arborele execuției simbolice
Aplicabilitatea execuției simbolice
Bibliografie

Verificarea și Validarea Sistemelor Soft

Curs 05. Execuție simbolică

Lector dr. Camelia Chisăliță-Crețu

Universitatea Babeș-Bolyai
Cluj-Napoca

24 Martie 2020

1 Execuție normală. Execuție simbolică
- Execuție normală
- Execuție simbolică
- Comutativitatea CE – SE
- Stare simbolică

2 Execuția simbolică cu ramificații
- Execuția simbolică a structurii alternative
- Execuția simbolică a structurii repetitive

3 Arborele execuției simbolice
- Arborele execuției simbolice
- Arbori ai execuției simbolice. Exemple.
- Proprietăți ale arborelui de execuție simbolică

4 Aplicabilitatea execuției simbolice
- Corectitudinea programelor
- Istoric al execuției simbolice

5 Bibliografie

Execuție normală. Definiție.

- **execuție normală** (*engl.* conventional execution, **CE**)
  - execuție convențională/obișnuită în care datele de intrare sunt valori concrete, particulare;
- **funcția sumă**
  - 1: `int suma(int a, int b, int c){`
  - 2: `  int x := a + b;`
  - 3: `  int y := b + c;`
  - 4: `  int z := x + y - b;`
  - 5: `  return z;`
  - 6: `}`
- CE cu datele de intrare a=1, b=3, c=5
  $\Rightarrow$ apelul `suma(1,3,5)`.

Execuție normală (cont.)

- funcția sumă;
- CE pentru apelul `suma(1,3,5)`:
  - 1: `int suma(int a, int b, int c){`
  - 2: `int x := a + b;`
  - 3: `int y := b + c;`
  - 4: `int z := x + y - b;`
  - 5: `return z;`
  - 6: `}`

	a	b	c	x	y	z
1	1	3	5	-	-	-
2	1	3	5	4	-	-
3	1	3	5	4	8	-
4	1	3	5	4	8	9
5	1	3	5	4	8	9

- rezultatul apelului `suma(1,3,5)` este 9.

Execuție simbolică. Definiție

- **valoare simbolică**
  - un substitut (înlocuitor) furnizat programului în locul datelor intrare obișnuite (concrete), indicând valori arbitrare;
- **execuție simbolică**
  - execuția unui program pentru care datele de intrare sunt valori simbolice;
  - similară unei execuții obișnuite, instrucțiunile fiind reprezentate de operații pe expresii descrise pe baza valorilor simbolice.
    - int m(1, 2) $\Rightarrow$ int m(α, β)
    - valori simbolice folosite: $\alpha, \beta, \alpha_1, \alpha_2, ...$

Execuție simbolică (cont.)

	a	b	c	x	y	z
1	α	β	γ	-	-	-

	a	b	c	x	y	z
1	α	β	γ	-	-	-
2	α	β	γ	α+β	-	-

	a	b	c	x	y	z
1	α	β	γ	-	-	-
2	α	β	γ	α+β	-	-
3	α	β	γ	α+β	β+γ	-

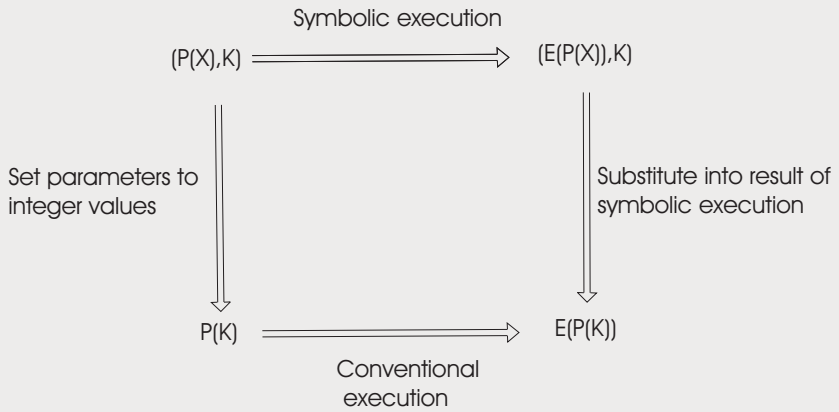
	a	b	c	x	y	z
1	α	β	γ	-	-	-
2	α	β	γ	α+β	-	-
3	α	β	γ	α+β	β+γ	-
4	α	β	γ	α+β	β+γ	α+β+γ

	a	b	c	x	y	z
1	α	β	γ	-	-	-
2	α	β	γ	α+β	-	-
3	α	β	γ	α+β	β+γ	-

- **execuție simbolică** (*engl.* symbolic execution, **SE**);
- funcția sumă;
- SE pentru apelul `suma(`α`,`β`,`γ`)`:
  - `1: int suma(int a, int b, int c){`

Comutativitatea CE – SE

- comutativitatea CE – SE
  - rezultatul obținut prin CE este similar celui obținut aplicând SE;
- CE
  - suma(a, b, c) $\Rightarrow$ suma(1, 3, 5);
  - suma(1, 3, 5) = 9;
- SE
  - suma(a, b, c) $\Rightarrow$ suma(α, β, γ);
  - suma(α, β, γ) = $\alpha + \beta + \gamma$;
  - instanțierea rezultatului simbolic $\Rightarrow \alpha = 1$, $\beta = 3$ și $\gamma = 5 \Rightarrow$ 1+3+5=9;



- rezultatul obținut prin execuția normală cu valori particulare este identic cu cel obținut prin execuția simbolică a programului urmat de instanțierea rezultatului simbolic.

Starea simbolică. Componente

- **stare simbolică** (*engl.* symbolic state)
  - **mulțime de stări concrete** (particulare) neinstanțiate;
- rezultatul unei execuții simbolice este echivalent cu un număr consistent de execuții convenționale ale programului.
- o **stare simbolică** se descrie prin:
  1. **variabile**
     - valorile variabilelor sunt valori/expresii simbolice;
  2. condiția de parcurgere a drumului (*engl.* **path condition**, **pc**)
     - condiții impuse asupra valorilor simbolice;
     - pentru fiecare drum se construiește o *condiție de parcurgere* care va fi verificată dacă este satisfacută sau nu, pentru ca execuția să urmeze drumul asociat;
  3. instrucțiunea executată (*engl.* **program counter**);
     - fiecare instrucțiune este numerotată.

Condiția de parcurgere a unui drum

- **condiția de parcurgere a unui drum** (*engl.* path condition, **pc**)
  - condiția de execuție a unei instrucțiuni;
- particularități ale **pc**:
  - la începutul execuției simbolice avem: **val(**pc**)** $=$ `true`;
  - condiția se cumulează de la o stare (instrucțiune) la alta, reflecând proprietățile drumului curent;
    - dacă τ este condiția de execuție a instrucțiunii I atunci **pc'** $=$ **pc** $\wedge \tau$(I).

Execuția simbolică cu ramificații

- Instrucțiunea `if`
  - `if e`
  - `then A`
  - `else B`
- Instrucțiunea `while`
  - `while e then`
  - `A`
  - `endwhile;`
  - `B`

Instrucțiunea `if`

- execuția simbolică a instrucțiunii `if`
  - `if e`
  - `then A`
  - `else B`
- e – condiție sau expresie simbolică;
- σ(e) – predicatul simbolic asociat expresiei e în starea σ;
- pe durata execuției simbolice **val(σ(e))** $\in$ {`true`, `false`, expresie simbolica} .

Instrucțiunea `if` (cont.)

- ramificarea execuției simbolice a instrucțiunii `if`
  - pe baza condiției curente **pc**, există două expresii, care permit ramificarea execuției:
    a) pc' = pc $\wedge$ σ(e);
    b) pc' = pc $\wedge$ $\neg\sigma$(e);
  - fără ramificare (**non-forking**):
    - dacă a) este `true`, se transmite controlul ramurii `then`;
    - dacă b) este `true`, se transmite controlul ramurii `else`;
  - cu ramificare (**forking**):
    - dacă nu se poate determina dacă a) sau b) este `true`, atunci fiecare alternativă este posibilă și execuția se ramifică pe ambele drumuri;
    - pe ramura `then`, se consideră că a) este `true`;
    - pe ramura `else`, se consideră că b) este `true`.

Instrucțiunea `if`. Execuție convențională

- funcția **nrPar**
  - 1: `boolean nrPar(int x){`
  - 2: `boolean b = false;`
  - 3: `if (x modulo 2 == 0)`
  - 4: `then b = true;`
  - 5: `else b = false;`
  - 6: `nrPar = b;`
  - 7: `}`
- CE pentru apelul `nrPar(6)`=true.

	x	b	If condition
1	6	-	-
2	6	False	-
3	6	False	6 modulo 2=0
4	6	True	6 modulo 2=0
6	6	True	6 modulo 2=0

Instrucțiunea `if`. Execuție simbolică

- funcția **nrPar**
  - 1: `boolean nrPar(int x){`
  - 2: `boolean b = false;`
  - 3: `if (x modulo 2 == 0)`
  - 4: `then b = true;`
  - 5: `else b = false;`
  - 6: `nrPar = b;`
  - 7: `}`
- ramificația 1: `nrPar(`α`)=true,` `dacă (`α `modulo 2 = 0) = true;`
- ramificația 2: `nrPar(`α`)=false,` `dacă not(`α `modulo 2 = 0) = true;`

	x	b	Path condition
1	α	-	True
2	α	False	True
3	α	False	α modulo 2=0
Case (α modulo 2=0) is True			
3	α	False	α modulo 2=0
4	α	True	α modulo 2=0
6	α	True	α modulo 2=0
Case (not (α modulo 2=0)) is True			
5	α	False	not(α modulo 2=0)
6	α	False	not(α modulo 2=0)

Instrucțiunea `while`

- execuția simbolică a instrucțiunii `while`
  - `while e then`
  - `A`
  - `endwhile;`
  - `B`
- e – condiție sau expresie simbolică;
- σ(e) – predicatul simbolic asociat expresiei *e* în starea σ;
- pe durata execuției simbolice **val(σ(e))** $\in \{$`true`, `false`, expresie simbolica$\}$.
- condiția de execuție a blocului A: pc' = pc $\wedge$ σ(e);
- condiția de execuție a blocului B: pc' = pc $\wedge \neg$ σ(e).

Instrucţiunea `while`. Execuţie convenţională

- subalgoritmul **putere**: z=xŷ;
  - 1: `putere(int x, int y, int z){`
  - 2: `z := 1;`
  - 3: `u := 1;`
  - 4: `while(u≤y)`
  - 5: `z := z*x;`
  - 6: `u := u+1;`
  - 7: `endwhile;`
  - 8: `}`
- CE pentru apelul `putere(5, 3, z)`, unde z=125.

	x	y	z	u	While condition
1	5	3	-	-	
2	5	3	1	-	
3	5	3	1	1	
4	5	3	1	1	1<=3
5	5	3	5	1	
6	5	3	5	2	
4	5	3	5	2	2<=3
5	5	3	25	2	
6	5	3	25	3	
4	5	3	25	3	3<=3
5	5	3	125	3	
6	5	3	125	4	
4	5	3	125	4	not 4<=3
7					
8	5	3	125	4	

Instrucțiunea `while`. Execuție simbolică

- subalgoritmul **putere**

```
1:  putere(int x, int y, int z){
2:    z := 1;
3:    u := 1;
4:    while(u≤y)
5:      z := z*x;
6:      u := u+1;
7:    endwhile;
8:  }
```

	x	y	z	u	Path condition	Remarks
1	α	β	-	-	True	
2	α	β	1	-		
3	α	β	1	1		
4	α	β	1	1	1<=β	
Case not(1<=β), → 1>β						
4	α	β	1	1	1>β	
8	α	β	1	1		β=0 and z=1
Case (1<=β)						
4	α	β	1	1	1<=β	
5	α	β	α	1	1<=β	
6	α	β	α	2	1<=β	
7						
4	α	β	α	2	2<=β and 1<=β	
Case not(2<=β) and 1<=β, → 2>β and 1<=β						
4	α	β	α	2	2>β and 1<=β	
8	α	β	α	2		β=1 and z=α
Case (2<=β) and 1<=β						
4	α	β	α	2	2<=β and 1<=β	
5	α	β	α^2	2	2<=β and 1<=β	
6	α	β	α^2	3	2<=β and 1<=β	
7						
4	α	β	α^2	3	3<=β and 2<=β and 1<=β	
Case not(3<=β) and 2<=β and 1<=β → 3>β and 2<=β and 1<=β						
4	α	β	α^2	3	3>β and 2<=β and 1<=β	
8	α	β	α^2	3		β=2 and z=α^2

Arborele execuției simbolice

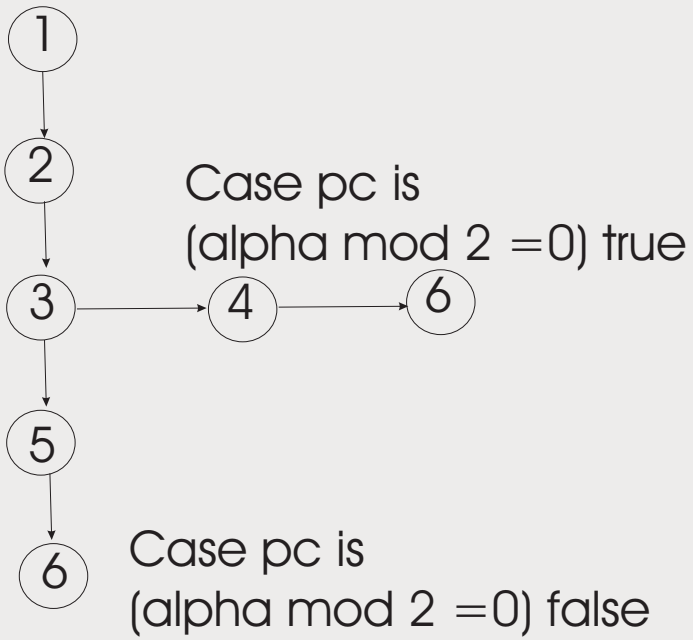
- **arborele execuției simbolice** (*engl.* symbolic execution tree, **SET**)
  - se generează prin descrierea drumurilor de parcurse în timpul execuției simbolice;
- etape de elaborare:
  - se asociază un nod fiecărei instrucțiuni executate;
  - se asociază un arc care conectează nodurile asociate fiecărei tranziții dintre instrucțiuni;
  - nodul asociat instrucțiunii `if` are două arce de ieșire care sunt etichetate cu "T" și "F", corespunzătoare ramurilor `true` și `false`;
  - se asociază o stare curentă completă fiecărui nod identificat, i.e., valorile variabilelor, condiția de parcurgere (**pc**), numărul instrucțiunii.

Arborele execuției simbolice. Funcția sumă.

- funcția sumă;
  - 1: `int suma(int a, int b, int c) {`
  - 2: `  int x := a + b;`
  - 3: `  int y := b + c;`
  - 4: `  int z := x + y - b;`
  - 5: `  return z;`
  - 6: `}`

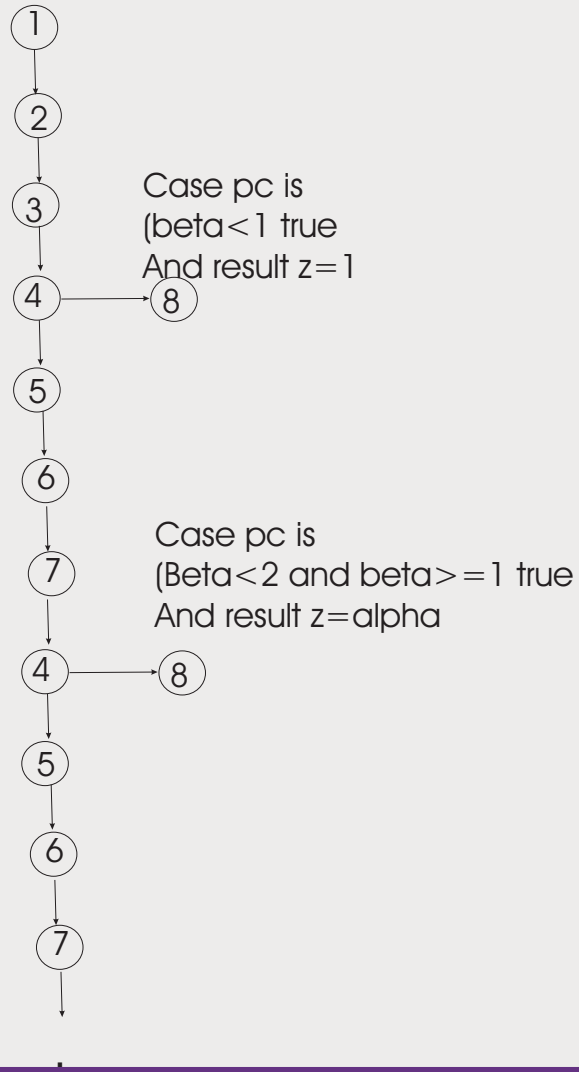
Arborele execuției simbolice. Funcția nrPar.

- funcția **nrPar**
  - 1: `boolean nrPar(int x){`
  - 2: `boolean b = false;`
  - 3: `if (x modulo 2 == 0)`
  - 4: `then b = true;`
  - 5: `else b = false;`
  - 6: `nrPar = b;`
  - 7: `}`



Arborele execuției simbolice. Subalgoritmul putere.

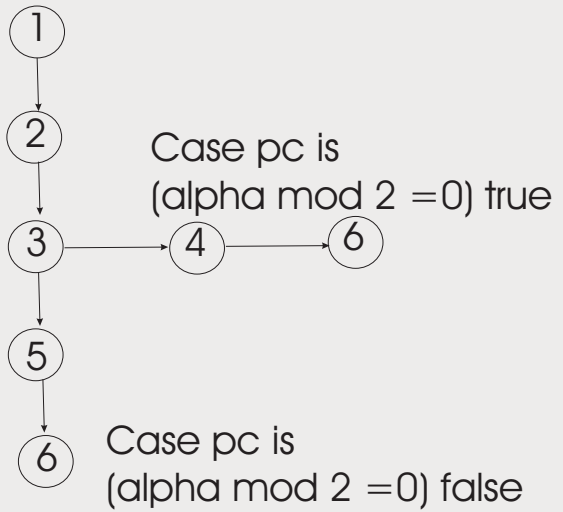
- subalgoritmul **putere**
  - 1: `putere(int x, int y, int z){`
  - 2: `z := 1;`
  - 3: `u := 1;`
  - 4: `while(u≤ y)`
  - 5: `z := z*x;`
  - 6: `u := u+1;`
  - 7: `endwhile;`
  - 8: `}`



Proprietăți ale arborelui de execuție simbolică

1. pentru fiecare nod terminal există date de intrare non-simbolice diferite;
2. **pc** asociat cu orice două noduri terminale este diferit;

- funcția **nrPar**
  - `1: boolean nrPar(int x){`
  - `2: boolean b = false;`
  - `3: if (x modulo 2 == 0)`
  - `4: then b = true;`
  - `5: else b = false;`
  - `6: nrPar = b;`
  - `7: }`



3. **Care este reprezentarea codului sursă, asemănătoare arborelui execuției simbolice, studiată anterior?**
   - SET (*arbore*) = CFG (*graf orientat*) desfășurat.

Aplicabilitatea execuției simbolice

- execuția simbolică permite **construirea predicatelor ce caracterizează condițiile drumurilor parcurse**;
- aplicabilitatea predicatelor identificate pe durata execuției simbolice:
  1. analiza programelor
     - *identificarea drumurilor* nefezabile (neparcurse) sau a celor care ar trebui parcurse în execuție;
  2. identificarea cazurilor de testare
     - *generarea cazurilor de testare* corespunzătoare parcurgerii unor drumuri particulare sau părți concrete ale programului;
  3. verificarea formală (demonstrarea) corectitudinii programelor
     - derivarea predicatelor și utilizarea lor cu instrumentele de demonstrare a teoremelor, e.g., execuția simbolică a drumurilor asociate punctelor de tăietură permite *generarea condițiilor de verificare* – necesare în demonstrarea corectitudinii;
     - *mediile integrate de dezvoltare* (IDE) verifică anumite precondiții la transformările aplicate programelor, e.g., refactorizare.

Generarea cazurilor de testare

- Cazuri de testare
  - fiecare instrucțiune să fie executată cel puțin o dată;
  - fiecare ramificație să fie parcursă cel puțin o dată;
  - Cum identificăm cazurile de testare necesare pentru parcurgerea unui anumit nod al programului?
    - prin instanțierea **pc** cu valori particulare;
- **pc** satisface o clasă de cazuri de testare echivalente și orice soluție care satisface **pc** este fezabilă;
- execuția simbolică permite descrierea rezultatelor programului sub forma unor expresii simbolice, pentru toate datele de intrare din aceeași clasă de echivalență.

Execuția simbolică și corectitudinea programelor [Fre10]

- **SET finit** (fără bucle) $\Rightarrow$ parcurgere exhaustivă a drumurilor:
  - pe baza precondiției și postcondiției;
  - demonstrarea se face pentru drumurile parcurse de la predicatul de intrare până la fiecare predicat de ieșire al SET;
- **SET infinit** (cu bucle); parcurgere exhaustivă a drumurilor **nu** este posibilă $\Rightarrow$ inducție matematică:
  - nu se poate demonstra corectitudinea absolută;
  - aplicarea inducției matematice pentru părțile *infinite* ale SET, reprezintă aplicarea metodei lui Floyd pentru demonstrarea total corectitudinii (**Curs 08**).

Cercetarea în domeniul execuției simbolice (1)

- **1975** - definită ca și concept
- **1976** - King [Kin76], Clarke [Cla76]
- **2005** - Microsoft: DART [GKS05]
- **2006** - Universitatea Stanford: EXE [CGP+06];
  Universitatea Illinois: CUTE și jCUTE [SA06];
- **2007** - Microsoft Research: SAGE;
  conclic executor (concrete+symbolic), bazat pe DART;
  identifica bug-urile din parser-ele asociate unor tipui de fisiere: .jpeg, .docx, .ppt;
  SAGE Impact folosit uzual pentru aplicatiile Windows si Office;
- **2008** - Universitatea Stanford: KLEE [CDE08]; executie simbolica clasica, bazata pe *fork*; continua abordarea EXE si folosit de Apple;

Cercetarea în domeniul execuției simbolice (2)

- **2011** - Universitatea Maryland: Otter; executie simbolica pentru programe descrise in C; incearca generarea unui caz de testare pentru o linie de cod data;
- **2012** - Universitatea Carnegie Mellon: Mayhem; executie simbolica pentru fisiere binare;
- **2014** - Universitatea Carnegie Mellon: Mergepoint; folosit uzual in Linux;
- **1999 - 2016** - NASA: Symbolic (Java) Path Finder (JPF) [PV09, CS13].

Pentru examen...

- definiţii: valoare simbolică, execuţie simbolică, stare simbolică(variabile + pc + counter);
- descriere: comutativitate CE – SE;
- execuţia simbolică a structurilor secvenţiale, alternative şi repetitive;
- SET: definiţie, proprietăţi, construire SET, CFG vs. SET;
- execuţia simbolică şi generarea cazurilor de testare.

Bibliografie I

[CDE08] C. Cadar, d. Dunbar, and D.E Engler.
Klee: Unassisted and automatic generation of high-coverage tests for complex systems programs.
In *OSDI*, pages 209–224, 2008.

[CGP+06] C. Cadar, V. Ganesh, P.M. Pawlowski, D.L. Dill, and D.R. Engler.
Exe: automatically generating inputs of death.
In *ACM Conference on Computer and Communications Security*, pages 322–335, 2006.

[Cla76] L. A. Clarke.
A system to generate test data and symbolically execute programs.
IEEE Trans. Softw. Eng., 2(3):215–222, 1976.

[CS13] C. Cadar and K. Sen.
Symbolic execution for software testing: three decades later.
Communications of the ACM, 56(2):82–90, 2013.

[Fre10] M. Frentiu.
Verificarea și validarea sistemelor soft.
Presa Universitară Clujeană, 2010.

[GKS05] P. Godefroid, N. Klarlund, and K. Sen.
Dart: directed automated random testing.
In *Proceedings of the 2005 ACM SIGPLAN conference on Programming language design and implementation*, pages 213–223. ACM, 2005.

Bibliografie II

[Kin76] J. C. King.
Symbolic execution and program testing.
Communications of ACM, 19(7):385–394, 1976.

[PV09] C.S. Pasareanu and W. Visser.
A survey of new trends in symbolic execution for software testing and analysis.
STTT, 11(4):339–353, 2009.

[SA06] K. Sen and G. Agha.
Cute and jcute: Concolic unit testing and explicit path model-checking tools.
In *CAV*, pages 419–423, 2006.