Software Systems Verification and Validation

Lecture 05 - Symbolic execution

Lect. dr. Andreea Vescan

Babeş-Bolyai University
Cluj-Napoca

2015-2016

Symbolic execution - research

- 1975 - First introduced
- King [Kin76], Clarke [Cla76]
- 2005 - Microsoft: DART [God05]
- 2006 - Universitatea Stanford: EXE, Universitatea Illinois: CUTE si jCUTE [SA06]
- 2008 - KLEE [CDE08]
- 1999 - 2016 - NASA: Symbolic (Java) Path Finder [PV09], [CS13]

What is symbolic execution ?

- Symbolic execution
  - Execution of program with symbols as argument.
  - Symbolic execution supplies symbols (as input to a program) representing arbitrary values.
  - int FunctionName(1, 2) ⇒ int FunctionName(a1 , a2)
- The execution proceeds as in a normal execution except that values may be symbolic formulae over the input symbols.

Symbolic state.

- Symbolic state
  - Set of (particular) concrete states, yet not instantiated.
  - Symbolic states represent sets of concrete states.
- A symbolic state is described by:
  - Variables, i.e. symbolic values/expressions for variables;
  - Path condition - a conjunct of constraints on the symbolic input values;
  - Program counter - the statement that is executed.

Conventional execution (CE)

- Function Sum
- 1: int Sum(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:
- Normal execution result of Sum(1,3,5)

Conventional execution (CE) (cont.)

- Function Sum
- Normal execution result of Sum(1,3,5)
- 1 : int Sum(int a, int b, int c)
- 2 : int x := a + b;

	a	b	c	x	y	z
1	1	3	5	-	-	-

Conventional execution (CE) (cont.)

- Function Sum
- Normal execution result of Sum(1,3,5)
- 1 : int Sum(int a, int b, int c)
- 2 : int x := a + b;
- 3: int y := b + c;

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

Conventional execution (CE) (cont.)

- Function Sum
- Normal execution result of Sum(1,3,5)
- 1 : int Sum(int a, int b, int c)
- 2 : int x := a + b;
- 3: int y := b + c;
- 4: int z := $x + y - b$;

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

Conventional execution (CE) (cont.)

- Function Sum
- Normal execution result of Sum(1,3,5)
- 1 : int Sum(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;

	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

Conventional execution (CE) (cont.)

- Function Sum
- Normal execution result of Sum(1,3,5)
- 1 : int Sum(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

Symbolic execution (SE)

- Function Sum
- Symbolic execution result of Sum(α,β,γ)
- 1 : int Sum(int a, int b, int c)

	a	b	c	x	y	z
1	α	β	γ	-	-	-

Symbolic execution (SE)

- Function Sum
- Symbolic execution result of Sum(α,β,γ)
- 1 : int Sum(int a, int b, int c)
- 2 : int x := a + b;

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

Symbolic execution (SE)

- Function Sum
- Symbolic execution result of Sum(α,β,γ)
- 1 : int Sum(int a, int b, int c)
- 2 : int x := a + b;
- 3: int y := b + c;

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

Symbolic execution (SE)

- Function Sum
- Symbolic execution result of Sum(α,β,γ)
- 1 : int Sum(int a, int b, int c)
- 2 : int x := a + b;
- 3: int y := b + c;
- 4: int z := *x* + *y* - *b*;

	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

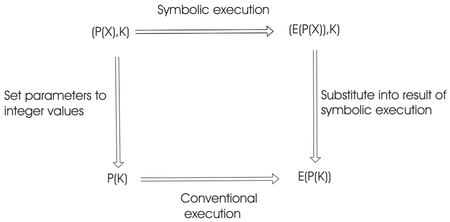
Symbolic execution (SE)

- Function Sum
- Symbolic execution result of Sum(α,β,γ)
- 1 : int Sum(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	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
5	α	β	γ	α+β	β+γ	α+β+γ

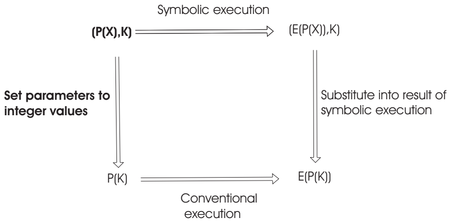
Commutativity

- The same result is obtained using normal execution or using symbolic execution.
- Conventional execution (CE)



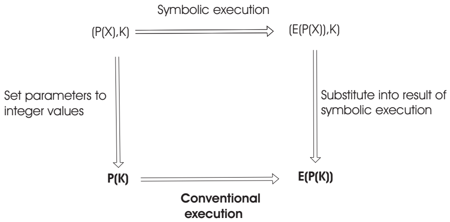
Commutativity

- The same result is obtained using normal execution or using symbolic execution.
- Conventional execution (CE)
    - Sum(a, b, c) $\Rightarrow$ Sum(1, 3, 5)



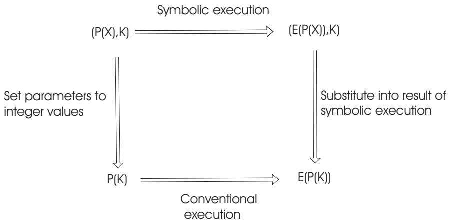
Commutativity

- The same result is obtained using normal execution or using symbolic execution.
- Conventional execution (CE)
  - Sum(a, b, c) $\Rightarrow$ Sum(1, 3, 5)
  - Sum(1, 3, 5) $= 9$



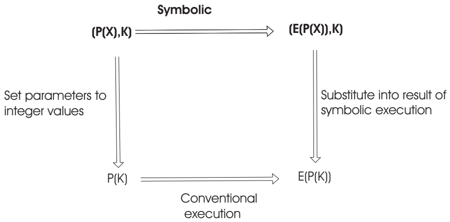
Commutativity

- The same result is obtained using normal execution or using symbolic execution.
- Conventional execution (CE)
  - Sum(a, b, c) $\Rightarrow$ Sum(1, 3, 5)
  - Sum(1, 3, 5) $=$ 9
- Symbolic execution (SE)



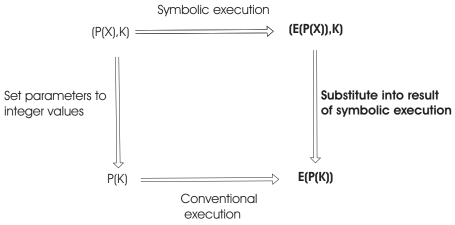
Commutativity

- The same result is obtained using normal execution or using symbolic execution.
- Conventional execution (CE)
  - Sum(a, b, c) $\Rightarrow$ Sum(1, 3, 5)
  - Sum(1, 3, 5) $= 9$
- Symbolic execution (SE)
  - Sum(a, b, c) $= \alpha + \beta + \gamma$



Commutativity

- The same result is obtained using normal execution or using symbolic execution.
- Conventional execution (CE)
  - Sum(a, b, c) $\Rightarrow$ Sum(1, 3, 5)
  - Sum(1, 3, 5) $=$ 9
- Symbolic execution (SE)
  - Sum(a, b, c) $= \alpha + \beta + \gamma$
  - Instantiate the symbolic result $\Rightarrow$ $\alpha = 1$, $\beta = 3$ and $\gamma = 5 \Rightarrow$ $1+3+5=9$.



Sequential structure execution

- path condition
  - condition to execute a statement;
- when the symbolic execution starts, the value(pc) = true
- the condition is updated from one statement to other
  - If τ represents the condition to execute statement $< I >$ then $pc' = pc \wedge \tau(I)$

Sequential execution - Conventional and Symbolic

- See previous slides examples for Function Sum.

If statement

- Symbolic execution of an IF statement
  - if η then
  - A
  - else
  - B.
- During symbolic execution $\Rightarrow$ **value(η)** could be **true**, **false**, or some symbolic formula over the input symbols.
  $\Rightarrow$ "unresolved" execution of a conditional statement
- If value(η) and reaching a statement with condition τ
  $\Rightarrow$ value(η) $\supset$ value(τ) or value(η) $\supset \neg$ value(τ)
- Path Condition (Initial value of *pc* is true)
  - pc $\rightarrow \eta$
  - pc $\rightarrow \neg\eta$

Conventional execution

- Function IsEven
- 1 : boolean IsEven(int a)
- 2 : boolean b := False;
- 3: If (x modulo 2 =0) then
- 4: b:=true;
- else
- 5: b:=false;
- 6: IsEven:=b;
- 7:

	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

Symbolic execution

- Function IsEven
- 1 : boolean IsEven(int a)
- 2 : boolean b := False;
- 3: If (x modulo 2 = 0) then
- 4: b:=true;
- else
- 5: b:=false;
- 6: IsEven:=b;
- 7:

	x	b	Path condition
1	α	-	True

Symbolic execution

- Function IsEven
- 1 : boolean IsEven(int a)
- 2 : boolean b := False;
- 3: If (x modulo 2 = 0) then
- 4: b:=true;
- else
- 5: b:=false;
- 6: IsEven:=b;
- 7:

	x	b	Path condition
1	α	-	True
2	α	False	True

Symbolic execution

- Function IsEven
- 1 : boolean IsEven(int a)
- 2 : boolean b := False;
- 3: If (x modulo 2 = 0) then
- 4: b:=true;
- else
- 5: b:=false;
- 6: IsEven:=b;
- 7:

	x	b	Path condition
1	α	-	True
2	α	False	True
3	α	False	α modulo 2=0

Symbolic execution

- Function IsEven
- 1 : boolean IsEven(int a)
- 2 : boolean b := False;
- 3: If (x modulo 2 = 0) then
- 4: b:=true;
- else
- 5: b:=false;
- 6: IsEven:=b;
- 7:

	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

Symbolic execution

- Function IsEven
- 1 : boolean IsEven(int a)
- 2 : boolean b := False;
- 3: If (x modulo 2 = 0) then
- 4: b:=true;
- else
- 5: b:=false;
- 6: IsEven:=b;
- 7:

	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

Symbolic execution

- Function IsEven
- 1 : boolean IsEven(int a)
- 2 : boolean b := False;
- 3: If (x modulo 2 = 0) then
- 4: b:=true;
- else
- 5: b:=false;
- 6: IsEven:=b;
- 7:

	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

Symbolic execution

- Function IsEven
- 1 : boolean IsEven(int a)
- 2 : boolean b := False;
- 3: If (x modulo 2 = 0) then
- 4: b:=true;
- else
- 5: b:=false;
- 6: IsEven:=b;
- 7:

	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)

Symbolic execution

- Function IsEven
- 1 : boolean IsEven(int a)
- 2 : boolean b := False;
- 3: If (x modulo 2 = 0) then
- 4: b:=true;
- else
- 5: b:=false;
- 6: IsEven:=b;
- 7:

	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)

While statement

- Symbolic execution of an WHILE statement
  - while η then
  - A
  - endWh;
  - B.
- During symbolic execution $\Rightarrow$ **value(η)** could be **true**, **false**, or some symbolic formula over the input symbols.
  $\Rightarrow$ "unresolved" execution of a conditional statement
- Condition to execute A: pc for executing "while" and η.
- Condition to execute B: pc for executing "while" and $\neg\ \eta$.

Conventional execution

- Subalg. Power
- 1 : Power(int x, int y, int z)
- 2 : z := 1;
- 3: u:=1
- 4: while($u \leq y$)
- 5: z:=z*x;
- 6: u:=u+1
- 7: endwh;
- 8:

	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	5	3	3<=3
5	5	3	75	3	
6	5	3	75	4	
4	5	3	75	4	not 4<=3

Symbolic execution

- Subalg. Power
- 1 : Power(int x, int y, int z)
- 2 : z := 1;
- 3: u:=1
- 4: while($u \leq y$)
- 5: z:=z*x;
- 6: u:=u+1
- 7: endwh;
- 8:

	x	y	z	u	Path condition	Remarks
1	α	β	-	-	True	

Symbolic execution

- Subalg. Power
- 1 : Power(int x, int y, int z)
- 2 : z := 1;
- 3: u:=1
- 4: while($u \leq y$)
- 5: z:=z*x;
- 6: u:=u+1
- 7: endwh;
- 8:

	x	y	z	u	Path condition	Remarks
1	α	β	-	-	True	
2	α	β	1	-		

Symbolic execution

- Subalg. Power
- 1 : Power(int x, int y, int z)
- 2 : z := 1;
- 3: u:=1
- 4: while($u \leq y$)
- 5: z:=z*x;
- 6: u:=u+1
- 7: endwh;
- 8:

	x	y	z	u	Path condition	Remarks
1	α	β	-	-	True	
2	α	β	1	-		
3	α	β	1	1		

Symbolic execution

- Subalg. Power
- 1 : Power(int x, int y, int z)
- 2 : z := 1;
- 3: u:=1
- 4: while($u \leq y$)
- 5: z:=z*x;
- 6: u:=u+1
- 7: endwh;
- 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

Symbolic execution

- Subalg. Power
- 1 : Power(int x, int y, int z)
- 2 : z := 1;
- 3: u:=1
- 4: while($u \leq y$)
- 5: z:=z*x;
- 6: u:=u+1
- 7: endwh;
- 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	

Symbolic execution

- Subalg. Power
- 1 : Power(int x, int y, int z)
- 2 : z := 1;
- 3: u:=1
- 4: while($u \leq y$)
- 5: z:=z*x;
- 6: u:=u+1
- 7: endwh;
- 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	

Symbolic execution

- Subalg. Power
- 1 : Power(int x, int y, int z)
- 2 : z := 1;
- 3: u:=1
- 4: while($u \leq y$)
- 5: z:=z*x;
- 6: u:=u+1
- 7: endwh;
- 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						

Symbolic execution

- Subalg. Power
- 1 : Power(int x, int y, int z)
- 2 : z := 1;
- 3: u:=1
- 4: while($u \leq y$)
- 5: z:=z*x;
- 6: u:=u+1
- 7: endwh;
- 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	

Symbolic Execution Tree

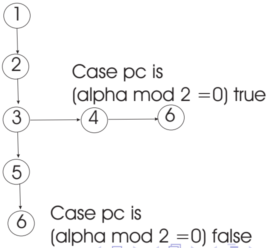
- We can generate symbolic execution tree characterizing the execution paths followed during the symbolic execution.
  - Associate a node with each statement executed.
  - Associate a directed arc connecting the associated nodes with each transition between statements.
  - For IF statement execution, the associated node has two arcs leaving the node which are labeled "T" and "F" for the true and false part, respectively.
  - Associate the complete current execution state, i.e. variable values, statement counter, and pc with each node.

Symbolic Execution Tree

- Function Sum
- 1: int Sum(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:

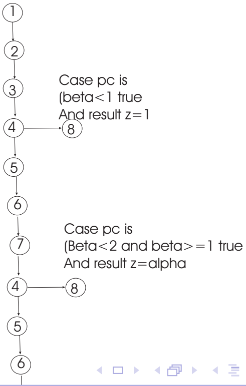
Symbolic Execution Tree

- Function IsEven
- 1 : boolean IsEven(int a)
- 2 : boolean b := False;
- 3: If (x modulo 2 =0) then
- 4: b:=true;
- else
- 5: b:=false;
- 6: IsEven:=b;
- 7:



Symbolic Execution Tree

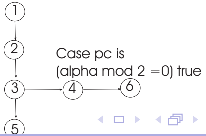
- Subalg. Power
- 1 : Power(int x, int y, int z)
- 2 : z := 1;
- 3: u:=1
- 4: while($u \leq y$)
- 5: z:=z*x;
- 6: u:=u+1
- 7: endwh;
- 8:



Properties of the Symbolic Execution Tree

- For each terminal leaf exists a particular nonsymbolic input.
- The pc associated with any two terminal leaves are distinct.

- Function IsEven
- 1 : boolean IsEven(int a)
- 2 : boolean b := False;
- 3: If (x modulo 2 =0) then
- 4: b:=true;
- else
- 5: b:=false;



Test case generation

- Test cases:
  - to execute every statement at least once.
  - to include execution of each branch both ways.
  - finding input values to reach a particular point in a program ?

  Remaining problem - to instantiate the pc with particular values.
- The pc specifies a class of equivalent tests, and any feasible solution to the constraints (represented by the pc) would be a representative member.
- The symbolic execution also provides expressions describing the program outputs for all inputs in this set.

Correctness by Symbolic testing

- Informal induction
- Program verification
  - A proof to be performed in terms of symbolic execution, based on standard inductive assertion method.

Next lecture

- Lecture 06 - Compulsory Attendance
- Date: 1 April 2016 - FRIDAY
- Hours: 12:00-14:00
- Room: A2, FSEGA Building
- EVOZON presentation
- Testing Automation. Selenium WebDriver

Bibliografie I

[CDE08] Cristian Cadar, Daniel Dunbar, and Dawson Engler.
Klee: Unassisted and automatic generation of high-coverage tests for complex systems programs.
In *Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation*, pages 209–224, 2008.

[Cla76] L. A. Clarke.
A system to generate test data and symbolically execute programs.
IEEE Transactions on Software Engineering, SE-2(3):215–222, 1976.

[CS13] Cristian Cadar and Koushik Sen.
Symbolic execution for software testing: Three decades later.
Commun. ACM, 56(2):82–90, 2013.

[God05] P. Godefroid.
Dart: directed automated random testing.
pages 213–223, 2005.

Bibliografie II

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

[PV09] Corina S. Pasareanu and Willem Visser.
A survey of new trends in symbolic execution for software testing and analysis.
Int. J. Softw. Tools Technol. Transf., 11(4):339–353, 2009.

[SA06] Koushik Sen and Gul Agha.
Cute and jcute: Concolic unit testing and explicit path model-checking tools.
In *Proceedings of the 18th International Conference on Computer Aided Verification*, pages 419–423, 2006.