Generating Checking Sequences: When Reseting is not an Option

Adenilso Simão - USP - Brazil
adenilso@icmc.usp.br

Departamento de Sistemas de Computação
Instituto de Ciências Matemáticas e de Computação
Universidade de São Paulo

TAROT - Paris - France - 2016-07-04

2 Agenda

- Goals
  - To present the main concept of checking sequence generation
  - To present recent methods
  - To demonstrate why those methods work
  - To point future research
- Public
  - Newcomers to the area
    - Intuition over formulae
  - New PhD students

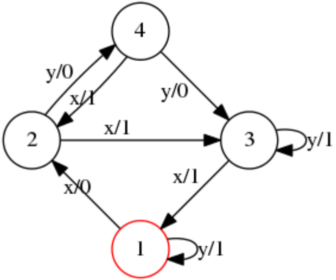
3 Model Based Testing

- Test Generation is Always Model-Based
  - Implicit models
    - System Understanding
  - Explicit models
    - Diagrams
    - State Machines

4 State Machines

- Simplest explicit models
  - Vanilla models
  - Understandable for non-experts
  - Semantic is the model itself

5 Finite State Machine



- It can be seen as
  - A regular language over pairs of input and outputs
  - A function from inputs sequences to output sequences

6 Test from State Machines

- Given a specification FSM
- Given an implementation
  - As a black-box
  - Only output sequences (in response to input sequences) are observable
- Is the implementation correct?
  - Does it behave accordingly?
  - Does it represent the same function?
    - Or an equivalent one (in some sense)?

7 Test from State Machines (II)

- Is it even possible to answer that?
  - A failed test is a negative answer
- For a positive answer
  - Is a finite test enough?

8 Testing Hypothesis

- Enter testing hypothesis
  - Without assumptions, the problem is unsolvable
  - With too many assumptions, the problem is trivial
  - With the right assumptions, the problem is interesting

9 Testing Hypothesis (II)

- Modelling assumption
  - The implementation can be modelled as an (unknown) FSM
    - Big assumption
    - Reduces the complexity of knowing how to test
- Input Compatibility
  - The implementation accepts the same inputs as the specification

10 Testing Hypothesis (III)

- Boundness
  - There is a known upper bound on the number of state in the unknown FSM
    - This is the most disputable one!
- Determinism
  - Always the same answer to a given input sequence
    - Verifiable in the specification
    - Assumed in the implementation

11 Checking experiments

- A set of input sequences (with corresponding output sequences) which identify uniquely the specification
  - Resets are used to bring the specification and the implementation the initial state
  - It is assumed to be reliable in the implementation
    - Yet another assumption

12 Checking experiments

- Generation Methods
  - W[1]
  - Wp[2]
  - HSI[3]

[1] T. S. Chow. “Testing Software Design Modeled by Finite-State-Machines”. In: *IEEE Transactions on Software Engineering* 4.3 (May 1978), pp. 178–186.

[2] Susumu Fujiwara et al. “Test Selection Based on Finite State Models”. In: *IEEE Trans. Software Eng.* 17.6 (1991), pp. 591–603.

[3] N. Yevtushenko and A. Petrenko. “Synthesis of test experiments in some classes of automata”. In: *Automatic Control and Computer Sciences* 24.4 (1990), pp. 50–55.

13 Checking experiments (II)

- Generation Methods
  - H[4]
  - SPY[5]

[4] Rita Dorofeeva, Khaled El-Fakih, and Nina Yevtushenko. "An Improved Conformance Testing Method". In: *Formal Techniques for Networked and Distributed Systems - FORTE 2005, 25th IFIP WG 6.1 International Conference, Taipei, Taiwan, October 2-5, 2005, Proceedings*. 2005, pp. 204–218.

[5] Adenilso Simao, Alexandre Petrenko, and Nina Yevtushenko. "Generating Reduced Tests for FSMs with Extra States". In: *Testing of Software and Communication Systems, 21st IFIP WG 6.1 International Conference, TESTCOM 2009 and 9th International Workshop, FATES 2009, Eindhoven, The Netherlands, November 2-4, 2009. Proceedings*. 2009, pp. 129–145.

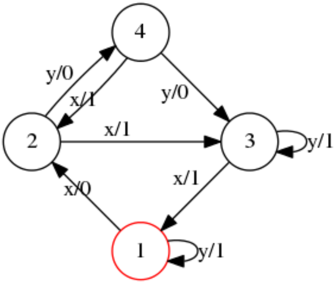
14 Checking experiments (III)

- Test Cases from HSI Method
  - $\{xxxx, xxyx, xxyy, xyxxy, xyxy, xyyx, xyyy, yx\}$
    - Length 39
- Test Cases from SPY Method
  - $\{xxxx, xxyx, xyxxyy, xyxyyy, xyyx, yx\}$
    - Length 32

15 Checking sequence

- A checking experiment with a single input sequence
  - No resets required
  - Strongly connected

16 Generation Methods

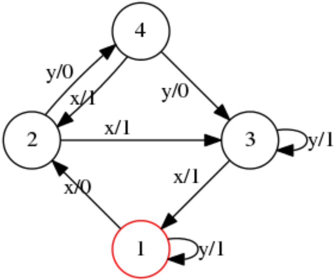


- Assume the FSM
- Assume the implementation can be modeled as an FSM with same input alphabet and at most 4 states

17 Generation Methods (II)

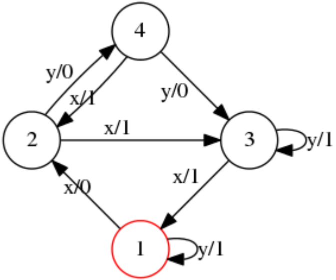
- Consider the input sequence
  - $\omega = yxyxyxyyxyxxyxyxxyyxyyxyyyx$
- It is a checking sequence
  - Why?

18 # Generation Methods (III)



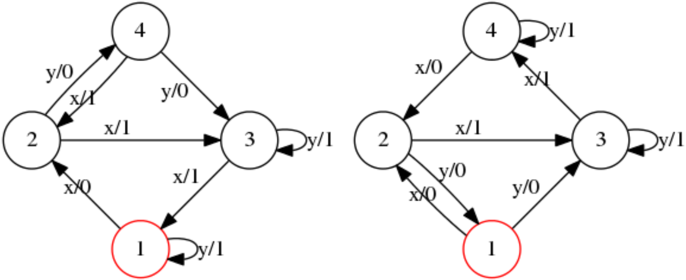
- Consider the input sequence
  - $\omega = yxyxyxyyxyxxyxyxxyyxyyxyyyx$
    - Length 27
- The output sequence is
  - $\mu = 100101001101111011111100011$
- There is only one (out of more than 16 millions) FSM with at most 4 states, which answer ω with μ
  - There are possible renamed FSMs

19 Generation Methods (IV)



- Consider the input sequence
  - $\omega' =$ *xyyyxxxyxyxxyxyxxxyxxyxyyxyyxyyxyyxyxyyyyxyyxxyyxxyyxxyyxx*
    - Length 58
- The output sequence is
  - $\mu' =$ 0001101111011110111011111000111000110001111101111000100010

20 Generation Methods (V)



- Consider the input sequence
  - $\omega' =$ *xyyyxxxyxyxxyxyxxxyxxyxyyxyyxyyxyyxyxyyyyxyyxxyyxxyyxxyyxx*
- The output sequence is
  - $\mu' =$ 0001101111011110111011111000111000110001111101111000100010

21 Checking Sequence

- What is a checking sequence after all
  - Given a specification FSM
    - An input sequence (with the respective output sequence) which identifies uniquely (up to isomorphism) this FSM among a set of candidate FSMs
- The candidate FSMs
  - Set of FSMs with at most as many states as the specification FSM
  - The fault domain

22 Using a Checking Sequence

- Given a checking sequence
- Given a black-box implementation
- Assuming the implementation can be modeled by an FSM from the fault domain
- If the implementation passes the test (i.e., it produces the expected output)
  - The implementation is correct

23 Generating Checking Sequences

- Long tradition
  - Moore, 1958[6]
    - The seminal paper
    - The problem is set here

[6] Edward F. Moore. "Gedanken-Experiments on Sequential Machines". In: *J. Symbolic Logic* 23.1 (1958).

24 Generating Checking Sequences (II)

- Long tradition
  - Hennie, 1965[7]
    - Generating checking sequences
    - The method is quite good, but not very algorithmic

[7] F. C. Hennie. “Fault-detecting experiments for sequential circuits”. In: *Proceedings of Fifth Annual Symposium on Circuit Theory and Logical Design*. 1965, pp. 95–110.

25 Generating Checking Sequences (III)

- Long tradition
  - Gonenc, 1970[8]
    - An algorithmic method
    - Let us have a look

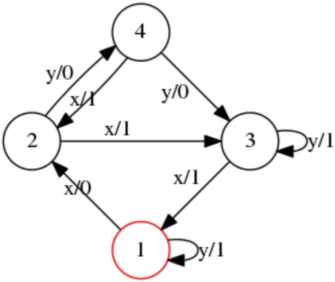
[8] G. Gonenc. “A method for the design of fault detection experiments”. In: *IEEE Transactions on Computers* 19 (1970), pp. 551–558.

26 Distinguishing States

- A way to distinguish all states of the FSMs[9]
- Distinguishing Sequence
  - Preset
    - An input sequence
  - Adaptive
    - An decision tree (nodes are inputs, edges are outputs)
- Distinguishing Set
  - A preset set of sequences, which common prefixes
  - Equivalent to an Adaptive Distinguishing Sequence

[9] David Lee and Mihalis Yannakakis. "Testing Finite-State Machines: State Identification and Verification". In: *IEEE Trans. Computers* 43.3 (1994), pp. 306–320.

27 Distinguishing States (II)



- Preset Distinguishing Sequence
- $X_d = yxy$
  - State 1 answers with 100
  - State 2 answers with 010
  - State 3 answers with 111
  - State 4 answers with 011

28 Distinguishing States (III)

- Insights
  - X_d can be used to identify a unknown state of the machine
  - X_d can be used to confirm that the machine is in a given state
  - If X_d is applied to every state of the specification and the implementation
  - If the implementation answers as the specification
    - Then, the implementation has at least 4 states
    - X_d is a preset distinguishing sequence for the implementation

29 Distinguishing States (IV)

- Insights
  - $X_d X_d$ can be used to confirm the state reached by the first application of X_d
  - If $X_d X_d$ is applied in each state
    - Then, we can identify which state the implementation is in

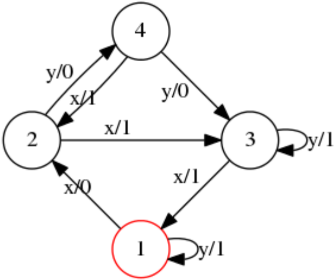
30 Distinguishing States (V)

- Insights
  - For a given transition from a (known) state with a given input
    - X_d can be used to confirm that the reached state is correct

31 Generating a Checking Sequence

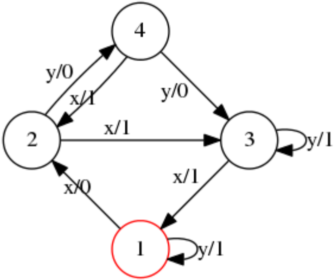
- α-sequences
  - Recognizing states
- β-sequences
  - Confirming transitions
- T-sequences (transfer sequences)
  - Bridging from one state to another
  - Gluing the sequences
    - Avoiding circularity in the assumptions

32 Generating a Checking Sequence (II)



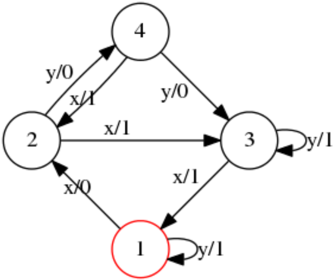
- Consider the sequence
  - $X_d X_d X_d = yxyyxyyxy$ from state 1
  - with outputs y.1 x.0 y.0 y.0 x.1 y.1 y.1 x.0 y.0

33 Generating a Checking Sequence (III)



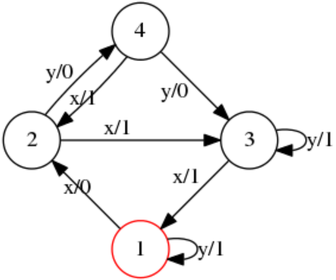
- Consider the sequence
  - $X_d X_d X_d = yxyyxyyxy$ from state 1
  - with outputs [1]y.1 x.0 y.0 [4]y.0 x.1 y.1 [1]y.1 x.0 y.0

34 Generating a Checking Sequence (IV)



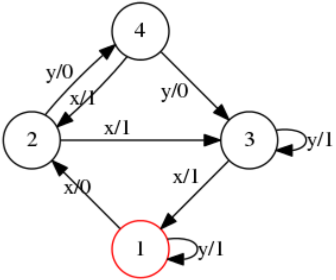
- Consider the sequence
  - $X_d X_d X_d = yxyyxyyxy$ from state 1
    - with outputs [1]y.1 x.0 y.0 [4]y.0 x.1 y.1 [1]y.1 x.0 y.0 (4)
  - This is a α-sequence $(1, X_d X_d X_d)$

35 Generating a Checking Sequence (V)



- Consider the sequence
  - $X_d X_d = yxyyxy$ from state 2
    - with outputs [2] y.0 x.1 y.0 [4] y.0 x.1 y.1 (4)
  - This is another α-sequence $(2, X_d X_d)$

36 Generating a Checking Sequence (VI)



- Consider the sequence
  - $X_d X_d = yxyyxy$ from state 3
    - with outputs [3] y.1 x.1 y.1 [1] y.1 x.0 y.0 (4)
  - This is another α-sequence

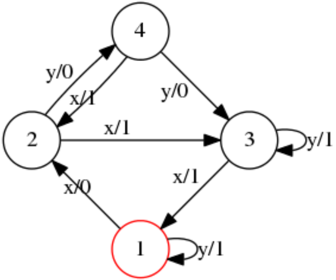
37 Generating a Checking Sequence (VII)

- The alpha set is the set of α-sequences, marked with the respective starting states
  - There are three $(1, X_d X_d X_d), (2, X_d X_d), (3, X_d X_d)$

38 Generating a Checking Sequence (VIII)

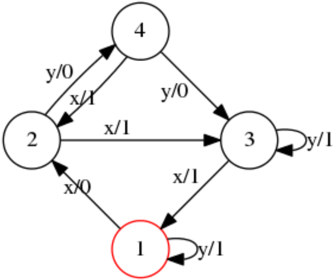
- The β-sequences are generated per transition
  - Consider the transition $(1, x)$
    - The corresponding β-sequence is $xX_d = (1)x.0[2]y.0x.1y.0(4)$
    - Actually $(1, xX_d)$
  - There are, then, eight β-sequences, one for each transitions

39 Generating a Checking Sequence (IX)



- Testing fragments
  - $(1, X_d X_d X_d, 4)$, $(2, X_d X_d, 4)$, $(3, X_d X_d, 4)$
  - $(1, xX_d, 4)$, $(1, yX_d, 4)$, $(2, xX_d, 1)$, $(2, yX_d, 1)$, $(3, xX_d, 4)$, $(3, yX_d, 1)$, $(4, xX_d, 4)$, $(4, yX_d, 1)$

40 Generating a Checking Sequence (X)



- Gluing them
  - Using transfer sequences (T-sequences) if needed
    - Avoid circularity
  - $(1, X_d X_d X_d, 4)$ $(4, x, 2)$ $(2, X_d X_d, 4)$ $(4, y, 3)$ $(3, X_d X_d, 4)$ $(4, xX_d, 4)$ $(4, yX_d, 1)$ $(1, yX_d, 4)$ $(4, x, 2)$ $(2, xX_d, 1)$ $(1, xX_d, 4)$ $(4, x, 2)$ $(2, yX_d, 1)$ $(1, xx, 3)$ $(3, xX_d, 4)$ $(4, y, 3)$ $(3, yX_d, 1)$

41 Generating a Checking Sequence (XI)

- Extracting the checking sequence
  - $X_d X_d X_d x X_d X_d y X_d X_d x X_d y X_d y X_d xx X_d x X_d xy X_d xxx X_d yy X_d$
  - *yxyyxyyxyxyxyyxyyxyyyyxyyxyxyxyyyxyyyxyxxyxyxyxyxyxyxyyxyxxxyxyyyyyxy*
    - Length 60

42 Generating a Checking Sequence (XII)

- Optimizing T-sequences[10]
  - Generating a graph with α-, β- and (sort of) T-sequences
  - Find the shortest sequence with all α- and β-sequences
    - T-sequences are optional
    - Rural Chinese Postman Problem (RCPP)[11]
  - In the best scenario, no T-sequences.
    - The shortest possible with this approach is of length 53

[10] H. Ural, X. Wu, and F. Zhang. "On minimizing the lengths of checking sequences". In: *IEEE Transactions on Computers* 46.1 (1997), pp. 93–99.

[11] R. M. Hierons and H. Ural. "Optimizing the length of checking sequences". In: *IEEE Transactions on Computers* 55.5 (2006), pp. 618–629.

43 Local Optimization Method

- Greedy approach[12]
- Until all transitions are verified
  - (Case 1) the current state is not recognized
    - Apply the distinguishing sequence for that state
  - (Case 2) the current state is recognized and there is a non-verified input at the end state
    - Apply the input plus the distinguishing sequence
  - (Case 3) the current state is recognized and all inputs are verified at the end state
    - Transfer to a state with non-verified input, using only verified transitions

[12] Adenilso Simao and Alexandre Petrenko. "Generating Checking Sequences for Partial Reduced Finite State Machines". In: *Testing of Software and Communicating Systems, 20th IFIP TC 6/WG 6.1 International Conference, TestCom 2008, 8th International Workshop, FATES 2008, Tokyo, Japan, June 10-13, 2008, Proceedings*. 2008, pp. 153–168.

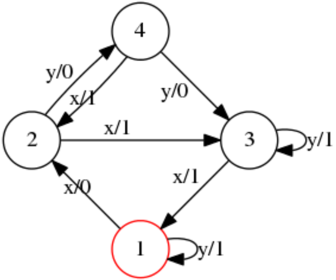
44 Local Optimization Method (II)

- Insights
  - Sometimes it is possible to use shorter sequence to distinguish
    - As in[13]
  - Sometimes a transition is verified indirectly
    - As in[14]

[13] Hasan Ural and Fan Zhang. "Reducing the Lengths of Checking Sequences by Overlapping". In: *Testing of Communicating Systems, 18th IFIP TC6/WG6.1 International Conference, TestCom 2006, New York, NY, USA, May 16-18, 2006, Proceedings*. 2006, pp. 274–288.

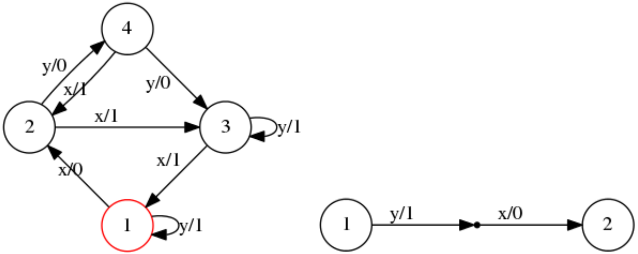
[14] Jessica Chen et al. "Eliminating Redundant Tests in a Checking Sequence". In: *Testing of Communicating Systems, 17th IFIP TC6/WG 6.1 International Conference, TestCom 2005, Montreal, Canada, May 31 - June 2, 2005, Proceedings*. 2005, pp. 146–158.

45 Local Optimization Method (III)



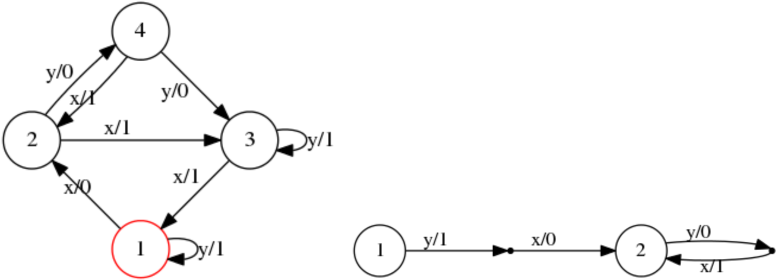
- $X_d^1 = y.1x.0$
- $X_d^2 = y.0x.1y.0$
- $X_d^3 = y.1x.1$
- $X_d^4 = y.0x.1y.1$

46 Local Optimization Method (IV)



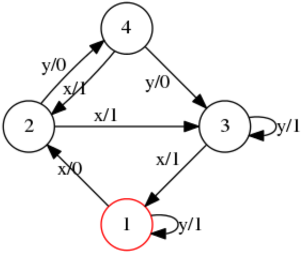
- We start applying X_d
  - $\omega = [1]y.1x.0$
- We apply X_d again
  - $\omega = [1]y.1x.0[2]y.0x.1y.0$
  - The fragment $(1, yx, 2)$ is verified

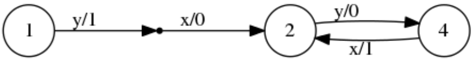
47 Local Optimization Method (V)



- We apply X_d again, but using the fact that a suffix of ω is a prefix of X_d
  - $\omega = [1]y.1x.0[2]y.0x.1[2]y.0x.1y.0$
  - The fragment $(2, yx, 2)$ is verified
  - Then
    - $\omega = [1]y.1x.0[2]y.0x.1[2]y.0x.1[2]y.0$

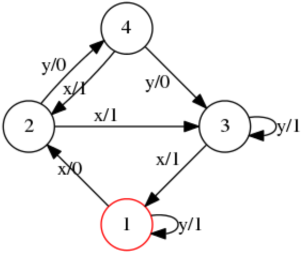
Local Optimization Method (VI)





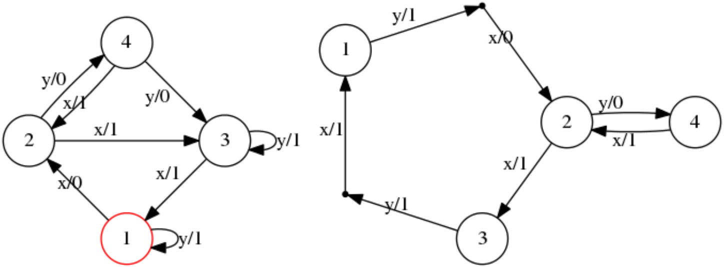
- We apply X_d again
  - This time, there is no point in reusing the suffix
  - $\omega = [1]y.1x.0[2]y.0x.1[2]y.0x.1[2]y.0[4]y.0x.1y.1$
  - The fragment $(2, y, 4)$ is verified
    - It is the transition $(2, y)$
  - As fragments $(2, yx, 2)$ and $(2, y, 4)$ are verified, so is $(4, x, 2)$
    - Another transition is verified: $(4, x)$
  - $\omega = [1]y.1x.0[2]y.0(4)x.1[2]y.0(4)x.1[2]y.0[4]y.0x.1y.1$

49 Local Optimization Method (VII)



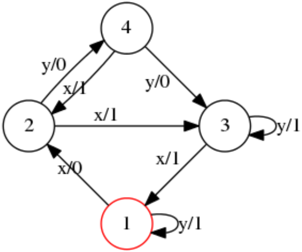
- We apply X_d again
  - $\omega = [1]y.1x.0[2] \ldots [4]y.0x.1[1]y.1x.0[2]$
    - Since $(1, yx, 2)$ is verified

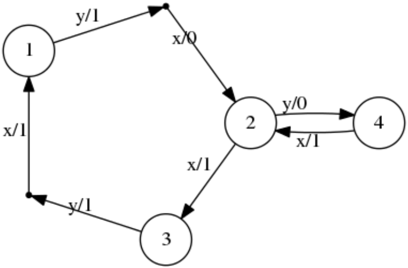
50 Local Optimization Method (VIII)



- There is an unverified transition in state 2, name $(2, x)$
  - Then, apply input x, followed by the X_d
  - $\omega = [1]y.1x.0[2]...[4]y.0x.1[1]y.1x.0[2]x.1[3]y.1x.1$
  - The fragment $(2, x, 3)$ is verified
    - Transition $(2, x)$ is verified

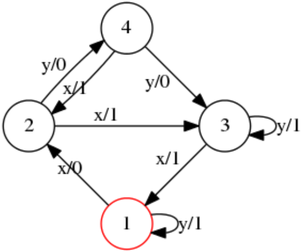
51 Local Optimization Method (IX)





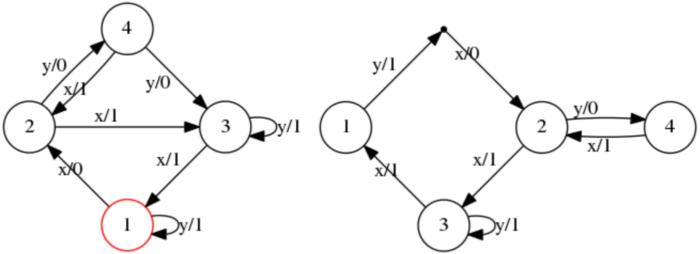
- We apply X_d
  - $\omega = [1]y.1x.0[2]\ldots[2]x.1[3]y.1x.1[1]y.1x.0[2]$
    - Since $(1, yx, 2)$ is verified
  - The fragment $(3, yx, 1)$ is verified

52 Local Optimization Method (X)



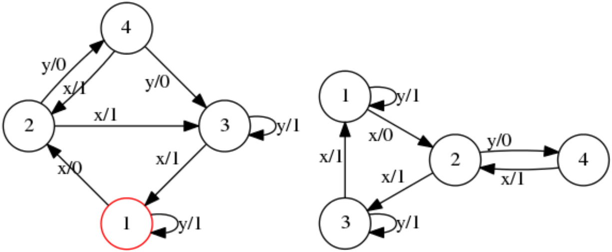
- There is no unverified transition from state 2
  - Transfer to a state where there is
    - Using only verified transitions
  - $\omega = [1]y.1x.0[2]\ldots[1]y.1x.0[2]x.1[3]$
    - Since $(2, x, 3)$ is verified

53 Local Optimization Method (XI)



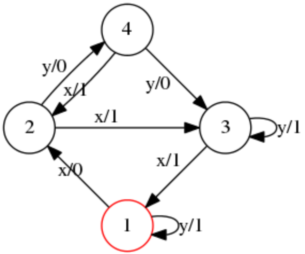
- Verify transitions either $(3, x)$ or $(3, y)$
  - Let us choose $(3, y)$
    - Apply y, then X_d
- $\omega = [1]y.1x.0[2] \ldots [1]y.1x.0[2]x.1[3]y.1[3]y.1(3)x.1[1]$
  - The fragment $(3, y, 3)$ is verified
  - So are transitions $(3, y)$ and $(3, x)$

54 Local Optimization Method (XII)



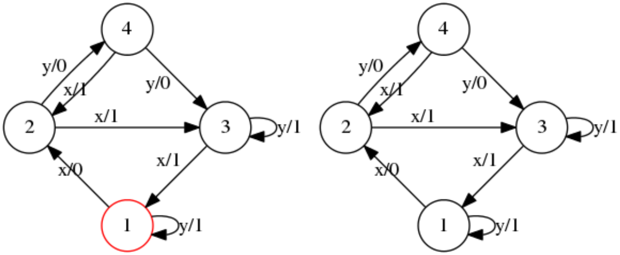
- Verify transitions either $(1, x)$ or $(1, y)$
  - Let us choose $(1, y)$
    - Apply y, then X_d
- $\omega = [1]y.1x.0[2]...[3]y.1[3]x.1[1]y.1[1]y.1(1)x.0[2]$

55 Local Optimization Method (XIII)



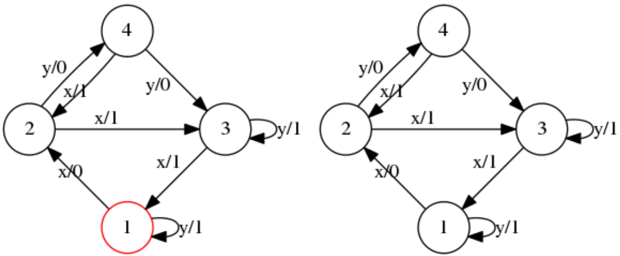
- There is no unverified transition from state 2
    - Transfer to a state where there is
    - $\omega = [1]y.1x.0[2]...[1]y.1(1)x.0[2]y.0[4]$
        - Since $(2, y, 4)$ is verified

56 Local Optimization Method (XIV)



- Verify transitions $(4, y)$
  - Apply y, then X_d
- $\omega = [1]y.1x.0[2]...[2]y.0[4]y.0[3]y.1x.1$

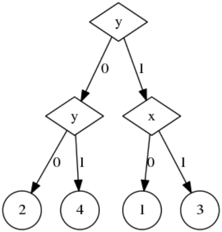
57 Local Optimization Method (XV)



- $\omega = yxyxyxyyxyxxyxyxxyyxyyxyyyx$
  - Length 27

58 Distinguishing Set

- One sequence for each state
- For each pair of states, there exists a common prefix of both corresponding sequences which separates them
  - $X_d^1 = y.1x.0$
  - $X_d^2 = y.0y.0$
  - $X_d^3 = y.1x.1$
  - $X_d^4 = y.0y.1$
- Adaptive Distinguishing Sequence



59 Distinguishing Set (II)

- A shorter checking sequence (in some cases)
- In the running example
  - $\omega = yxyyyxyxxyxyyxxyyxxyxyy$
  - Length 23

60 Without Distinguishing Sequence

- A characterization set[15]
- A set of sequences
  - For each pair of states, there exists a sequence which separates them
  - Always available for minimal machines

[15] T. S. Chow. "Testing Software Design Modeled by Finite-State-Machines". In: *IEEE Transactions on Software Engineering* 4.3 (May 1978), pp. 178–186.

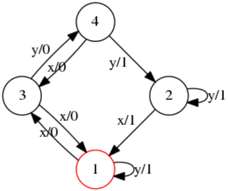
61 Without Distinguishing Sequence (II)

- The set of sequence should be applied to same state of the implementation
  - Signature
- How to return to the same state in the implementation?
  - Locating sequence[16],[17]

[16] F. C. Hennie. "Fault-detecting experiments for sequential circuits". In: *Proceedings of Fifth Annual Symposium on Circuit Theory and Logical Design*. 1965, pp. 95–110.

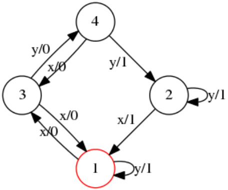
[17] Ali Rezaki and Hasan Ural. "Construction of checking sequences based on characterization sets". In: *Computer Communications* 18.12 (1995), pp. 911–920.

62 Without Distinguishing Sequence (III)



- Characterization set
  - $W = \{x, yx\}$
- Suppose we are at state s we suspect to be 2
  - Apply yx observing 11
    - It can be state 4 instead
- How to come back to s to apply x?

63 Without Distinguishing Sequence (IV)



- Repeating *yx* many times
  - We do the following
    - $L_2 = s_1\ yx\ xyy\ s_2\ yx\ xyy\ s_3\ yx\ xyy\ s_4\ yx\ xyy\ s_5\ yx\ xyy\ s_6\ x$
    - *yxxyy* cycles from state 2 back to state 2, in the specification

64 Without Distinguishing Sequence (V)

- $L_2 = s_1\ yx\ xyy\ s_2\ yx\ xyy\ s_3\ yx\ xyy\ s_4\ yx\ xyy\ s_5\ yx\ xyy\ s_6\ x$
  - As there are 4 states
    - Two of the states in the set $\{s_1, s_2, s_3, s_4, s_5\}$ should be the same
    - Then, s_6 should be one of the states $\{s_1, s_2, s_3, s_4\}$ for which we know the answer for yx
    - We then can infer which state it is, from the answers for yx and x

65 Without Distinguishing Sequence (VI)

- $L_2 = yxxyyyxxyyyxxyyyxyyyxxyy[2]x$
  - L_2 is a locating sequence for state 2
- $L_1 = yxxyxxyxxyxxyxx[1]x$
  - L_1 is a locating sequence for state 1
- $L_3 = yxyxyxyxyx[3]x$
  - L_3 is a locating sequence for state 3
- $L_4 = yxxyyxxyyxxyyxxyyxxy[4]x$
  - L_4 is a locating sequence for state 4

66 Without Distinguishing Sequence (VII)

- Apply all locating sequences
  - It should be done first
- Suppose we would like to check the end state after an input sequence α after state 4
- Apply $L_2 T_4 \alpha yx$ and $L_2 T_4 \alpha x$
  - T_4 transfer to state 4 in the specification

67 Sufficient Conditions

- Why the method work
  - A framework for proving correctness

68 Sufficient Conditions (II)

- Confirmed Sequences
  - When it is possible to be sure in which state the implementation is at
- Convergence (and Divergence)
  - When it is possible to be sure that two sequences reach the same state (or distinct states)

69 Sufficient Conditions (III)

- Consider the sequence
  - *xyyxyxxyxyxyxyyyx*
    - Length 17
- It is a checking sequence
  - It can be proved by using the sufficient conditions
    - Theorems and Lemmas in[18]

[18] Adenilso Simao and Alexandre Petrenko. “Checking Completeness of Tests for Finite State Machines”. In: *IEEE Transactions on Computers* 59 (2010), pp. 1023–1032.

70 Sufficient Conditions (IV)

- Adding the outputs
  - $?x0?y0?y0?x1?y1?x0?x1?y1?x1?y1?x0?y0?x1?y0?y0?y1?x1?$

71 Sufficient Conditions (IV)

- Adding the outputs
  - $?x0?y0?y0?x1?y1?x0?x1?y1?x1?y1?x0?y0?x1?y0?y0?y1?x1?$

72 Sufficient Conditions (IV)

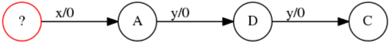
- Adding the outputs
  - ?x0?y0?y0?x1?y1?x0?x1?y1?x1?y1?x0?y0?x1?y0?y0?y1?x1?
- Identifying 4 states which cannot be the same in any implementation passing the test
  - ?x0?y0?y0?x1?y1?x0?x1?y1?x1?y1?x0?y0?x1?y0?y0?y1?x1?
    - Finding an n-clique in an n-partite graph!
    - NP-Complete

73 Sufficient Conditions (V)

- Marking them
  - $?x0Ay0?y0?x1By1?x0?x1Cy1?x1?y1?x0?y0?x1?y0Dy0?y1?x1?$

74 Sufficient Conditions (VI)

- Finding states which cannot be three of them
  - ?x0Ay0?y0?x1By1?x0?x1Cy1?x1?y1?x0?y0?x1?y0Dy0?y1?x1?
- Marking them with the only one it can be
  - ?x0Ay0?y0?x1By1?x0?x1Cy1?x1By1?x0?y0?x1?y0Dy0?y1?x1?
  - ?x0Ay0?y0?x1By1?x0?x1Cy1?x1By1?x0?y0?x1Ay0Dy0?y1?x1?
  - ?x0Ay0?y0?x1By1?x0?x1Cy1?x1By1?x0?y0?x1Ay0Dy0Cy1?x1?

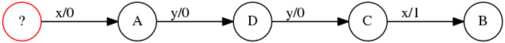


75 Sufficient Conditions (VII)

- How about this state?
  - $?x0Ay0?y0?x1By1?x0?x1Cy1?x1By1?x0?y0?x1Ay0Dy0Cy1?x1?$
    - Either A or D

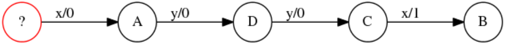
76 Sufficient Conditions (VIII)

- As the implementation is deterministic, if two points are the same state, the common suffixes lead to the same state
  - ?x0Ay0?y0?x1By1?x0?x1Cy1?x1By1?x0?y0?x1Ay0Dy0Cy1?x1?
  - ?x0Ay0Dy0Cx1By1?x0?x1Cy1?x1By1?x0?y0?x1Ay0Dy0Cy1?x1?
  - ?x0Ay0Dy0Cx1By1?x0?x1Cy1?x1By1?x0?y0?x1Ay0Dy0Cy1?x1?



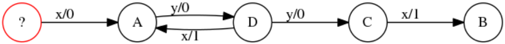
77 Sufficient Conditions (IX)

- We now that the previous state cannot be *D*
  - $?x0Ay0Dy0Cx1By1?x0?x1Cy1?x1By1?x0?y0?x1Ay0Dy0Cy1?x1?$
  - $?x0Ay0Dy0Cx1By1?x0?x1Cy1?x1By1?x0Ay0?x1Ay0Dy0Cy1?x1?$



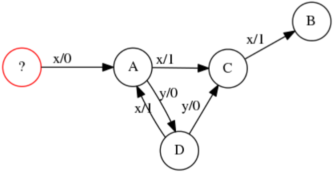
78 Sufficient Conditions (X)

- Common suffixes again
  - ?x0Ay0Dy0Cx1By1?x0?x1Cy1?x1By1?x0Ay0?x1Ay0Dy0Cy1?x1?
  - ?x0Ay0Dy0Cx1By1?x0?x1Cy1?x1By1?x0Ay0Dx1Ay0Dy0Cy1?x1?



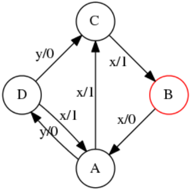
79 Sufficient Conditions (XI)

- Common suffixes again
  - ?x0Ay0Dy0Cx1By1?x0?x1Cy1?x1By1?x0Ay0Dx1Ay0Dy0Cy1?x1?
  - ?x0Ay0Dy0Cx1By1?x0Ax1Cy1?x1By1?x0Ay0Dx1Ay0Dy0Cy1?x1?



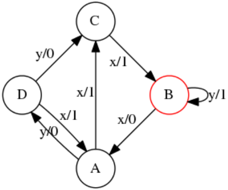
80 Sufficient Conditions (XII)

- We can infer the initial state now
  - ?x0Ay0Dy0Cx1By1?x0Ax1Cy1?x1By1?x0Ay0Dx1Ay0Dy0Cy1?x1?
  - Bx0Ay0Dy0Cx1By1?x0Ax1Cy1?x1By1?x0Ay0Dx1Ay0Dy0Cy1?x1?



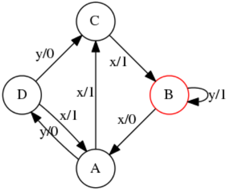
81 Sufficient Conditions (XIII)

- We can also infer other states
  - $Bx0Ay0Dy0Cx1By1?x0Ax1Cy1?x1By1?x0Ay0Dx1Ay0Dy0Cy1?x1?$
  - $Bx0Ay0Dy0Cx1By1Bx0Ax1Cy1?x1By1?x0Ay0Dx1Ay0Dy0Cy1?x1?$



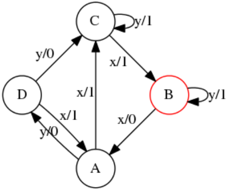
82 Sufficient Conditions (XIV)

- We can also infer other states
  - $Bx0Ay0Dy0Cx1By1Bx0Ax1Cy1?x1By1?x0Ay0Dx1Ay0Dy0Cy1?x1?$
  - $Bx0Ay0Dy0Cx1By1Bx0Ax1Cy1?x1By1Bx0Ay0Dx1Ay0Dy0Cy1?x1?$



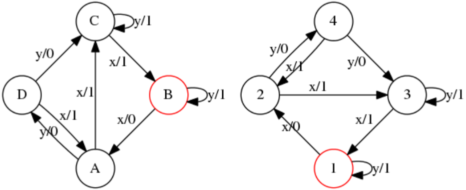
83 Sufficient Conditions (XV)

- We can also infer other states
  - $Bx0Ay0Dy0Cx1By1Bx0Ax1Cy1?x1By1Bx0Ay0Dx1Ay0Dy0Cy1?x1?$
  - $Bx0Ay0Dy0Cx1By1Bx0Ax1Cy1Cx1By1Bx0Ay0Dx1Ay0Dy0Cy1?x1?$



84 Sufficient Conditions (XVI)

- All transitions are inferred
  - $BxOAy0Dy0Cx1By1Bx0Ax1Cy1Cx1By1Bx0Ay0Dx1Ay0Dy0Cy1?x1?$



85 Sufficient Conditions (XVII)

- Thus, it is a checking sequence
  - *xyyxyxxyxyxyxyyyx*
    - Length 17
- It is very close to the shortest possible
  - *xyxyyyyyxyxxxx*
    - Length 14

86 Related work

- Extensions to the RCPP approach
  - Using adaptive distinguishing sequences[19]
  - Avoiding verifying some transitions[20]
  - Allowing for overlapping[21]

[19] Robert M. Hierons et al. "Using adaptive distinguishing sequences in checking sequence constructions". In: *Proceedings of the 2008 ACM Symposium on Applied Computing (SAC), Fortaleza, Ceara, Brazil, March 16-20, 2008*. 2008, pp. 682–687.

[20] Jessica Chen et al. "Eliminating Redundant Tests in a Checking Sequence". In: *Testing of Communicating Systems, 17th IFIP TC6/WG 6.1 International Conference, TestCom 2005, Montreal, Canada, May 31 - June 2, 2005, Proceedings*. 2005, pp. 146–158.

[21] Hasan Ural and Fan Zhang. "Reducing the Lengths of Checking Sequences by Overlapping". In: *Testing of Communicating Systems, 18th IFIP TC6/WG6.1 International Conference, TestCom 2006, New York, NY, USA, May 16-18, 2006, Proceedings*. 2006, pp. 274–288.

87 Related work (II)

- Extension to greedy approach
  - Using UIOs[22]
  - Dealing with non-deterministic machines[23]

[22] Adenilso Simao and Alexandre Petrenko. "Checking Sequence Generation Using State Distinguishing Subsequences". In: *Second International Conference on Software Testing Verification and Validation, ICST 2009, Denver, Colorado, USA, April 1-4, 2009, Workshops Proceedings*. 2009, pp. 48–56.

[23] Alexandre Petrenko, Adenilso Simao, and Nina Yevtushenko. "Generating Checking Sequences for Nondeterministic Finite State Machines". In: *Fifth IEEE International Conference on Software Testing, Verification and Validation, ICST 2012, Montreal, QC, Canada, April 17-21, 2012*. 2012, pp. 310–319.

88 Related work (III)

- Recent work
  - Generating good adaptive distinguishing sequences[24]
  - Removing (some) repetition in Locating Sequences[25]
  - Combining several distinguishing sequences[26]

---

[24] Uraz Cengiz Türker, Tonguç Ünlüyurt, and Hüsnü Yenigün. "Effective algorithms for constructing minimum cost adaptive distinguishing sequences". In: *Information & Software Technology* 74 (2016), pp. 69–85.

[25] Guy-Vincent Jourdan, Hasan Ural, and Hüsnü Yenigün. "Reducing locating sequences for testing from finite state machines". In: *Proceedings of the 31st Annual ACM Symposium on Applied Computing, Pisa, Italy, April 4-8, 2016*. 2016, pp. 1654–1659.

[26] Canan Güniçen, Guy-Vincent Jourdan, and Hüsnü Yenigün. "Using Multiple Adaptive Distinguishing Sequences for Checking Sequence Generation". In: *Testing Software and Systems - 27th IFIP WG 6.1 International Conference, ICTSS 2015, Sharjah and Dubai, United Arab Emirates, November 23-25, 2015, Proceedings*. 2015, pp. 19–34.

89 Concluding Remarks

- Checking sequence
  - When reseting is not an option
- Long tradition
  - Old, but gold
- Not rocket science
- Still active

90 Concluding Remarks (II)

- Thank you! ☺

Generating Checking Sequences: When Reseting is not an Option

Adenilso Simão - USP - Brazil
adenilso@icmc.usp.br

Departamento de Sistemas de Computação
Instituto de Ciências Matemáticas e de Computação
Universidade de São Paulo

TAROT - Paris - France - 2016-07-04