Symbolic Test Generation Using a Temporal Logic with Constrained Events

PENG WU[a], DAGUANG LIU[b,c], AND HUIMIN LIN[b]

[a]CNRS & LIX, Ecole Polytechnique, 91128 Palaiseau CEDEX, France

[b]Lab of Computer Sciences, Institute of Software, Chinese Academy of Sciences, Beijing 100080, China

[c]Graduate School, Chinese Academy of Sciences, Beijing 100049, China

ABSTRACT

1. TLCE - A Temporal Logic with Constrained Events
   a) temporal relationships among input and output (I/O) events
   b) data dependencies between event parameters
2. TLCE-based symbolic test generation algorithm
3. Case Study: Conference Protocol
4. Case Study: Cache Coherence Protocol

1 Introduction

1.1 Model-Based Testing

- Automated generation of efficient test cases using *system models* and *specified properties*
- Particularly useful for testing of concurrent systems (more difficult to find oracles)

1.2 STG - Symbolic Test Generation

- [1, 2, 3] - "restricted to the control structure" - the data dependencies between event parameters are ignored, which would result in less precise test cases
- improved through an approximate analysis on reachability/co-reachability [4]
- But in practice, it is still tedious and error-prone to represent test purposes as transition systems

Example - Bluetooth Service Discovery Protocol (SDP)

[TP] - *After a Partial Service Response message with a Continuation State parameter sent to the SUT, a subsequent Service Request will be received with the same Continuation State parameter.*
In this case, all partial response messages should use the same session identifiers as specified in preceding request messages.

1.3 Our Approach [5]

System Models *Symbolic Transition Graphs* - extended FSM with state variables and I/O actions

Property Specification *Temporal Logic with Constrained Events (TLCE)* - formulas as test purposes

Test Case Generation *slicing* the model with respect to formulas

Main Advantage:

- Can specify **both** synchronization behaviors and input-output data-dependency
- Can release the effort in constructing transition systems for test purposes
- Can support for a wider range of temporal properties to be adopted for automated test generation

1.4 Not Quite Clear

- PSCL takes a CTL (Computation Tree Logic) [6] like syntax, but the semantic interpretation of the temporal operator X ("next") in PSCL depends on the preceding path quantifier

1.5 Main Contribution

- TLCE - an extension of PSCL with model operators $\langle\rangle$ and $[]$ [7]
- TLCE-based symbolic test generation algorithm

2 TLCE

EVENT β	::=	$a?\bar{z}$	input event
		$\mid a!\bar{z}$	output event
PATH FORMULA η	::=	$\mathbf{G}\varphi$	always
		$\mid \varphi\mathbf{U}\varphi$	until
STATE FORMULA φ	::=	tt	true
		$\mid \neg\,\varphi$	negation
		$\mid \varphi \wedge \varphi$	conjunction
		$\mid \mathbf{E}\eta$	some path
		$\mid \mathbf{A}\eta$	every path
		$\mid \langle\beta : b\rangle\varphi$	some next state
		$\mid [\beta : b]\varphi$	every next state

- Syntactic Categories
  - Let Val be a set of values, ranged over by v
  - Let Var be a set of variables, ranged over by x, y, z
  - b is a boolean expression over $Val \cup Var$
  - a is a channel name
- The parameters of an output event are *variables*, not expressions
  - one can only obtain the output data without prior knowledge of how it was computed by the system under test (black-box testing)
  - the output data could be referred to in subsequent data dependency constraints

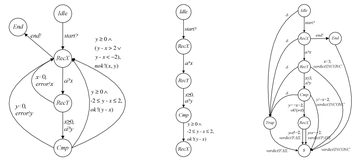
Example [TP] - $\mathbf{AG}([sdpRsp?(sid_1, cs_1) : cs_1 \neq NULL]\mathbf{A}(tt\ \mathbf{U}\langle sdpReq!(sid_2, cs_2) : cs_2 = cs_1\rangle tt))$

3 TLCE-based Symbolic Test Generation

Given a system model $G(N)$ and a test purpose φ in TLCE, the workflow of our approach is as follows:

1. Make the system model deterministic by eliminating internal transitions and resolving conflict I/O transitions
2. Slice the resulting model according to the temporal relationship specified in φ
3. Customize the resulting model slice as a symbolic test case according to the data dependency constraints specified in φ

Example The test purpose represented as a transition system in Fig. 2 of [4] can be directly described as a TLCE formula $\mathbf{EF}(\langle a?x : x \geq 3\rangle\mathbf{EF}(\langle ok!p : p == 2\rangle tt))$, where $\mathbf{EF}(\varphi) = \mathbf{E}(tt\ \mathbf{U}\ \varphi)$.

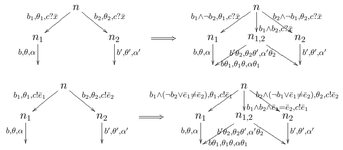


3.1 Determinization

a) Collecting Observable Transitions

Hypothesis - the system model does not contain a cycle of internal transitions.

b) Eliminating Conflict Transitions



Herein, even with conflict guards on the same output actions, the target nodes can still be distinguished if the output results are different. Therefore, the conflict condition above is $b_1 \wedge b_2 \wedge \bar{e}_1 = \bar{e}_2$.

3.2 TLCE-Based Slicing

A model slice consists of *potential* witnesses supporting the given test purpose. The *characteristic vector* $CV(\varphi)$ in the form of $(c_1, \cdots, c_{|N|})^T$ is defined to compute out recursively the model slice with respect to φ. Let $CV_i(\varphi)$ be the i-th element of $CV(\varphi)$, i.e. c_i. For $n_i \in N$, $n_i \vdash \varphi$ if $CV_i(\varphi) = 1$.

φ	$CV(\varphi)$	
ff	$c_i = 0, 1 \leq i \leq \lvert N\rvert$	
tt	$c_i = 1, 1 \leq i \leq \lvert N\rvert$	
$\psi_1 \wedge \psi_2$	$CV(\psi_1) * CV(\psi_2)$	
$\psi_1 \vee \psi_2$	$CV(\psi_1) + CV(\psi_2)$	
$\langle\beta : b\rangle\varphi$	$CM(\beta) \times CV(\varphi)$	
$[\beta : b]\varphi$	$CM(\beta) \circ CV(\varphi)$	
φ	$CV(\varphi) = \nu V.\ V + g(V)$	
	V_0	$g(V)$
$\mathbf{A}[\psi_1\ \mathbf{U}\ \psi_2]$	$CV(\psi_2)$	$CV(\psi_1) * (CM(G) \circledcirc V)$
$\mathbf{E}[\psi_1\ \mathbf{U}\ \psi_2]$	$CV(\psi_2)$	$CV(\psi_1) * (CM(G) \otimes V)$
φ	$CV(\varphi) = \nu V.\ V * g(V)$	
	V_0	$g(V)$
$\mathbf{A}[\psi_1\ \mathbf{R}\ \psi_2]$	$CV(\psi_2)$	$CV(\psi_1) + (CM(G) \circledcirc V)$
$\mathbf{E}[\psi_1\ \mathbf{R}\ \psi_2]$	$CV(\psi_2)$	$CV(\psi_1) + (CM(G) \otimes V)$
$\mathbf{A}[\mathbf{G}\ \psi]$	$CV(\psi)$	$CM(G) \circ V$
$\mathbf{E}[\mathbf{G}\ \psi]$	$CV(\psi)$	$CM(G) \times V$

3.3 Customizing Model Slices

Let $\bar{z}_\beta$ and $\bar{v}_{\alpha_\beta}$ denote the parameters of the event β and the corresponding action α_β, respectively,

1. For each $n \in N_\varphi$ and $n \vdash \langle\beta : b\rangle tt$, two outgoing edges, $(\neg\ b[\bar{v}_{\alpha_\beta}/\bar{z}_\beta], verdict!FAIL)$ and $(b[\bar{v}_{\alpha_\beta}/\bar{z}_\beta], verdict!PASS)$ are attached to every immediate α_β-derivative of n in N_φ with \$ as end nodes; and append a trap edge $n \xrightarrow{\delta} trap$;
2. For each $n \in N_\varphi$ and $n \vdash [\beta : b]\mathit{ff}$, two outgoing edges, $(\neg\ b[\bar{v}_{\alpha_\beta}/\bar{z}_\beta], verdict!PASS)$ and $(b[\bar{v}_{\alpha_\beta}/\bar{z}_\beta], verdict!FAIL)$ are attached to every α_β-derivative of n in N_φ with \$ as end nodes; If n has no α_β-derivative, append a trap edge $n \xrightarrow{\alpha_\beta} trap$;
3. if $n \in N_\varphi$ and $n \vdash [\beta : b]\psi$ (or $\langle\beta : b\rangle\psi$) with $\varphi \not\equiv tt$ (or ff), then for every α_β-derivatives n' of n which satisfies φ, each outgoing edge (b_1, θ, α) of n' is substituted with $(b_1 \wedge b[\bar{v}_{\alpha_\beta}/\bar{z}_\beta], (\bar{z}_\beta := \bar{v}_{\alpha_\beta})\theta, \alpha)$; while one outgoing edge $(\neg\ b[\bar{v}_{\alpha_\beta}/\bar{z}_\beta], verdict!INCONC)$ is attached to n' with \$ as the end node; for any non α_β-derivatives m of n, append $m \xrightarrow{verdict!INCONC.}$ \$.
4. If $trap \in N_\tau$, append $trap \xrightarrow{verdict!FAIL.}$ \$.

4 Case Studies

We have implemented the TLCE-based symbolic test generation algorithm in Object Caml. The TorX tool environment [8] was used as the execution engine for test campaigns, which proceeded as follows:

1. Generate symbolic test cases with regards to given test purposes
2. Derive parameterized test sequences, associated with specified data dependency constraints
3. Instantiate the input and output parameters, using a constraint solver Bonus [9], to produce executable test cases
4. Convert executable test cases as test specifications for TorX
5. Perform random testing in TorX within the fixed number of steps

4.1 Conference Protocol [10]

The protocol provides a multicast chatbox service to its user. Each user can initiate a conference, which users can join and leave at any time. Each user in a conference can exchange messages with all other users in the conference.

System Model 3 conference users, 22 nodes and 189 reachable transitions

Property Specifications 14 test purposes

Test Case Generation 287 test cases, full state and transition coverage

Mutants 15 erroneous mutants of the protocol, distributed within the TorX example package

Test Result TorX is configured to run automatically no more than 35 steps.

No.	P	F	I	FD	S	TorX Result	TorX S
14	105	182	0	63.00%	25	FAIL	13
16	69	218	0	75.00%	23	FAIL	27
17	0	211	76	73.00%	2	FAIL	2
18	91	196	0	68.00%	25	FAIL	15
19	210	77	0	26.00%	20	FAIL	19
20	69	218	0	75.00%	23	FAIL	27
22	28	259	0	90.00%	10	FAIL	27
23	219	68	0	23.00%	19	FAIL	17
24	203	84	0	29.00%	20	FAIL	14
27	22	265	0	92.00%	7	FAIL	23
28	221	66	0	22.00%	21	FAIL	18
31	209	78	0	27.00%	20	FAIL	15
32	205	82	0	28.00%	20	**PASS**	-
33	208	79	0	27.00%	20	FAIL	24
60	22	265	0	92.00%	7	FAIL	13
MS	100%					93%	

- Better Mutation Score (MS) (100% vs. 93%)
  - Mutant 32 can not be detected by the TorX test specification within 35 steps
- FD - fault detection ratio, i.e. the ratio between the number of failed test cases and the total number of test cases [11]
- No effort is required to design test cases for particular mutants

4.2 Cache Coherence Protocol [12]

The protocol aims to maintain the cache coherence among multi-processors with shared memory.

System Model 3 processes, based on the formal description of the protocol presented in [13]

Property Specification 6 test purposes

Test Case Generation 730 test cases, full state and transition coverage

Mutants 14 erroneous mutants

Test Result The column *TorX* shows the test results with 300 random test cases generated by TorX.

No.	FD(%)	S	TorX FD(%)	TorX S
1	64.79	11.39	99.50	9.42
2	64.79	11.39	97.00	10.10
3	13.42	8.41	-	-
4	22.19	5.08	-	-
5	0.14	2.00	-	-
6	37.53	11.96	40.50	19.98
7	0.27	4.50	14.00	19.54
8	6.03	12.55	-	-
9	10.82	14.19	39.50	18.63
10	1.23	7.44	19.50	16.95
11	20.00	29.30	-	-
12	0.68	19.10	30.00	21.53
13	38.36	4.00	-	-
14	0.96	14.92	29.50	19.97

- Better Mutation Score (100% vs. 57.14%)
- Most mutants can be detected by the symbolic test cases within a fairly small number of steps.
- For those mutants that can be detected by both methods, the FDs of the symbolic test cases are less than those of the random test cases.
  - conforms to the observation stated in [4] on accurate test cases
  - A random test case, which is not correspondent to any test purpose, would drive the system under test to exhibit non-conformance more possibly
  - A symbolic test case, which reflects accurately the guiding test purpose, would show less capability of detecting errors that are not related to that test purpose

5 Conclusions

- TLCE - a better way to express test purposes involving data dependencies between event parameters
- Case studies have shown the effectiveness and efficiency of our approach

Future Work

- Integration with the symbolic testing framework [4]
  - Symbolic test execution engine

References

[1] Rusu, V., du Bousquet, L., Jéron, T.: An approach to symbolic test generation. In the Proceedings of the 2nd International Conference on Integrated Formal Methods (IFM 2000), Schloss Dagstuhl, Germany, November 1–3, 2000, pp. 338–357

[2] Clarke, D., Jéron, T., Rusu, V., Zinovieva, E.: STG: A symbolic test generation tool. In the Proceedings of the 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2002), Grenoble, France, April 6–14, 2002, pp. 470–475

[3] Rusu, V., Marchand, H., Jéron, T.: Verification and symbolic test generation for safety properties. Technical Report 5285, INRIA (2004)

[4] Jeannet, B., Jéron, T., Rusu, V., Zinovieva, E.: Symbolic test selection based on approximate analysis. In the Proceedings of the 11th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2005), Edinburgh, UK, April 4-8, 2005, pp. 349–364

[5] Wu, P., Lin, H. M.: Model-based testing of concurrent programs with predicate sequencing constraints. In the Proceedings of the 5th International Conference on Quality Software (QSIC 2005), Melbourne, Australia, September 19–20, 2005, pp. 3–10. A journal version of this paper is published in the International Journal of Software Engineering and Knowledge Engineering, **16**(5):727–746, 2006.

[6] Clarke, E. M., Grumberg, O., Peled, D. A.: Model Checking. The MIT Press (1999)

[7] Stirling, C.: Modal and temporal properties of processes. Springer-Verlag New York, Inc., New York, NY, USA (2001)

[8] Tretmans, J., Brinksma, E.: Côte de Resyste – Automated Model Based Testing. In the Proceedings of Progress 2002 – 3rd Workshop on Embedded Systems, Utrecht, The Netherlands, October 24, 2002, pp. 246–255

[9] Zhang, J.: Specification analysis and test data generation by solving boolean combination of numeric constraints. In the Proceedings of the 1st Asia-Pacific Conference on Quality Software (APAQS 2000), Hong Kong, China, October 30–31, 2000, pp. 267–274.

[10] Belinfante, A., Feenstra, J., de Vries, R. G., Tretmans, J., Goga, N., Feijs, L., Mauw, S. and Heerink, L.: Formal test automation: A simple experiment. In the Proceedings of the 12th International Workshop on Testing of Communicating Systems (IWTCS 1999), Budapest, Hungary, September 1–3, 1999, pp. 179–196.

[11] Wu, P., Shi, X. C., Tang, J. J., Lin, H. M., Chen, T. Y.: Metamorphic testing and special case testing: A case study. Journal of Software **16**(7) (2005) 1210–1220

[12] German, S. M., Janssen, G.: Tutorial on verification of distributed cache memory protocols. In the Proceedings of the 5th International Conference of Formal Methods in Computer-Aided Design (FMCAD 2004), Austin, Texas, USA, November 14–17, 2004

[13] Pan, H., Lin, H. M., Lv, Y.: Model checking data consistency for cache coherence protocols. Journal of Computer Science and Technology **21**(5) (2006) 765–775

A Constraint Oriented Slicing Algorithm

During the iterative computation of $CV(\varphi)$, the program model is sliced simultaneously. The following illustrates the slicing algorithm based on the fixed point definitions of TLCE(PSCL) constraints.

function lfp-slicing(V_0, g, N, E) =
  $V = V_0$; $DV = g(V)$; $V' = V + DV$;
  $N = \{n_k \mid DV(k) = 1\} \cup N$;
  $E = \{n_k \xrightarrow{b,\theta,\alpha} n_l \mid DV(k) = 1, V(l) = 1\} \cup E$;
  while $V' \neq V$ **do**
    $DV = g(V)$; $V' = V + DV$;
    $N = \{n_k \mid DV(k) = 1\} \cup N$;
    $E = \{n_k \xrightarrow{b,\theta,\alpha} n_l \mid DV(k) = 1, V(l) = 1\} \cup E$;
  end while;
  return (V, N, E)
end function

function gfp-slicing(V_0, g, N, E) =
  $V = V_0$; $DV = g(V)$; $V' = V * DV$;
  $N = \{n_k \mid DV(k) = 1\} \cup N$;
  $E = \{n_k \xrightarrow{b,\theta,\alpha} n_l \mid DV(k) = 1, V(l) = 1\} \cup E$;
  while $V' \neq V$ **do**
    $DV = g(V)$; $V' = V * DV$;
    $N = \{n_k \mid DV(k) = 1\} \cup N$;
    $E = \{n_k \xrightarrow{b,\theta,\alpha} n_l \mid DV(k) = 1, V(l) = 1\} \cup E$;
  end while;
  return (V, N, E)
end function

THE 19^{th} IFIP INTERNATIONAL CONFERENCE ON TESTING OF COMMUNICATING SYSTEMS AND THE 7^{th} INTERNATIONAL WORKSHOP ON FORMAL APPROACHES TO TESTING OF SOFTWARE
POSTER SESSION, JUNE 28, 2007, TALLINN, ESTONIA