Appendices Used in the Formal Analysis of Software Library

Shahbaz Ali[1,2], Hailong Sun[1,2], and Yongwang Zhao[1,2]

[1] School of Computer Science and Engineering, Beihang Unviersity, Beijing, China
[2] Beijing Advanced Innovation Center for Big Data and Brain Computing Beihang University, Beijing, China
{bazgikian,sunhl,zhaoyw}@buaa.edu.cn

Appendix A

Learned Models

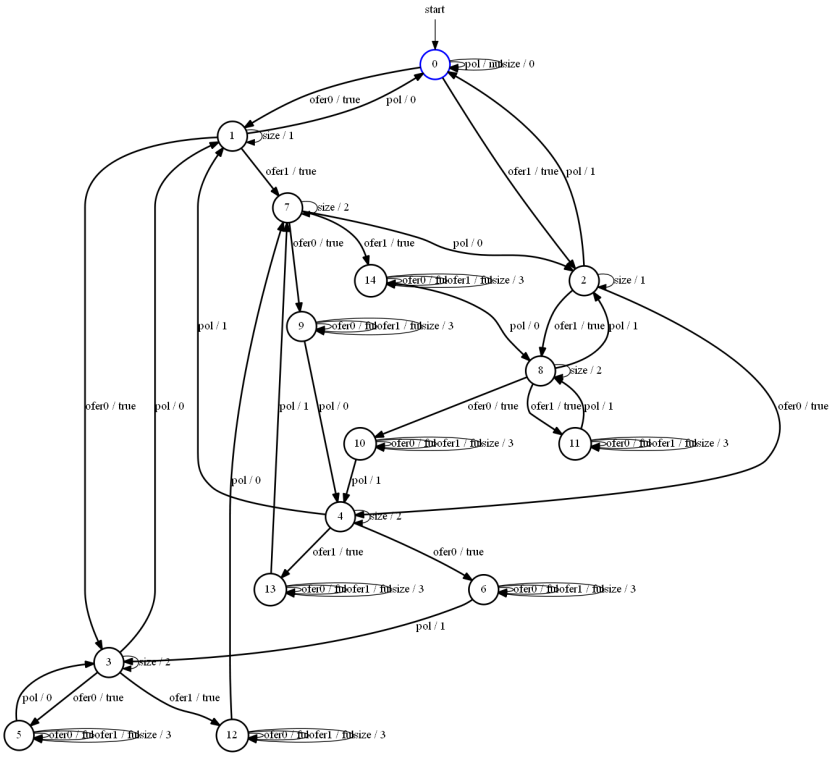

Fig. A.1: Learned Model of Queue Implementation (size=3)

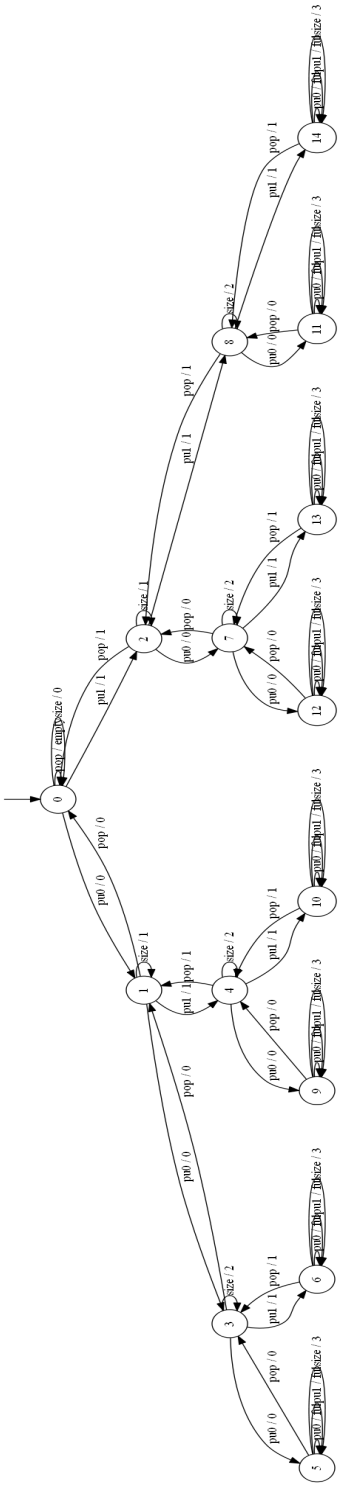

Fig. A.2: Learned Model of Stack Implementation (size=3)

Appendix B

Formal Specifications

Table B.1: Requirement Specifications for SET

Property	**Informal and Formal (LTL/CTL) Specifications**
DEFINE Set-Fixed-Size:= 2 to 10 DEFINE FullSet:= $((inp = add1 \mid inp = add2) \& out = ful)$ DEFINE EmptySet:= $(inp = size \& out = 0) \mid (inp = remov1 \& out = empty)$	
Safety Requirement Specifications	
P1	It is never the case that the "Set" indicates both "Empty" and "Full" simultaneously. $AG!(((inp = size \& out = 0) \mid ((inp = remov1 \mid inp = remov2) \& out = empty)) \& ((inp = add1 \& out = ful) \mid (inp = add2 \& out = ful) \mid (inp = size \& out = Set - Fixed - Size)))$
P2	It is never the case that Set-size becomes "greater than fixed-size" i.e., $(Set - Size > Set - Fixed - Size)$. $AG!(inp = size \& out = (Set - Fixed - Size) + 1)$.
P3	It is never the case that the "Set-size becomes less than zero". $AG!(inp = size \& out = -1)$ Or.
P4	It never happens that the "Set always remains Empty" (except at initial state). $AG!((state! = s0) \& (inp = size \& out = 0))$
P5	It never happens that the Set always remains Full (except at final state). $G!((inp = size \& out! = Set - Fixed - Size) \& FullSet)$ Or. $AG!((inp = size \& out! = Set - Fixed - Size) \& FullSet)$
P6	It should never be the case that the two operations "Add" and "Remove" occur at the same time. $G!((inp = add1 \& out = Tru) \& (inp = remov1 \& out = Tru))$ Or. $AG!(inp = add1 \& inp = remov1)$
P7	It should never be the case that the two operations "Add0" and "Add1" occur at the same time. $G!((inp = add1 \& out = Tru) \& (inp = add2 \& out = Tru))$ Or. $AG!(inp = add1 \& inp = add2)$

Continued on next page

Table B.1 – continued from previous page

Property	**Informal and Formal (LTL/CTL) Specifications**
P8	It never happens that if "Add/Remove" operation occurs then Set move in "Overflows" state. $AG!((inp = remov1 \mid inp = remov2) \& out = (Set - Fixed - Size) + 1)$ Or. $G!(inp = add1 \& out = (Set - Fixed - Size) + 1)$
P9	It never happens that if "Add/Remove" operation occurs then Set move in "Underflow" state. $G!(inp = add1 \& out = -1)$ Or. $AG!((inp = remov1 \mid inp = remov2) \& out = -1)$
Functional Requirement Specification	
P10	Set must not contain duplicated elements. $G(inp = add1 - > X(inp = add1 - > out! = Tru))$ Or $G(inp = add2 - > X(inp = add2 - > out! = Tru))$
P11	It should be 'Empty' only at initial state (not in every state). $G(state! = s0 - > !EmptySet)$ Or $G((inp = size \& out! = 0) - > !EmptySet)$
P12	It should be 'Filled' only at final state. (not in every state). $G((inp = size \& out! = Set - Fixed - Size) - > !FullSet)$
P13	"Add" operation must increase the Set-size. $G((inp = add1 \mid inp = add2) - > X(inp = size - > out! = 0))$ $G((inp = size \& out = 0) - > X(inp = add1 - > X(inp = add2 - > X(inp = size - > out = 2))))$
P14	"Remove" operation must decrease the Set-size. $G((inp = size \& out = Set - Fixed - Size) - > X(inp = remov1 - > X(inp = remov2 - > X(inp = size - > out = (Set - Fixed - Size) - 2))))$ Or $G((FullSet) - > X((inp = remov1 \mid inp = remov2) - > X((inp = remov1 \mid inp = remov2) - > X(inp = size - > out! = Set - Fixed - Size))))$
P15	Set should not "Overflow" or "Underflow". $G((!FullSet \& (inp = add1 \mid inp = add2)) - > X(inp = size - > out! = Set - Overflow))$ $G(!EmptySet \& inp = remov1 - > X(inp = size - > out! = Set - Underflow))$
P16	If equal number of "Add" operations followed by same number of "Remove" operations then Set-size should remain same. $G((inp = size \& out = 1) - > X(inp = add1 - > X(!FullSet \& inp = add2 - > X(inp = remov1 - > X(inp = remov1 - > X(inp = size - > out = 1))))))$ $G((FullSet) - > X(inp = add1 - > X(!FullSet \& inp = add2 - > X(inp = remov1 - > X(inp = remov2 - > X(FullSet))))))$

Continued on next page

Table B.1 – continued from previous page

Property	**Informal and Formal (LTL/CTL) Specifications**
P17	"Add1" operation must rightly inserts "1" (not any other number like 2) on the Set. And "Add2" must inserts "2" (not 1) on the Set. $G(!FullSet\&inp = add1 -> X(inp = remov1 -> out = 1))$ $G(!FullSet\&inp = add2 -> X(inp = remov2 -> out = 2))$
Liveness Requirement Specifications	
P18	If "Add" operation is done forever on a non-filled Set and "Remove" operation never done then the Set eventually becomes "Full". $G((!FullSet\&((inp = add1 \mid inp = add2)UFullSet)) -> X(inp = size -> out = Set - Fixed - Size))$ $G((!EmptySet\&(inp = add1 \mid inp = add2)UFullSet\&inp! = remov1) -> X(inp = size -> out = Set - Fixed - Size))$
P19	If "Remove" operation is done forever on non-empty Set and "Push" operation never done then the Set eventually becomes "Empty". $G(!FullSet -> (((inp = remov1 \mid inp = remov2)UEmptySet) -> F(inp = size -> out = 0)))$ $G(!EmptySet -> ((inp = remov1 \mid inp = remov2)UEmptySet) -> X(inp = size -> out = 0))$
P20	If "Add" operation wants to write on the Set then it eventually did this successfully without "Overflow". $AG((!FullSet\&(inp = add1 \mid inp = add2)) -> AF(out! = Set - Underflow \mid out! = Set - Overflow))$ $AG((inp = add1 \mid inp = add2) -> AF(out! = Set - Overflow))$
P21	If "Remove" operation wants to remove an element from the Set, then it eventually did this successfully without "Underflow". $AG(!EmptySet\&inp = remov1 -> AF(out! = Set - Underflow))$

Table B.2: Requirement Specifications for Queue

Property	**Informal and Formal (LTL/CTL) Specifications**
DEFINE Queue-Fixed-Size:= 3 to 16 DEFINE FullQueue:= $((inp = ofer0 \mid inp = ofer1)\&out = ful)$ DEFINE EmptyQueue:= $(inp = size\&out = 0) \mid (inp = pol\&out = null)$	
Safety Requirement Specifications	
P22	It is never the case that the "Queue" indicates both "Empty" and "Full" simultaneously. $AG!(((inp = size\&out = 0) \mid (inp = pol\&out = null))\&((inp = ofer0\&out = ful) \mid (inp = ofer1\&out = ful) \mid (inp = size\&out = Queue - Fixed - Size)))$

Continued on next page

Table B.2 – continued from previous page

Property	**Informal and Formal (LTL/CTL) Specifications**
P23	It is never the case that Set-size becomes "greater than fixed-size" i.e., $(Set - Size > Set - Fixed - Size)$. $AG!(inp = size \& out = (Set - Fixed - Size) + 1)$
P24	It is never the case that the "Que-size becomes less than zero". $AG!(inp = size \& out = -1)$
P25	It never happens that the "Queue always remains Empty" (except at initial state). $AG!((state! = s0) \& EmptyQueue)$
P26	It never happens that the "Queue always remains Full" (except at final state). $AG!((inp = size \& out! = Queue - Fixed - Size) \& FullQueue)$
P27	It should never be the case that the two operations "offer" and "pol" occur at the same time. $AG!((inp = ofer0 \& out = 0) \& (inp = pol \& out = 0))$
P28	It should never be the case that the two operations "ofer0" and "ofer1" occur at the same time. $AG!((inp = ofer0 \& out = 0) \& (inp = ofer1 \& out = 1))$
P29	It never happens that a "offer/pol" operation occurs but/and the Queue is in "Overflows" state. $AG!(inp = ofer0 \& out = Queue - Fixed - Size + 1)$ Or. $AG!(inp = pol \& out = Queue - Fixed - Size + 1)$
P30	It never happens that a "offer/pol" operation occurs but/and the Queue is in "Underflow" state. $AG!(inp = ofer0 \& out = -1)$ Or. $AG!(inp = pol \& out = -1)$
P31	It is never the case that the behavior of "offer/pol" operation is different in different state. $G!((inp = ofer0 \& out = 0) \& X(!FullQueue \& inp = ofer0 \& out = 1))$
	Functional Requirement Specification
P32	Queue should behaves like 'FIFO' manner(not like 'LIFO'). $G(!FullQueue - > (inp = ofer1 - > X(!FullQueue \& inp = ofer0 - > X(inp = pol - > out = 1))))$
P33	It should be 'Empty' only at initial state (not in every state). $G((inp = size \& out! = 0) - > !EmptyQueue)$
P34	It should not be 'Filled' everywhere except at final state. $G((inp = size \& out! = Queue - Fixed - Size) - > !FullQueue)$
P35	"Insert (offer" operation increases the Queue-size. $G((inp = ofer0 \| inp = ofer1) - > X(inp = size - > out! = 0))$ $G((inp = size \& out = 0) - > X(inp = ofer0 \| inp = ofer1 - > X(!FullQueue \& inp = ofer0 \| inp = ofer1 - > X(inp = size - > out = 2))))$

Continued on next page

Table B.2 – continued from previous page

Property	**Informal and Formal (LTL/CTL) Specifications**
P36	"Remove (pol)" operation decreases the Queue-size. $G((inp = size \& out = Queue - Fixed - Size) -> X(inp = pol -> X(inp = pol -> X(inp = size -> out = (Queue - Fixed - Size) - 2))))$
P37	Queue should not "Overflow" or "Underflow". $G((!FullQueue \& (inp = ofer0 \mid inp = ofer1)) -> X(inp = size -> out! = Queue - Overflow))$ $G(!EmptyQueue \& inp = pol -> X(inp = size -> out! = Queue - Underflow))$
P38	If equal number of "offer" operations followed by same number of "poll" operations occur then Queue-size should remain same. $G((inp = size \& out = 1) -> X(inp = ofer0 -> X(!FullQueue \& inp = ofer1 -> X(inp = pol -> X(inp = pol -> X(inp = size -> out = 1))))))$
P39	"Insert0 (ofer0)" operation must inserts "Underflow" (not 1) on the Queue. And "Insert1 (ofer1)" must inserts "Underflow" (not 0) on the Queue. $inp = ofer1 -> X(((inp = ofer0 \mid inp = ofer1) U !FullQueue) -> (inp = pol -> out = 1))$ $inp = ofer0 -> X(((inp = ofer0 \mid inp = ofer1) U !FullQueue) -> (inp = pol -> out = 0))$
Liveness Requirement Specifications	
P40	If "offer" operation is done forever on a non-filled Queue and "poll" operation never done then the Queue eventually becomes "Full". $G((!FullQueue \& ((inp = ofer0 \mid inp = ofer1) U FullQueue)) -> X(inp = size -> out = Queue - Fixed - Size))$ $G((!EmptyQueue \& (inp = ofer0 \mid inp = ofer1) U FullQueue) -> X(inp = size -> out = Queue - Fixed - Size))$
P41	If "pol" operation is done forever on non-empty Queue and "offer" operation never done then the Queue eventually becomes "Empty". $G(!FullQueue -> ((inp = pol U EmptyQueue) -> F(inp = size -> out = 0)))$ $G(!EmptyQueue -> ((inp! = ofer0 \mid inp! = ofer1) \& inp = pol U EmptyQueue) -> X(inp = size -> out = 0))$
P42	If "offer" operation wants to write on the Queue, then it eventually did this successfully without "Overflow". $AG((!FullQueue \& (inp = ofer0 \mid inp = ofer1)) -> AF(out! = Queue - Underflow \mid out! = Queue - Overflow))$
P43	If "pol" operation wants to remove an element from the Queue, then it eventually did this successfully without "Underflow". $AG(!EmptyQueue \& inp = pol -> AF(out! = Queue - Underflow))$

Table B.3: Requirement Specifications for Stack

Property	**Informal and Formal (LTL/CTL) Specifications**
DEFINE Stack-Fixed-Size:= 3 to 16 DEFINE FullStack:= $((inp = pu0 \mid inp = pu1) \& out = ful)$ DEFINE EmptyStack:= $(inp = size \& out = 0) \mid (inp = pul \& out = null)$	
Safety Requirement Specifications	
P44	It never happens that the Stack is "Full" and "Empty" simultaneously. $AG!(((inp = size \& out = 0) \mid (inp = pop \& out = empty)) \& ((inp = pu0 \& out = ful) \mid (inp = pu1 \& out = ful) \mid (inp = size \& out = Stack - Fixed - Size)))$
P45	It is never the case that Stack-size becomes "greater than fixed-size" i.e., $(Stack - Size > Stack - Fixed - Size)$. $AG!(inp = size \& out = (Stack - Fixed - Size) + 1)$
P46	It is never the case that the "Stack-size becomes less than zero". $AG!(inp = size \& out = -1)$
P47	It never happens that the "Stack always remains Empty" (except at initial state). $AG!((state! = s0) \& EmptyStack)$
P48	It never happens that the "Stack always remains Full" (except at final state). $AG!((inp = size \& out! = Stack - Fixed - Size) \& FullStack)$
P49	It should never be the case that the two operations "Push" and "Pop" occur at the same time. $AG!((inp = pu0 \& out = 0) \& (inp = pop \& out = 0))$
P50	It should never be the case that the two operations "Push0" and "Push1" occur at the same time. $AG!((inp = pu0 \& out = 0) \& (inp = pu1 \& out = 1))$
P51	It never happens that a "Push/Pop" operation occurs but/and the Stack is in "Overflows" state. $AG!(inp = pop \& out = Stack - Fixed - Size + 1)$ Or. $AG!(inp = pu0 \& out = Stack - Fixed - Size + 1)$
P52	It never happens that a "Push/Pop" operation occurs but/and the Stack is in "Underflow" state. $AG!((inp = pu0 \mid inp = pu1) \& out = -1)$ Or. $AG!(inp = pol \& out = -1)$
P53	It is never the case that the behavior of "Push/Pop" operation is different in different state. $G!((inp = pu0 \& out = 0) \& X(!FullStack \& inp = pu0 \& out = 1))$
Functional Requirement Specification	

Continued on next page

Table B.3 – continued from previous page

Property	**Informal and Formal (LTL/CTL) Specifications**
P54	Stack should behaves like "LIFO" manner not like "FIFO" manner. $G(!FullStack- > (inp = pu1- > X(!FullStack\&inp = pu0- > X(inp = pop- > out! = 1))))$
P55	It should be 'Empty' only at initial state (not in every state). $G((inp = size\&out! = 0)- >!EmptyStack)$
P56	It should not be 'Filled' everywhere except at final state. $G((inp = size\&out! = Stack - Fixed - Size)- >!FullStack)$
P57	"Push" operation increases the stack-size. $G((inp = size\&out = 1)- > X(inp = pu0 \mid inp = pu1- > X(!FullStack\&inp = pu0 \mid inp = pu1- > X(inp = size- > out = 3))))$
P58	"Pop" operation decreases the stack-size. $G((FullStack)- > X(inp = pop- > X(inp = pop- > X(inp = size- > out = (Stack - Fixed - Size) - 2))))$
P59	Stack should not "Overflow" or "Underflow". $G((!FullStack\&(inp = pu0 \mid inp = pu1))- > X(inp = size- > out! = Stack - Overflow))$ $G(!EmptyStack\&inp = pop- > X(inp = size- > out! = Stack - Underflow))$
P60	If equal number of "Push" operations followed by same number of "Pop" operations occur then stack-size should remain same. $G((inp = size\&out = 2)- > X(inp = pu0- > X(!FullStack\&inp = pu1- > X(inp = pop- > X(inp = pop- > X(inp = size- > out = 2))))))$
P61	"Push0" operation must inserts "Underflow" (not 1) on the Stack. And "Push1" must inserts "Underflow" (not 0) on the stack. $G(!FullStack\&inp = pu0- > X(inp = pop- > out = 0))$ $G(!FullStack\&inp = pu1- > X(inp = pop- > out = 1))$
Liveness Requirement Specifications	
P62	If "Push" operation is done forever on a non-filled stack and "Pop" operation never done then the Stack eventually becomes "Full". $G((!FullStack\&((inp = pu0 \mid inp = pu1)UFullStack))- > X(inp = size- > out = Stack - Fixed - Size))$ $G((!EmptyStack\&(inp = pu0 \mid inp = pu1)UFullStack)- > X(inp = size- > out = Stack - Fixed - Size))$
P63	If "Pop" operation is done forever on non-empty stack and "Push" operation never done then the Stack eventually becomes "Empty". $G(!FullStack- > ((inp = popUEmptyStack)- > (inp = size- > out = 0)))$ $G(!EmptyStack- > (inp = popUEmptyStack)- > F(inp = size- > out = 0))$

Continued on next page

Table B.3 – continued from previous page

Property	**Informal and Formal (LTL/CTL) Specifications**
P64	If "Push" operation wants to write on the Stack, then it eventually did this successfully without "Overflow". $AG((!FullStack\&(inp = pu0 \vert inp = pu1)) -> AF(out! = Stack - Underflow \vert out! = Stack - Overflow))$
P65	If "Pop" operation wants to remove an element from the Stack, then it eventually did this successfully without "Underflow". $AG(!EmptyStack\&inp = pop -> AF(out! = Stack - Underflow))$