LOGICAL MODELS OF SYSTEMS

Theory and Software



Wolfgang Schreiner <Wolfgang.Schreiner@risc.jku.at> Research Institute for Symbolic Computation (RISC) Johannes Kepler University, Linz, Austria





Logical Models

What is the purpose of logical modeling?

- Precisely describe the problem to be solved.
 - □ Clarification of mind, resolution of ambiguities.
 - □ Specification of program to be developed.
- Software-supported analysis of the problem and its solution.
 - □ Validation of specification.
 - □ Validation/verification of solution.
 - □ Interactive/automatic provers and model checkers.
- Automatic computation of solution respectively simulation of execution.
 - Logical solvers (SMT: Satisfiability Modulo Theories).
 - □ Perhaps: rapid prototyping of a later manually written program.

To profit from software, we need computer-understandable models.

1. Modeling Systems

2. The Temporal Logic of Actions (TLA)

Computational Systems

Programs are just special cases of "(computational) systems".

Computational System

- One or more active components.
- Deterministic or nondeterministic behavior.
- May or may not terminate.

Safety

- "Nothing bad will ever happen."
- Partial correctness of programs: for every admissible input, if the program terminates, its output does not violate the output condition.

Liveness

- □ "Something good will eventually happen."
- Termination of programs: for every input, the program eventually terminates.

General goal is to establish the safety and liveness of computational systems.

Transition Systems

Any computational system can be modeled as a transition system T = (S, I, R).

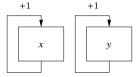
- State space $S = S_1 \times ... \times S_n$: the set of all possible system states.
 - Determined by the possible values of system variables x_1, \ldots, x_n with values from (finite or infinite) domains S_1, \ldots, S_n .
- Initial states $I \subseteq S$: the possible starts of the execution of the system.
 - □ Typically defined by an a predicate I_x on the system variables x_1, \ldots, x_n .
- **Transition relation** $R \subseteq S \times S$: the possible execution steps.
 - □ Typically defined by a predicate $R_{x,x'}$ between the prestate values x and the poststate values x' of the program variables.

Nondeterminism: for some prestate x there may be multiple poststates x'.

Example

System C = (S, I, R) with counters x und y which may be independently incremented.

$$S := \mathbb{Z} \times \mathbb{Z}$$
$$I(x, y) :\Leftrightarrow x = y \land y \ge 0$$
$$R(\langle x, y \rangle, \langle x', y' \rangle) :\Leftrightarrow$$
$$(x' = x + 1 \land y' = y) \lor$$
$$(x' = x \land y' = y + 1)$$



Infinitely many starting states.

$$[x = 0, y = 0], [x = 1, y = 1], [x = 2, y = 2], \dots$$

In each state two possibilities.

$$[x = 2, y = 3] \rightarrow [x = 3, y = 3]$$
$$\rightarrow [x = 2, y = 4]$$

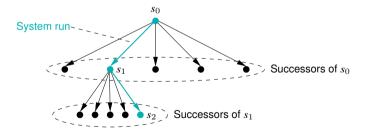
A nondeterministic system.

System Runs

Transition system T = (S, I, R).

System run: (finite or infinite) sequence $s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow \ldots$ of states in *S*.

- \square s₀ is initial: $I(s_0)$.
- \Box $s_i \rightarrow s_{i+1}$ ist a transition: $R(s_0, s_1)$.
- □ If run stops in s_n , then s_n has no successor: $\neg R(s_n, s')$, for all $s' \in S$.



System runs can be understood as paths in a directed graph.

System Properties

Properties of a transition system can be specified in linear temporal logic (LTL).

System S satisfies LTL formula P, if each possible run of S satisfies P.
 Action: A

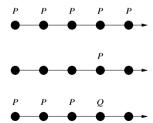
- □ Classical logic formulas with variables x, y, ... and x', y', ...
- First state pair (s_0, s_1) of run satisfies A with x, y, \ldots interpreted in s_0 and x', y', \ldots interpreted in s_1 .

■ Always: □P

- **Run** satisfies property *P* from every position i on.
- Eventually: ◇P
 - **Run** satisfies P from at least one position i on.

• Until: $P \cup Q$

□ Run satisfies property Q from at least one position i on; from all previous positions j < i it satisfies property P.



7/42

Example

System C = (S, I, R).

$$S := \mathbb{Z} \times \mathbb{Z}$$
$$I(x, y) :\Leftrightarrow x = y \land y \ge 0$$
$$R(\langle x, y \rangle, \langle x', y' \rangle) :\Leftrightarrow$$
$$(x' = x + 1 \land y' = y) \lor$$
$$(x' = x \land y' = y + 1)$$

Safety: $\Box(x \ge 0 \land y \ge 0)$

\square Both *x* and *y* never become negative.

System satisfies specification, because every run has this property.

Liveness: $\diamond x \ge 1$.

 \Box Variable *x* will eventually have a value greater equal 1.

System violates specification, because one run does not have this property:

 $[x=0, y=0] \rightarrow [x=0, y=1] \rightarrow [x=0, y=2] \rightarrow [x=0, y=3] \rightarrow \dots$

Liveness properties may be violated by unfair runs; we want to ignore such runs.

Verifying Safety

We only consider the verification of a safety property.

- $\blacksquare M \models \Box F.$
 - □ Verify that formula F is an invariant of system M.
- $\blacksquare M = (S, I, R).$
 - $\Box \ I(s):\Leftrightarrow\ldots$
 - $\square R(s,s') :\Leftrightarrow R_0(s,s') \lor R_1(s,s') \lor \ldots \lor R_{n-1}(s,s').$
- Proof by induction.
 - $\Box \quad \forall s. \ I(s) \Rightarrow F(s).$

. . .

- F holds in every initial state.
- $\Box \ \forall s, s'. \ F(s) \land R(s, s') \Longrightarrow F(s').$
 - Each transition preserves F.
 - Reduces to a number of subproofs:

$$F(s) \land R_0(s, s') \Rightarrow F(s')$$

 $F(s) \wedge R_{n-1}(s, s') \Longrightarrow F(s')$

Fairness

- Infinity: Infinite $P : \Leftrightarrow \Box \Diamond P$
 - For every position *i* there is a position $j \ge i$ at which property *P* holds.
 - □ Property *P* is satisfied infinitely often.
- **Stability:** Stable $P : \Leftrightarrow \Diamond \Box P$
 - □ There is a position *i* such that at all positions $j \ge i$ property *P* holds.
 - Property P eventually permanently holds.
- Executability: Enabled A
 - □ Action *A* describes a transition that is executable in the current state *s*: there is a state *s'* with R(s, s') such that A(s, s').
- Weak Fairness: WF $A : \Leftrightarrow$ Stable (Enabled $A) \Rightarrow$ Infinite A
 - □ If *A* is eventually permanently enabled, then *A* will (infinitely often) be executed.
- **Strong Fairness:** SF $A : \Leftrightarrow$ Infinite (Enabled $A) \Rightarrow$ Infinite A
 - If A is infinitely often enabled, then A will (infinitely often) be executed. 10/42

Example

System C = (S, I, R). $S := \mathbb{Z} \times \mathbb{Z}$ $I(x, y) :\Leftrightarrow x = y \land y \ge 0$ $R(\langle x, y \rangle, \langle x', y' \rangle) :\Leftrightarrow$ $(x' = x + 1 \land y' = y) \lor$ $(x' = x \land y' = y + 1)$

Liveness under the Assumption of Weak Fairness:

$$(\mathsf{WF} \ x' = x + 1 \land y' = y) \Rightarrow \diamondsuit x \ge 1$$

□ If first action is eventually permanently enabled, it is infinitely often executed.

- □ The action is always enabled (Enabled $x' = x + 1 \land y' = y \equiv true$).
- □ Thus it is infinitely often executed such that eventually $x \ge 1$ holds ($\Diamond x \ge 1$).

The process scheduler must implement the required fairness properties.

1. Modeling Systems

2. The Temporal Logic of Actions (TLA)

The Temporal Logic of Actions (TLA)

Leslie Lamport (Microsoft Research since 2001).

- □ ACM Turing Award 2013.
- **TLA** model of a system:

 $I_x \wedge \Box[R]_x \wedge \mathsf{WF}_x(A) \wedge \ldots$

- □ Initial condition I_x .
- Transition relation $[R]_x$:
 - $[R]_x \equiv (R \lor x = x')$
 - x = x': stutter step (nothing changes).
- Fairness conditions:
 - Conjunction of formulas $WF_x(A)$ and/or $SF_x(A)$ for actions A.

http://research.microsoft.com/en-us/um/people/lamport/tla/tla.html

Example

$$X \equiv \land x' = x + 1$$

$$\land y' = y$$

$$Y \equiv \land y' = y + 1$$

$$\land x' = x$$

$$S \equiv \land (x = 0) \land (y = 0)$$

$$\land \Box [X \lor Y]_{\langle x, y \rangle}$$

$$\land WF_{\langle x, y \rangle}(X) \land WF_{\langle x, y \rangle}(Y)$$

$$[x=0,x=0] \rightarrow [x=1,y=0] \rightarrow [x=1,y=0] \rightarrow [x=1,y=1] \rightarrow \dots$$

System is described in a structured way by the logical composition of actions.

TLA+

TLA is not just a logic.

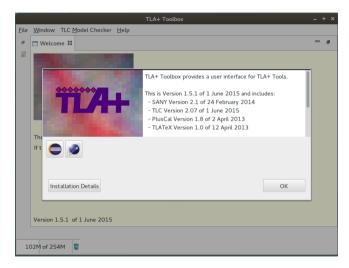
- TLA+: A formal specification language based on TLA.
 - □ Values from the theory of sets (no static type system).

Chris Newcombe et al. *How Amazon Web Services Uses Formal Methods.* Communications of the ACM, vol. 58 no. 4, pages 66-73, April 2015. https://doi.org/10.1145/2699417

- **TLA+** Toolbox: an IDE for various TLA tools.
 - □ Writing and syntax checking of TLA+ specifications.
 - □ Pretty printer for generation of LATEX documents.
 - □ Translator from the algorithmic language PlusCal to TLA+.
 - □ Simulation and model checking of TLA+-specifications.
 - Derivation and checking of TLA+ proofs.

http://research.microsoft.com/en-us/um/people/lamport/tla/tools.html

TLA+ Toolbox



Example (Plain Text)

----- MODULE Counter -----EXTENDS Naturals VARIABLE x,y

I == x = 0 / y = 0 (* the initial state condition *)

var == «x,y» (* the system variables *)

 $C == I / [][R]_var / WF_var(X) / WF_var(Y) (* the whole specification *)$

NotNegative == [](x >= 0 /\ y >= 0) (* some properties *) BecomeOne == <>(x = 1 /\ y = 1)

Example (LATEX)

- MODULE Counter _____

EXTENDS Naturals VARIABLE x, y

the initial state condition $I \stackrel{\Delta}{=} x = 0 \land y = 0$ $X \stackrel{\Delta}{=} \land x' = x + 1 \text{ increment } x$ $\land y' = y$ $Y \stackrel{\Delta}{=} \land x' = x \text{ increment } y$ $\land y' = y + 1$ $R \stackrel{\Delta}{=} \lor X$ $\lor Y$ $\lor Y$

 $var \stackrel{\Delta}{=} \langle x, y \rangle$ the system variables

the whole specification $C \stackrel{\Delta}{=} I \wedge \Box[R]_{var} \wedge \mathrm{WF}_{var}(X) \wedge \mathrm{WF}_{var}(Y)$

some properties NotNegative $\triangleq \Box(x \ge 0 \land y \ge 0)$ BecomeOne $\triangleq \diamondsuit(x = 1 \land y = 1)$

		TLA+ Toolbox		- +
le Edit Window TLC Mo	del Checker TLA Proof Manager Help			
御 へ 創 へ 🏷				
Spec Explorer	Counter.tla 🕀 Counter_1 🛱			- 0
😑 🐝 💌	Model Overview Advanced Options Model Chec	king Results		
🖆 Counter [Counter.tla]	Model Overview		c	• & = & ←
	O #? ≡ @ ↔			
	What is the behavior spec?	What is the model?		
		Specify the values of declared constants.		
	Initial predicate and next-state relation			Edit
	Init:			
	Next:			
	Temporal formula	Advanced parts of the model: Additional definitions.	Definition override.	
		State constraints, Action constraints,	Additional model values.	
	No Behavior Spec	How to run?		
	What to check?			
	what to check?			
	S Deadlock			
	Invariants			
	Properties			
	Temporal formulas true for every possible behavior.			
	NotNegative Add			
	SecomeOne Edit			
	Remove			

Select specification and properties to be checked.

File Edit Window TIC Medic Deckrer, TI A Proof Manager Help	
If Spec Explore □ Countert.1 If If Model Dearwise Model Dearwise Model Dearwise If Countert [Countert] If Model Dearwise If Counter [Countert] If If Countert If If Countert is If If Countert If If Countert is If If Countert If If If If Countert is If	
B Model Ourview (Monread Options Model Ourview (Monread Options Image: Source of Counter (I Counter Its) Image: Source of Options Image: Source of Options Image: Source of Options Image: Source of Options Image: Source of Options Image: Source of Options Image: Source of Options Image: Source of Options Image: Source of Options Image: Source of Options Image: Source of Options Image: Source of Options Image: Source of Options Image: Source of Options Image: Source of Options Image: Source of Options Image: Source of Options Image: Source of Options Image: Source of Options Image: Source of Options Image: Source of Options Image: Source of Options Image: Source of Options Image: Source of Options Image: Source of Options Image: Source of Options Image: Source of Options Image: Source of Options Image: Source of Options Image: Source of Options Image: Source of Options Image: Source of Options Image: Source of Options Image: Source of Options Image: Source of Options Image: Source of Options Image: Source of Options Image: Source of Options Image: Source of Options Image: Source of Option	
 Counter (Construction) Additional Derivations Additional Deri	- 0
 ■ Countref (Constructur) ■ # ■ # ■ Additional Definitions ■ Definitions required for the model checking in modules. ■ I = 10 ■ State Construint ■ Tate Construint ■ State Construint ■ State Construint ■ State Construint ■ Tate Construint 	
$\label{eq:constraint} \begin{split} & \text{M-distions} & \text{Definitions} & \text{Definitions or sequend for the model (decising in addition to the definitions in the specification modules. \\ & \text{If } = 10 & \text{If } \text{If } \text{C} \text{Options} & \text{If } \text{If } \text{C} \text{Options} & \text{If } \text{If } \text{C} \text{Options} & \text{If } \text{If } \text{Options} & \text{If } \text{Options} & \text{If } \text{Options} & \text{If } $	O 88 ≣ 0 ↔
Durintous required for the model checking in modules. If $u = 10$ If State Constraint a homolar structure the predicate. X \leftarrow If $//$ $y \sim$ If	
addition to the definitions in the specification $\label{eq:models} \begin{split} & \text{in Action Constraint} \\ & \text{in } = 10 \\ & \text{in State Constraint} \\ & \text{A table Constraint} \\ & \text{A table Constraint} \\ & \text{A table constraint} \\ & \text{is a formula restricting the possible states by a state predicate.} \\ & \text{is } & \text{in } / \cdot y < \infty \end{split}$	
modules. If [TLC Options] H = 10] If State Constraint is a formula restricting the possible states by a state predicate. $x \leftarrow H / / Y \leftarrow H$	
If State Constraint a a formula restricting the possible states by a state predicate. If $\kappa \in H / / Y \ll H$	
90M ir 208M	

If necessary, restrict state space to finite subset.

	TLA+ Toolbox	- + ×
File Edit Window TLC Mo	del Checker TLA Proof Manager Help	
初 へ 魚 へ 🏠		
📽 Spec Explorer 🔗 🗖	Counter.tla 🕀 Counter_1 14	- 0
😑 🙀 💌	Model Overview Advanced Options Model Checking Results	
🗉 📾 Counter [Counter.tla]	Model Checking Results	O 🍪 🖩 🌢 🕶
	○ # = ○ ← ○	
	General	
	Start time: Tue Jul 14 16:56:33 CEST 2015	
	End time: Tue Jul 14 16:56:33 CEST 2015	
	Last checkpoint time:	
	Current status: Not running	
	Errors detected: No errors	
	Fingerprint collision probability: calculated: 8.0E-16, observed: 7.5E-17	
	B Statistics	
	State space progress (click column header for graph) Coverage at 2015-07-14 16:56:33	
	Time Diamet States Fox Distinct State Queue Size Module Location Count	
	2015-07-14 16: 21 243 121 0 Counter line 10, col 9 to line 10, col 1 121	
	Counter line 11, col 9 to line 11, col 1 121	
	Counter line 8, col 9 to line 8, col 16 121	
	18 Evaluate Constant Expression	
	B User Output	
	Progress Output	
	103M of 247M	Spec Status : parsed

Check the selected properties.

TLA+ Toolbox			- + ×
File Edit Window TLC Model Checker TLA Proof Manager Help			
● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ●			
📽 Spec 😐 🗖 🔂 Counter.tla 🕀 Counter_1 18	- 0	も TLC Errors 웨	
😑 🎭 👻 Model Overview Advanced Options Model Checking Results		Counter_1	2
Counter [Counter C		Invariant RecomeOne is	violated.
O # ≡ @ ↔			
What is the behavior spec? What is the model? Specify the values of declared constants.			
Initial predicate and next-state relation			
Init:		Error-Trace Explora	tion
Next:		Error-Trace	
 Temporal formula 			Value
C Advanced parts of the model: Additional definitions. State constraints. Action constraints.	Definition over Additional mod		0
		Action line 18	State (num = 4)
O No Behavior Spec		• x	3
What to check?		= γ	0
M Deadlock		Action line 18	State (num = 5)
Invariants		• x	4
© Properties			0
Temporal formulas true for every		Action line 18	
possible behavior.			5
NotNegative Add		• y	•
BecomeOne Edit		Select line in E show its value h	
C ((x+y<5) Remove		bliow ics value in	ere.
			1
152M of 251M		Spe	ec Status : parsed

In the error case a violating system run is displayed.

Example

----- MODULE Counter -----EXTENDS Naturals, TLC VARIABLE x,y

. . .

. . .

 $C == I / [] [R / PrintT(«x,y»)]_var / WF_var(X) / WF_var(Y)$

User output may help to validate the model.

	TLA+ Toolbox	_ + ×
	del Checker TLA Proof Manager Help	
8 · 9 · 10		
📸 Spec Explorer 👘 🗖	Counter.tla - Counter_1 #	
😑 😫 💌	Model Overview Advanced Options Model Checking Results	
🗈 📾 Counter [Counter.tla]	Model Checking Results	O 🏭 🖴 🔶
	• # = • ↔	
	(i) General	
	II Statistics	
	18 Evaluate Constant Expression	
	User Output	
	TLC output generated by evaluating Print and PrintT expressions.	
	Cicle, 79 79 79 79 79 79 79 79 79 79 79 79 79	
	153M of 247M	Spec Status : parsed

The visited states are printed.

----- MODULE Counter -----EXTENDS Naturals VARIABLE x,y

I == x = 0 / y = 0 (* the initial state condition *)

```
var == «x,y» (* the system variables *)
```

 $C == I / [R]_var / WF_var(X) / WF_var(Y) (* the whole specification *)$ $S == (x = 0) / [][x' = x+1]_x / WF_x(x' = x+1) (* another system *)$

Specification of a more abstract system *S*.

	TLA+ Toolbox	- +
ile Edit Window TLC Model Checker TLA Proof Manager Help		
劉 ◇ 劉 ◇ ♥>		
f Spec Explorer 🔍 🗖 🔒 Counter.tla 🕀 Counter_1 🛱		
😑 😘 🗢 Model Overview Advanced Options Mo	del Checking Results	
🗠 Counter [Counter.tta] 💽 🛷 🔳 🙆 🐡		
@ Counter_1		
What is the behavior spec?	What is the model?	
	Specify the values of declared constants.	
Initial predicate and next-state re	lation	Edit
Init: Next:		
Temporal formula		
C	Advanced parts of the model: Additional definitions, Definition override,	
	State constraints, Action constraints, Additional model values.	
No Behavior Spec	How to run?	
What to check?		
S Deadlock		
🖲 Invariants		
E Properties		
Temporal formulas true for every possible behavior.		
NotNegative Ad	4	
BecomeOne Edi		
BecomeTooBig		
r s Remi	2ve	
	89M of 223M	ec Status : parsed

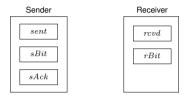
Check whether *C* refines *S* ($C \Rightarrow S$).

	TLA+ Toolbox	_ + ×
	del Checker TLA Proof Manager Help	
割 へ 値 へ 🤣		
📽 Spec Explorer 🔗 🗖	Counter.tla 🕀 Counter_1 🛤	- D
e 🐝 💌	Model Overview Advanced Options Model Checking Results	
🗖 👛 Counter [Counter.tla]	Model Checking Results	🖸 🔐 🖩 🔒 🕶
🕀 Counter_1	O &? = a ↔ lin	
	B General	
	Start time: Tue Jul 14 17:22:33 CEST 2015	
	End time: Tue Jul 14 17:22:34 CEST 2015	
	Last checkpoint time:	
	Current status: Not running	
	Errors detected: No errors	
	Fingerprint collision probability: calculated: 8.0E-16, observed: 7.5E-17	
	II Statistics	
	III Evaluate Constant Expression	
	II User Output	
	Progress Output	
	89M of 223M	Spec Status : parsed

System *C* is a valid refinement of *S*.

The Alternating Bit Protocol (Shared Memory)

Transmission of a sequence of bits between via shared registers.



var $sBit \in \{0, 1\}$, $sAck \in \{0, 1\}$, $rBit \in \{0, 1\}$, $sent \in Data$, $rcvd \in Data$ **init** sBit = sAck = rBit

loop // Sender		loop // Receiver
wait $sAck = sBit$		wait $rBit \neq sBit$
$sent = \dots; sBit = 1 - sBit$		rcvd = sent; rBit = sBit
		sAck = rBit

Liveness property: $\forall d \in Data. sent = d \land sBit \neq sAck \rightsquigarrow rcvd = d$

- $\square \text{ Response: } P \rightsquigarrow Q \equiv \square(P \Rightarrow \Diamond Q)$
- □ Request *P* is always followed by response *Q*.

The Alternating Bit Protocol (Shared Memory)

MODULE ABCorrectness EXTENDS Naturals CONSTANTS Data VARIABLES sBit, sAck, rBit, sent, rcvd

 $ABCInit \triangleq sBit \in \{0, 1\} \land sAck = sBit \land rBit = sBit \land sent \in Data \land rcvd \in Data$ $CSndNewValue(d) \triangleq \wedge sAck = sBit \wedge sent' = d \wedge sBit' = 1 - sBit$ \wedge UNCHANGED $\langle sAck, rBit, rcvd \rangle$ $CRcvMsa \triangleq \wedge rBit \neq sBit \wedge rBit' = sBit \wedge rcvd' = sent$ \wedge UNCHANGED (*sBit. sAck. sent*) $CRcvAck \triangleq \wedge rBit \neq sAck \wedge sAck' = rBit$ \wedge UNCHANGED $\langle sBit, rBit, sent, rcvd \rangle$ $ABCNext \triangleq (\exists d \in Data : CSndNewValue(d)) \lor CRcvMsg \lor CRcvAck$ $cvars \triangleq \langle sBit, sAck, rBit, sent, rcvd \rangle$ $ABCSpec \triangleq ABCInit \land \Box [ABCNext]_{cvars} \land WF_{cvars}(CRcvMsg) \land WF_{cvars}(CRcvAck)$ $TypeInv \triangleq sBit \in \{0, 1\} \land sAck \in \{0, 1\} \land rBit \in \{0, 1\} \land sent \in Data \land rcvd \in Data$ SentLeadsToRcvd $\triangleq \forall d \in Data : (sent = d) \land (sBit \neq sAck) \rightsquigarrow (rcvd = d)$

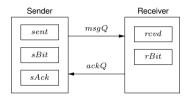
Model Checking the Protocol (Shared Memory)

		TLA+ Toolbox		
File Edit Window TLC Mod	lel Checker TLA Proof Manager Help			
📽 Spec Explorer 👘 🗖	G ABCorrectness_1 28			- 0
8 😵 💌	Model Overview Advanced Options Model Check	ing Results		
🗉 🚔 ABCorrectness [ABCorre	A Model Overview			🖸 🔐 🖩 🔒 🕶
🏥 AlternatingBit [Alternatin	0 # = 0 ++			
🏦 Counter [Counter.tla]				
mcabi [Mcabi.tia]	B What is the behavior spec?	What is the model?		
i MCAlternatingBit [MCAlt		Specify the values of declared constants.		
	Initial predicate and next-state relation	Data <- [model value] (d1, d2)		Edit
	Init:			
	Next:			
	Temporal formula ABCSpec	Advanced parts of the model: Additional definitions.	Definition override.	
	Ascopec	State constraints. Action constraints.	Additional model values.	
	No Behavior Spec	How to run?		
	What to check?			
	What to check?			
	S Deadlock			
	Invariants			
	Properties			
	Temporal formulas true for every possible behavior.			
	I []TypeInv Add			
	SentLeadsToRcvd Edit			
	Remove			
	Remove			
		146M of 252M 📋	:	Spec Status : parsed

No error: protocol satisfies specification.

The Alternating Bit Protocol (Distributed Memory)

Transmission of a sequence of bits by *lossy* communication channels.



- **m**sgQ: transmits messages $\langle sBit, sent \rangle$.
 - New values after update by sender.
- ackQ: transmits messages rBit.
 - New values after update by receiver.

This protocol shall satisfy the same correctness property as the original one.

The Alternating Bit Protocol (Distributed Memory)

——— MODULE AlternatinaBit EXTENDS Naturals, Sequences CONSTANTS Data VARIABLES msqQ, ackQ, sBit, sAck, rBit, sent, rcvd $ABInit \triangleq \land msqQ = \langle \rangle \land ackQ = \langle \rangle$ $\land sBit \in \{0, 1\} \land sAck = sBit \land rBit = sBit \land sent \in Data \land rcvd \in Data$ $ABNext \triangleq \lor (\exists d \in Data : SndNewValue(d))$ \lor ReSndMsa \lor RcvMsa \lor SndAck \lor RcvAck \lor LoseMsa \lor LoseAck abvars $\triangleq \langle msgQ, ackQ, sBit, sAck, rBit, sent, rcvd \rangle$ $ABSpec \triangleq \land ABInit \land \Box [ABNext]_{abvars}$ $\wedge WF_{abvars}(ReSndMsg) \wedge WF_{abvars}(SndAck) \wedge SF_{abvars}(RcvMsg) \wedge SF_{abvars}(RcvAck)$ $ABTypeInv \triangleq \land msgQ \in Seq(\{0, 1\} \times Data) \land ackQ \in Seq(\{0, 1\})$ $\land sBit \in \{0, 1\} \land sAck \in \{0, 1\} \land rBit \in \{0, 1\} \land sent \in Data \land rcvd \in Data$ INSTANCE ABCorrectness

The core of the specification.

The Alternating Bit Protocol (Distributed Memory)

 $SndNewValue(d) \triangleq \land sAck = sBit \land sent' = d \land sBit' = 1 - sBit$ $\wedge msgQ' = Append(msgQ, \langle sBit', d \rangle)$ \land UNCHANGED $\langle ackQ, sAck, rBit, rcvd \rangle$ $ReSndMsg \triangleq \wedge sAck \neq sBit$ $\wedge msaO' = Append(msaO, \langle sBit, sent \rangle)$ \land UNCHANGED (ackQ, sBit, sAck, rBit, sent, rcvd) $RcvMsg \triangleq \land msgQ \neq \langle \rangle \land msgQ' = Tail(msgQ) \land rBit' = Head(msgQ)[1] \land rcvd' = Head(msgQ)[2]$ \wedge UNCHANGED (ackO, sBit, sAck, sent) $SndAck \triangleq \wedge ackQ' = Append(ackQ, rBit)$ \wedge UNCHANGED $\langle msgQ, sBit, sAck, rBit, sent, rcvd \rangle$ $RcvAck \triangleq \wedge ackQ \neq \langle \rangle \wedge ackQ' = Tail(ackQ) \wedge sAck' = Head(ackQ)$ \wedge UNCHANGED $\langle msqQ, sBit, rBit, sent, rcvd \rangle$ $Lose(a) \triangleq \land a \neq \langle \rangle$ $\land \exists i \in 1 \dots Len(q) : q' = [i \in 1 \dots (Len(q) - 1) \mapsto \text{IF } i < i \text{ THEN } q[i] \text{ ELSE } q[i + 1]]$ \wedge UNCHANGED (*sBit*, *sAck*, *rBit*, *sent*, *rcvd*) $LoseMsg \triangleq Lose(msgQ) \land \text{UNCHANGED} ackQ$ $LoseAck \triangleq Lose(ackQ) \land \text{UNCHANGED} msgQ$

The actions of the specification.

State Space of the Protocol (Distributed Memory)

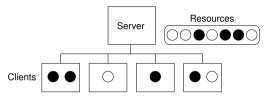
		TLA+ Toolbox		- 1	+ ×
File Edit Window TLC Mode	Checker TLA Proof Manager Help				
🐒 Spec Explorer 👘 🗖	AlternatingBit.tla 🕀 AlternatingBit_1 😫			-	
8 🔯 🔻	Model Overview Advanced Options Model Checking Re	esults			
TABCorrectness [ABCorre	Advanced Options		0 6	۵	6× .
🗉 🖨 AlternatingBit (Alternatin	O # = 0 ↔				
🏦 Counter [Counter.tla]					
iiir MCAB1 [MCAB1.tia]	Additional Definitions	Definition Override			
iii MCAlternatingBit [MCAll	State Constraint	Action Constraint			
	A state constraint is a formula restricting the possible states by a state predicate.	TLC Options			
	Len (nay) <= 2 // Len (ack) <= 2				
	167N	of 222M	Spec Status	 	_
	167/*	101 2 2 2 1 1	spec status	parsec	

Restriction of the state space to a finite subset.

Model Checking the Protocol (Distributed Memory)

			TLA+ Toolbox			- + ×
File Edit Window TLC Mod	el Checker - TLA Proof M	lanager Help				
館 Spec Explorer 👘 🗖	🗟 AlternatingBit.tla	🕀 AlternatingBit_1 😫				- 0
8 🔯 🔻		nced Options Model Check	ing Results			
TABCorrectness [ABCorre	Model Overvie	w				0 🔐 🖩 🔒 🐖
🛙 🖮 AlternatingBit (Alternatii	0 88 = 0 ++					
🏦 Counter [Counter.tla]						
🖆 MCAB1 [MCAB1.tla]	What is the behav	ior spec?	What is the model?			
📺 MCAlternatingBit [MCAlt			Specify the values of dec			
		and next-state relation	Data <- [model value]	(d1, d2)		Edit
	Init: Next:					
	Temporal formut					
	ABSpec		Advanced parts of the m	odel: Additional definitions,	Definition override,	
			State constraints.	Action constraints.	Additional model values.	
	No Behavior Spe	c	How to run?			
	B What to check?					
	C Deadlock					
	Invariants					
	Properties					
	Temporal formulas possible behavior.	true for every				
	ABTypeInv	Add				
	ABCSpec					
		Edit				
		Remove				
			167M of 222M			Spec Status : parsed

No error: the protocol refines the original one and thus inherits its correctness.



- A server allocates various resources to a set of clients.
- A client with no resources and no pending requests may request some resources.
- The server may assign some or all of the requested resources.
 - Resource requests can be processed in multiple parts; the client may potentially continue already with some part.
- The client may return a subset of its resources; ultimately it must return all of them.
- Safety: no resource is simultaneously allocated to two clients.
- Liveness: each resource request is eventually satisfied.

The method operates with the following variables.

Server:

- \Box *unsat*[*c*]: the resources requested by client *c* but not yet allocated by the server.
- \square *alloc*[*c*]: the resources requested by client *c* and allocated by the server.
- □ *sched*: the list of clients with pending requests.
 - · Older requests appear further ahead in the list and are preferably handled.

Client c:

- \Box requests[c]: the resources requested by client c that it has not yet received.
- \square *holding*[*c*]: the resources held by the client.

Netzwerk:

□ *network* : the messages pending in the network.

Since messages may be still pending in the network, the server view may be different from the client view.

— MODULE DistributedAllocator —

EXTENDS Naturals, Sequences CONSTANTS Clients, Resources VARIABLES unsat, alloc, sched, requests, holding, network

```
Messages \triangleq [type: { "request", "allocate", "return" }, clt: Clients, rsrc: SUBSET Resources]
Drop(seq, i) \triangleq SubSeq(seq, 1, i-1) \circ SubSeq(seq, i+1, Len(seq))
available \triangleq Resources \ (UNION { alloc[c] : c \in Clients })
Range(f) \triangleq \{f[x] : x \in \text{DOMAIN } f\}
Init \triangleq
   \land unsat = [c \in Clients \mapsto \{\}] \land alloc = [c \in Clients \mapsto \{\}]
   \land requests = [c \in Clients \mapsto \{\}] \land holding = [c \in Clients \mapsto \{\}]
   \land sched = \langle \rangle \land network = {}
Next \triangleq
   \forall \exists m \in network : RReg(m) \lor RAlloc(m) \lor RRet(m)
   \forall \exists c \in Clients, S \in SUBSET Resources : Request(c, S) \lor Allocate(c, S) \lor Return(c, S)
vars \triangleq \langle unsat, alloc, sched, requests, holding, network \rangle
Liveness \triangleq
   \wedge \forall c \in Clients : WF_{vars}(requests[c] = \{\} \land Return(c, holding[c]))
   \land \forall c \in Clients : WF_{vars} (\exists S \in SUBSET Resources : Allocate(c, S))
```

```
\land \forall m \in Messages : WF_{vars}(RReq(m)) \land WF_{vars}(RAlloc(m)) \land WF_{vars}(RRet(m))
```

 $Specification \stackrel{\Delta}{=} Init \land \Box[Next]_{vars} \land Liveness$

The core of the specification.

```
RReq(m) \triangleq
  \wedge m \in network \wedge m.type = "request"
  \wedge alloc[m.clt] = \{\} * don't handle request messages prematurely(!)
  \wedge unsat' = [unsat EXCEPT ![m.clt] = m.rsrc]
  \land network' = network \setminus \{m\}
  \land sched' = IF m.clt \in Range(sched) THEN sched ELSE Append(sched, m.clt)
  \wedge UNCHANGED (alloc, requests, holding)
RAlloc(m) \triangleq
  \wedge m \in network \wedge m.type = "allocate"
  \land holding' = [holding EXCEPT ![m.clt] = @ \cup m.rsrc]
  \land requests' = [requests EXCEPT ! [m.clt] = @ \ m.rsrc]
  \land network' = network \setminus \{m\}
  \wedge UNCHANGED (unsat, alloc, sched)
RRet(m) \triangleq
  \land m \in network \land m.tupe = "return"
  \wedge alloc' = [alloc \text{ EXCEPT } ! [m.clt] = @ \setminus m.rsrc]
  \land network' = network \setminus \{m\}
  \wedge UNCHANGED (unsat, sched, requests, holding)
```

The receipt of messages.

```
Request(c, S) \triangleq
   \land requests [c] = \{\} \land holding[c] = \{\}
   \land S \neq \{\} \land requests' = [requests \text{ EXCEPT } ! [c] = S]
   \land network' = network \cup {[type \mapsto "request", clt \mapsto c, rsrc \mapsto S]}
   \wedge UNCHANGED (unsat, alloc, sched, holding)
Allocate(c, S) \triangleq
   \land S \neq \{\} \land S \subseteq available \cap unsat[c]
   \land \exists i \in \text{DOMAIN sched}:
        \wedge sched[i] = c
        \land \forall j \in 1 \dots i - 1 : unsat[sched[j]] \cap S = \{\}
        \wedge sched' = IF S = unsat[c] THEN Drop(sched, i) ELSE sched
   \wedge alloc' = [alloc \text{ EXCEPT } ! [c]] = @ \cup S]
   \wedge unsat' = [unsat EXCEPT ! [c] = @ \setminus S]
   \land network' = network \cup \{[type \mapsto "allocate", clt \mapsto c, rsrc \mapsto S]\}
   \wedge UNCHANGED (requests, holding)
Return(c, S) \triangleq
  \land S \neq \{\} \land S \subseteq holding[c]
   \land holding' = [holding EXCEPT ![c] = @ \ S]
   \land network' = network \cup {[type \mapsto "return", clt \mapsto c, rsrc \mapsto S]}
   \wedge UNCHANGED (unsat. alloc. sched. requests)
```

The sending of messages.

 $\begin{array}{l} TypeInvariant \triangleq \\ \land unsat \in [Clients \rightarrow \texttt{SUBSET} Resources] \land alloc \in [Clients \rightarrow \texttt{SUBSET} Resources] \\ \land requests \in [Clients \rightarrow \texttt{SUBSET} Resources] \land holding \in [Clients \rightarrow \texttt{SUBSET} Resources] \\ \land sched \in Seq(Clients) \land network \in \texttt{SUBSET} Messages \\ ResourceMutex \triangleq \\ \forall c1, c2 \in Clients : holding[c1] \cap holding[c2] \neq \{\} \Rightarrow c1 = c2 \\ ClientsWillReturn \triangleq \\ \forall c \in Clients : (requests[c] = \{\} \sim holding[c] = \{\}) \\ Clients WillDottain \triangleq \\ \forall c \in Clients, r \in Resources : r \in requests[c] \sim r \in holding[c] \\ InfOftenSatisfied \triangleq \\ \forall c \in Clients : \Box \diamond (requests[c] = \{\}) \end{array}$

The correctness properties.

Model Checking of the Distributed Resource Allocator

		TLA+ Toolbox			-	+ ×
File Edit Window TLC Mode	al Checker TLA Proof Manager Help					
📽 Spec Explorer 👘 🗖	🕒 DistributedAllocator.tla 🕀 DistributedAl	locator_1 🛙				
😑 🍇 🔻	Model Overview Advanced Options Model Ch	acking Results				
ABCorrectness [ABCorre	Model Overview			0 🍀	•	**
AllocatorImplementation	• @ = % •					
🔛 AlternatingBit [Alternatir						
🛍 Counter [Counter.tla]	What is the behavior spec?	What is the model?				
🗉 😂 DistributedAllocator (Dis		Specify the values of declared constants.				
MCAB1 [MCAB1.tla]	Initial predicate and next-state relation	Clients <- [model value] {c1, c2, c3}			Edit	
📾 MCAlternatingBit [MCAlt	Init: Init Next: Next	Resources <- [model value] (r1, r2)				
	Temporal formula					
	Specification	Advanced parts of the model: Additional definitions.	Definition override.			
		State constraints, Action constraints,	Additional model values.			
	No Behavior Spec	How to run?				
	What to check?					
	Seadlock					
	Invariants					
	Properties					
	Temporal formulas true for every possible behavior.					
	ClientsWillReturn					
	ClientsWillObtain					
	InfOftenSatisfied					
	 Infortensatisfied 					
		95M of 202M	Spi	c Status :	pars	ed

The allocator satisfies the correctness property (for 3 clients and 2 resources).