Modelling and validating distributed systems with TLA+

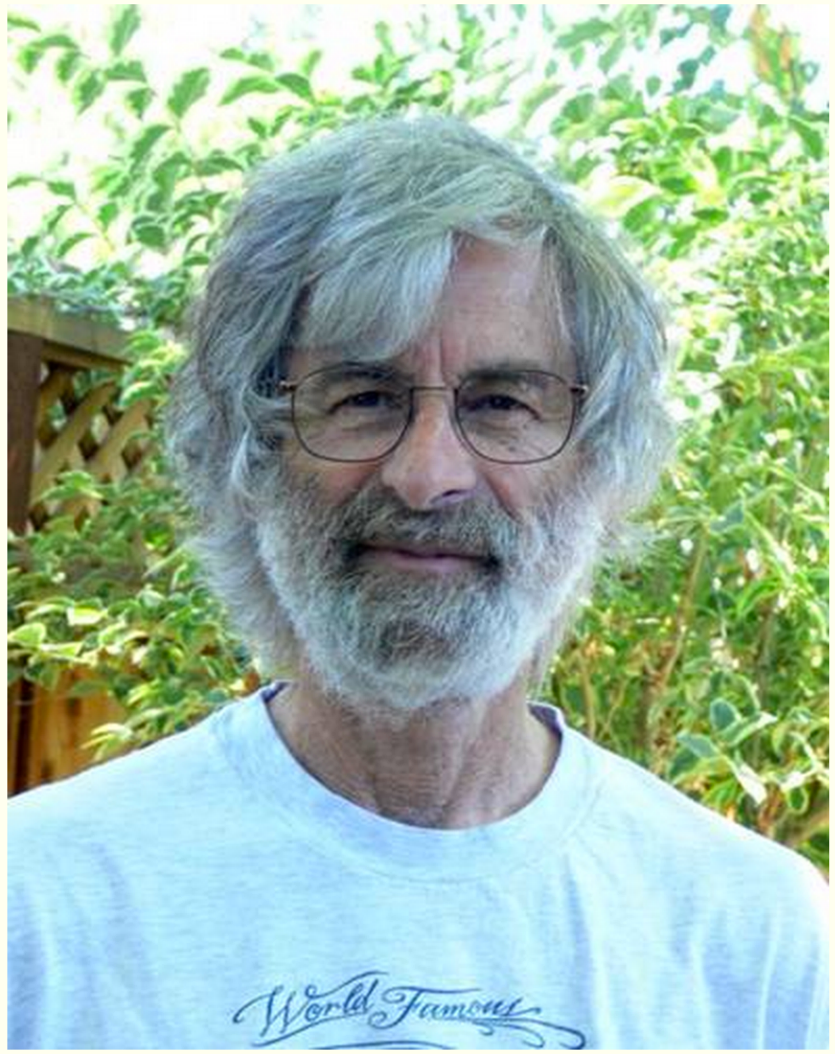
Carla Ferreira

29th April 2019

TLA+ specification language

- Formal language for describing and reasoning about distributed and concurrent systems.
- TLA+ is a model-oriented language:
  - based on mathematical logic and set theory plus temporal logic TLA (temporal logic of actions).
- Supported by the TLA Toolbox.
- References:
  - TLA+ Hyperbook (http://research.microsoft.com/en-us/um/people/lamport/tla/hyperbook.html)
  - TLA+ web page (http://research.microsoft.com/en-us/um/people/lamport/tla/tla.html)

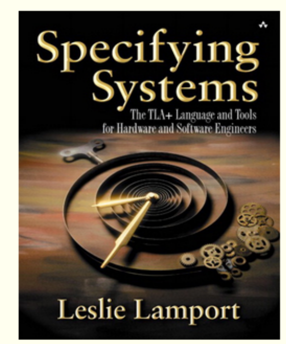
LESLIE LAMPORT'S HOME PAGE



NEW: TLA+ Use at Amazon

The TLA Web Page

My Collected Works



Turing Award 2013

For fundamental contributions to the theory and practice of distributed and concurrent systems, notably the invention of concepts such as causality and logical clocks, safety and liveness, replicated state machines, and sequential consistency.

Use of TLA+ at Amazon

“We have used TLA+ on 10 large complex real-world systems. In every case TLA+ has added significant value, either finding subtle bugs that we are sure we would not have found by other means, or giving us enough understanding and confidence to make aggressive performance optimizations without sacrificing correctness.“

Use of TLA+ at Amazon

Applying TLA+ to some of our more complex systems

System	Components	Line count (excl. comments)	Benefit
S3	Fault-tolerant low-level network algorithm	804 PlusCal	Found 2 bugs. Found further bugs in proposed optimizations.
	Background redistribution of data	645 PlusCal	Found 1 bug, and found a bug in the first proposed fix.
DynamoDB	Replication & group-membership system	939 TLA+	Found 3 bugs, some requiring traces of 35 steps
EBS	Volume management	102 PlusCal	Found 3 bugs.
Internal distributed lock manager	Lock-free data structure	223 PlusCal	Improved confidence. Failed to find a liveness bug as we did not check liveness.
	Fault tolerant replication and reconfiguration algorithm	318 TLA+	Found 1 bug. Verified an aggressive optimization.

First TLA+ Example

1-bit Clock

- Clock's possible behaviours:

$$b = 1 \longrightarrow b = 0 \longrightarrow b = 1 \longrightarrow b = 0 \longrightarrow \ldots$$

$$b = 0 \longrightarrow b = 1 \longrightarrow b = 0 \longrightarrow b = 1 \longrightarrow \ldots$$

1-bit Clock

- State variable:

  b

- Initial predicate:

  $$b = 1 \ \lor\ b = 0$$

- Next-step action (b’ is the variable at the next state):

  $$\begin{aligned} &\lor (b = 0) \land (b' = 1) \\ &\lor (b = 1) \land (b' = 0) \end{aligned}$$

The initial state and next-step action are formulas in TLA

1-bit Clock

- State variable:

  b

- Initial predicate:

  $b = 1 \lor b = 0$

- Next-step action (b’ is the variable at the next state):

  $\textbf{IF } b = 0 \textbf{ THEN } b' = 1$
  $\textbf{ELSE } b' = 0$

The initial state and next-step action are formulas in TLA

1-bit Clock: TLA specification

```
----------------------------- MODULE OneBitClock -----------------------------
VARIABLE b

Init == (b=0) \/ (b=1)



Next == \/ b = 0 /\ b' = 1
        \/ b = 1 /\ b' = 0





==============================================================================
```

What about the clock properties?

System’s properties

- Safety
  - Something bad never happens
  - E.g. system never deadlocks, the account balance is greater or equal to zero
- Liveness
  - Something good eventually happens
  - E.g. if a process request access to a critical region it will eventually be granted access, the light will eventually turn green

Let’s ignore liveness properties for now

1-bit Clock: TLA specification

```
----------------------------- MODULE OneBitClock -----------------------------
VARIABLE b

Init == (b=0) \/ (b=1)

TypeInv == b \in {0,1}

Next == \/ b = 0 /\ b' = 1
        \/ b = 1 /\ b' = 0

Spec == Init /\ [][Next]_<<b>>
------------------------------------------------------------------------------

THEOREM Spec => []TypeInv

==============================================================================
```

Typing information (TLA+ is untyped)

1-bit Clock: TLA specification

```
----------------------------- MODULE OneBitClock -----------------------------
VARIABLE b

Init == (b=0) \/ (b=1)

TypeInv == b \in {0,1}

Next == \/ b = 0 /\ b' = 1
        \/ b = 1 /\ b' = 0

Spec == Init /\ [][Next]_<<b>>
------------------------------------------------------------------------------

THEOREM Spec => []TypeInv

==============================================================================
```

```
[A]_<<f>> == A \/ (f’ = f)
```

The initial state satisfies *Init*
Every transition satisfies *Next* or leaves *b* unchanged

1-bit Clock: TLA specification

```
----------------------------- MODULE OneBitClock -----------------------------
VARIABLE b

Init == (b=0) \/ (b=1)

TypeInv == b \in {0,1}

Next == \/ b = 0 /\ b' = 1
        \/ b = 1 /\ b' = 0

Spec == Init /\ [][Next]_<<b>>
------------------------------------------------------------------------------
THEOREM Spec => []TypeInv
==============================================================================
```

Theorem specifies an invariant property

TLC model checker

- Exhaustive breath-first search of all reachable states
    - Finds (one of) the shortest path to the property violation

Computing all possible behaviours

- State graph is a directed graph `G`

1. Put into `G` to the set of all initial states

2. For every state `s` in `G` compute all possible states `t` such that `s` $\longrightarrow$ `t` can be a step in a behaviour

3. For every state `t` found in step 2 not in `G`, draw an edge from `s` to `t`

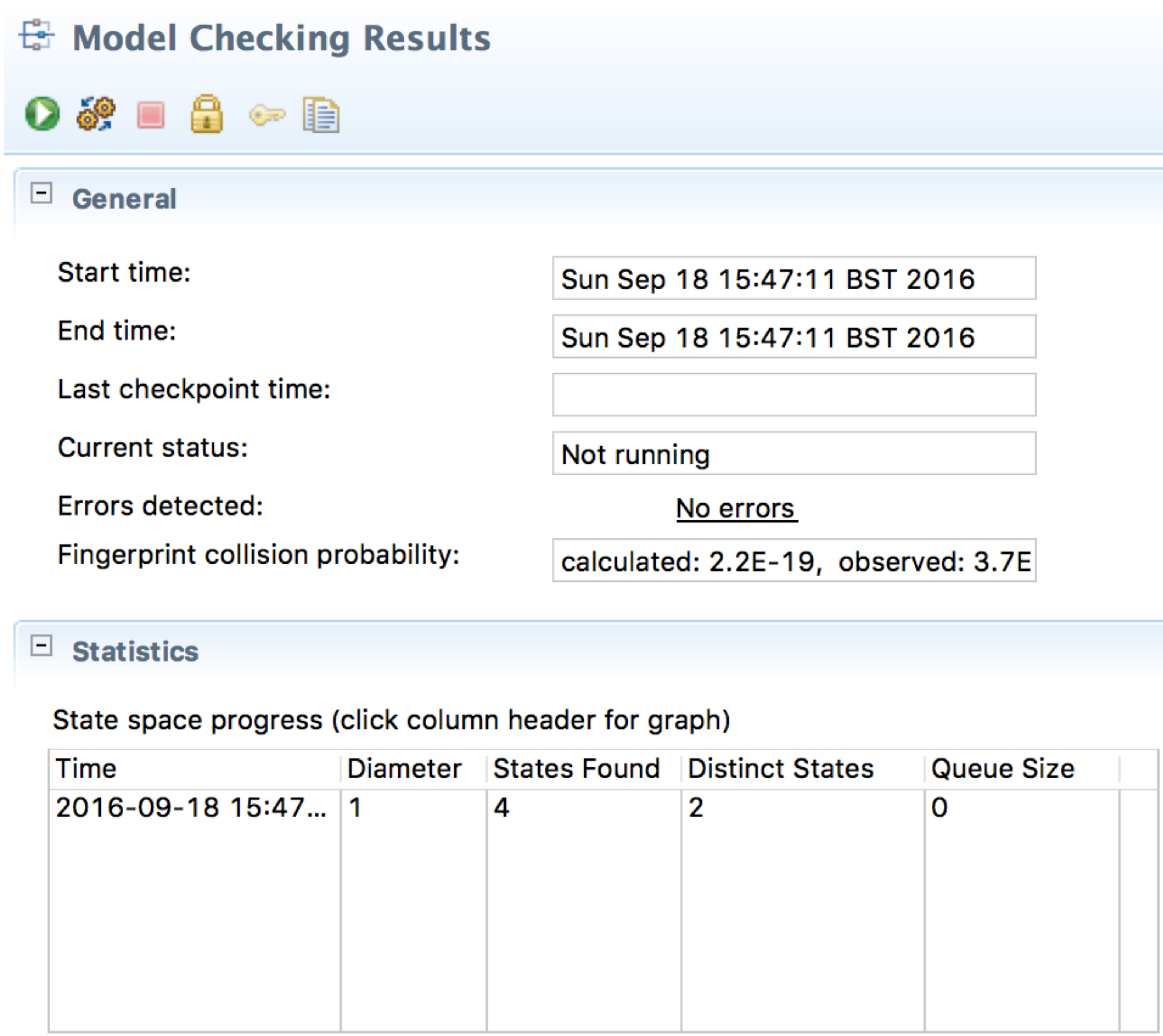
4. Repeat the previous steps until no new states or edges can be added to `G`

TLC: state space progress

- Diameter
  - Number of states in the longest path of `G` with no repeated states
- States found
  - Total number of states it examined in step 1 and 2
- Distinct states
  - Number of states that form the set of nodes of `G`
- Queue size
  - Number of states s in `G` for which step 2 has not yet been done

1-bit Clock: Model checking

- Checking the 1-bit clock with TLC model checker (demo)



Exercise 1

- Define a TLA+ specification of an hour clock
- Check with TLC the typing invariant

TLA+ Overview

TLA+ Module

```
---------------------------------- MODULE M ----------------------------------
EXTENDS M1,..., Mn
\* Incorporates the declarations, definitions, assumptions, and theorems from
\* the modules named M1,...,Mn into the current module.

CONSTANTS C1,..., Cn \* Declares the C1,..., Cn to be constant parameters.

ASSUME P \* Asserts P as an assumption.

VARIABLES x1,..., xn \* Declares x1,..., xn as variables.

TypeInv == exp \* Declares the types of variables x1,..., xn.

Init == exp \* Initializes variables x1,..., xn.

F(x1,..., xn) == exp
\* Defines F to be an operator such that
\* F(e1,...,en) equals exp with each identifier xk replaced by ek.

f[x \in S] == exp
\* Defines f to be the function with domain S such that f[x] = exp
\* for all x in S.
\* The symbol f may occur in exp, allowing a recursive definition.

THEOREM P
\*Asserts that P can be proved from the definitions and assumptions of the
\*current module.
==============================================================================
```

TLA+ syntax and semantics

- Logic
- Sets
- Functions
- Tuples, sequences and records
- EXCEPT, UNION, and CHOOSE operators

Logic

$\land \quad \lor \quad \neg \quad \Rightarrow \quad \equiv$

TRUE FALSE BOOLEAN [the set {TRUE, FALSE}]

$\forall x \in S : p \qquad \exists x \in S : p$

```
~(TRUE /\ b)

a => b

Next == b' = 0

b \in BOOLEAN

x \notin S

\A x \in {1, 2, 3, 4, 5} : x >= 0

\E x \in {1, 2, 3, 4, 5} : x % 2 = 0
```

Sets

$= \quad \neq \quad \in \quad \notin \quad \cup \quad \cap \quad \subseteq \quad \setminus$ [set difference]

$\{e_1, \ldots, e_n\}$	[Set consisting of elements e_i]
$\{x \in S : p\}$ (2)	Set of elements x in S satisfying p]
$\{e : x \in S\}$ (1)	Set of elements e such that x in S]
SUBSET S	Set of subsets of S]
UNION S	[Union of all elements of S]

```
S = {1, 2, 3}

S # {1, 2, 3}  S /= {1, 2, 3}

x \in S

x \notin S

S \union {1, 2, 3}

{ n \in {1, 2, 3, 4, 5} : n % 2 != 0 } = {1, 3, 5}

{ 2*n+1 : n \in {1, 2, 3, 4, 5} } = {3, 5, 7, 9, 11}

UNION { {1, 2}, {2, 3}, {3, 4} } = {1, 2, 3, 4}

SUBSET {1, 2} = {{}, {1}, {2}, {1, 2}}
```

CHOOSE

```
CHOOSE x \in S : P(x)
\* Equals some value v in S such that P(v) equals true, if such a value exists.
\* Its value is unspecified if no such v exists


CHOOSE x \in {1, 2, 3, 4, 5} : TRUE

CHOOSE x \in {1, 2, 3, 4, 5} : x % 2 = 0
```

CHOOSE is deterministic!

CHOICE vs. non-determinism

```
removeOneDet ==
IF procs \= {}
THEN procs' =
       procs \ {CHOOSE t \in procs : TRUE}
ELSE UNCHANGED procs
```

```
removeOneNonDet ==
IF procs \= {}
THEN \E x \in procs : procs' = procs \ {x}
ELSE UNCHANGED waiting
```

Deterministic

a single sucessor state

Non-deterministic

many of successor states

Functions

$f[e]$	[Function application]
DOMAIN f	[Domain of function f]
$[x \in S \mapsto e]$	[Function f such that $f[x] = e$ for $x \in S$]
$[S \to T]$	[Set of functions f with $f[x] \in T$ for $x \in S$]
$[f$ EXCEPT $![e_1] = e_2]$	[Function $\widehat{f}$ equal to f except $\widehat{f}[e_1] = e_2$. An @ in e_2 equals $f[e_1]$.]

```
[i \in {2,3,5,9} |-> i - 7] = (2 :> -5 @@ 3 :> -4 @@ 5 :> -2 @@ 9 :> 2)

DOMAIN [i \in {2,3,5,9} |-> i - 7] = {2, 3, 5, 9}

[ [i \in {2,3,5,9} |-> i - 7][3] = -4

[ {2,4} -> { "a", "b" } ] = { (2 :> "a" @@ 4 :> “a”), (2 :> "a" @@ 4 :> "b"),
                              (2 :> "b" @@ 4 :> “a”), (2 :> "b" @@ 4 :> "b") }

[ [i \in {2,3,5,9} |-> i - 7] EXCEPT ![2]= 12 ] =
                               (2 :> 12 @@ 3 :> -4 @@ 5 :> -2 @@ 9 :> 2)
```

Records

$e.h$	[The h-component of record e]
$[h_1 \mapsto e_1, \ldots, h_n \mapsto e_n]$	[The record whose h_i component is e_i]
$[h_1 : S_1, \ldots, h_n : S_n]$	[Set of all records with h_i component in S_i]
$[r \text{ EXCEPT } !.h = e]$	[Record $\widehat{r}$ equal to r except $\widehat{r}.h = e$. An @ in e equals $r.h$.]

```
[node |-> "n1", edge |-> "e1"]

[node |-> "n1", edge |-> "e1"].edge = "e1"

[nodes : {"n1","n2"}, edges : {"e1","e2"}]

[node |-> "n1", edge |-> "e1"] EXCEPT !.edge = "xpto"] =
                                        [node |-> "n1", edge |-> "xpto"]
```

Tuples

$e[i]$	[The i^{th} component of tuple e]
$\langle e_1, \ldots, e_n \rangle$	[The n-tuple whose i^{th} component is e_i]
$S_1 \times \ldots \times S_n$	[The set of all n-tuples with i^{th} component in S_i]

```
<<"ana", 32, 37495>>

<<"ana",32>>[2] = 32

<<"ana",32>>[1] = "ana"

{1,2,3} \times {"a","b"} = { <<1, "a">>, <<1, "b">>, <<1, "c">>,

                             <<2, "a">>, <<2, "b">>, <<2, "c">>,

                             <<3, "a">>, <<3, "b">>, <<3, "c">> }
```

Sequences

```
------------------------------ MODULE Sequences ------------------------------
LOCAL INSTANCE Naturals

Seq(S) == UNION {[1..n -> S] : n \in Nat}

Len(s) == CHOOSE n \in Nat : DOMAIN s = 1..n

s \o t == [i \in 1..(Len(s) + Len(t)) |-> IF i \leq Len(s) THEN s[i]
                                                           ELSE t[i-Len(s)]]
Append(s, e) == s \o <<e>>

Head(s) == s[1]

Tail(s) == [i \in 1..(Len(s)-1) |-> s[i+1]]

SubSeq(s, m, n) == [i \in 1..(1+n-m) |-> s[i+m-1]]

==============================================================================
```

Other constructs

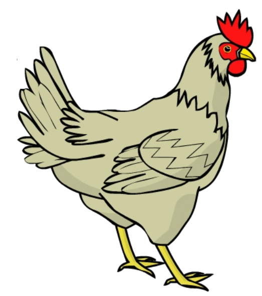
IF p THEN e_1 ELSE e_2	[e_1 if p true, else e_2]
CASE $p_1 \rightarrow e_1 \,\square\, \ldots \,\square\, p_n \rightarrow e_n$	[Some e_i such that p_i true]
CASE $p_1 \rightarrow e_1 \,\square\, \ldots \,\square\, p_n \rightarrow e_n \,\square\,$ OTHER $\rightarrow e$	[Some e_i such that p_i true, or e if all p_i are false]

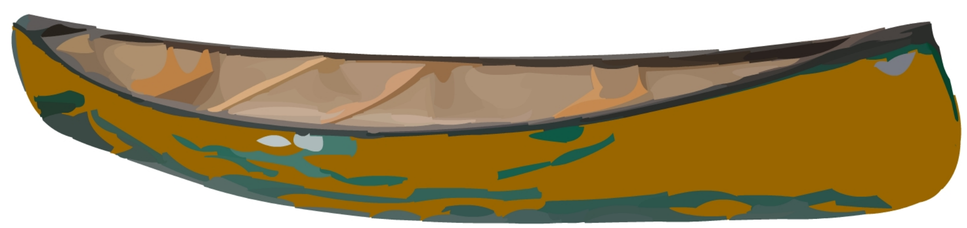
LET $d_1 \triangleq e_1 \;\ldots\; d_n \triangleq e_n$ IN e [e in the context of the definitions]

$$\begin{array}{l} \wedge\; p_1 \\ \quad\vdots \\ \wedge\; p_n \end{array}$$ [the conjunction $p_1 \wedge \ldots \wedge p_n$]

$$\begin{array}{l} \vee\; p_1 \\ \quad\vdots \\ \vee\; p_n \end{array}$$ [the disjunction $p_1 \vee \ldots \vee p_n$]

Crossing the river



- A farmer is on one shore of a river and has with him a fox, a chicken, and a sack of grain.
- He has a boat that fits one item besides himself.
- In the presence of the farmer nothing gets eaten, but if left without the farmer, the fox will eat the chicken, and the chicken will eat the grain.
- How can the farmer get all three items across the river safely?

Exercise: Crossing the river

- Define a TLA+ specification for this problem.
- Check with TLC the typing invariant.
- Add an invariant stating that is not possible to get all three items across the river.
- Use TLC to find a solution to this problem.

Only allow safe operations

Exercise: Crossing the river

- Some help:

```
------------------------------------ MODULE CrossingRiver ------------------
EXTENDS Integers

CONSTANTS Farmer, Fox, Chicken, Grain

Items == {Fox, Chicken, Grain}

safe(S) == ~({Fox, Chicken} \subseteq S \/ {Chicken, Grain} \subseteq S)

VARIABLES onLeftShore, onRightShore

TypeInv ==
    /\ onLeftShore \in SUBSET (Items \union {Farmer})
    /\ onRightShore \in SUBSET (Items \union {Farmer})
```

Crossing the river: Solution

- crossWithItem {Farmer,Chicken}
- crossAlone {Farmer}
- crossWithItem {Farmer,Grain}
- crossWithItem {Farmer,Chicken}
- crossWithItem {Farmer,Fox}
- crossAlone {Farmer}
- crossWithItem {Farmer,Chicken}

Name	Value
<Initial predicate>	State (num = 1)
onLeftShore	{Grain, Chicken, Farmer, Fox}
onRightShore	{ }
<crossWithItem line 35, col 7 to line 38, col 64 of modul...	State (num = 2)
onLeftShore	{Grain, Fox}
onRightShore	{Chicken, Farmer}
<crossAlone line 29, col 8 to line 32, col 52 of module Cr...	State (num = 3)
onLeftShore	{Grain, Farmer, Fox}
onRightShore	{Chicken}
<crossWithItem line 35, col 7 to line 38, col 64 of modul...	State (num = 4)
onLeftShore	{Fox}
onRightShore	{Grain, Chicken, Farmer}
<crossWithItem line 39, col 8 to line 42, col 62 of modul...	State (num = 5)
onLeftShore	{Chicken, Farmer, Fox}
onRightShore	{Grain}
<crossWithItem line 35, col 7 to line 38, col 64 of modul...	State (num = 6)
onLeftShore	{Chicken}
onRightShore	{Grain, Farmer, Fox}
<crossAlone line 29, col 8 to line 32, col 52 of module Cr...	State (num = 7)
onLeftShore	{Chicken, Farmer}
onRightShore	{Grain, Fox}
<crossWithItem line 35, col 7 to line 38, col 64 of modul...	State (num = 8)
onLeftShore	{ }
onRightShore	{Grain, Chicken, Farmer, Fox}