

# TLA+/Pluscal Intro

Murat Demirbas



# Game plan

State-centric thinking/modeling

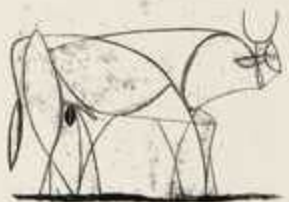
Computation model for state-centric thinking

2-color can problem

Hands-on modeling: 3-color can problem



# The art of abstraction & state-centric modeling



Pinto

# Abstraction is a powerful tool for avoiding distraction

Etymology: Latin for cut and draw away. With abstraction, you slice out the protocol from a complex system, omit unnecessary details, & simplify a complex system into a useful model

*Abstraction, abstraction, abstraction! That's how you win a Turing Award. – L. Lamport*

*The only mental tool by means of which a very finite piece of reasoning can cover a myriad cases is called **abstraction**; as a result, the effective exploitation of his powers of abstraction must be regarded as one of the most vital activities of a competent programmer. The purpose of abstraction is not to be vague, but to create a new semantic level in which one can be absolutely precise. – E. Dijkstra*

# Foundations of TLA+: specifying with math

TLA+ is useful for teaching engineers the art of abstraction and the right way to think/reason about distributed systems

TLA+ forces you to specifying using math, rather than a programming language. It forces you to focus on the **what** rather than the **how**. It shakes you out of your comfort zone. And you have to think mathematically and in a state-centric manner.

# TLA+: global state-transition model

TLA+ gives you a stop-the-world transition from one global state to another state by way of a protocol action

It gives you the God view into the global state of the distributed system you are modeling

The consequence is that safety invariants and liveness properties are written as one liners since the entire global state is accessible for evaluation

This state-centric thinking model enables TLA+ users to model and reason about distributed systems in a succinct and principled manner

# State-centric, invariant-based reasoning

Invariant-based reasoning provides a non-operational way to reason about concurrent systems and avoids the complexities/bugs of operational reasoning

Instead of happy-path operational thinking, we ask "what needs to go right?"

Safety properties: what the system is allowed to do e.g., at all times, all committed data is present & correct

Liveness properties: what the system must eventually do e.g., whenever the system receives a request, it must eventually respond to that request





# Computation model

# Computation model

Programs consist of:

1. A set of variables (global shared memory model)
2. A finite set of assignments (called "actions")

$\{state\}$  transitioning-action  $\{state'\}$

- $state'$  is the resulting state
- A Hoare Triple

# Guarded actions

The execution of an assignment can be conditional on a predicate (i.e., guard) being true

$$x > 0 \rightarrow x := 0; y := y + 1$$

Or in TLA+ notation

$$\begin{aligned} &\wedge x > 0 \\ &\wedge x' = 0 \\ &\wedge y' = y + 1 \end{aligned}$$

The guard is a predicate on the state space of the program. If the guard is true in some state, the action is said to be "enabled" in that state

# Program execution

1. A program can begin in any state satisfying the initial predicate
2. An action is nondeterministically selected & executed atomically

Once this action has completed, an action is again non-deterministically selected and executed. This process is repeated infinitely.

If the selected action is guarded and the guard is false, the action is simply a skip (i.e., state is unchanged)



Can problem

# Can problem

Consider a can of beans. Each bean is either white or black. The can is initially nonempty ( $w + b > 0$ ).

Choose two beans from the can;

- if they are same color, toss'em out & put in a white bean

- if they are different, toss'em out & put in a black bean

This action is repeated.

# Questions

Will we get to the bottom of the can eventually?

Can the number of beans in the can reach zero?

If the program terminates with  $b = 1$ , what must have been true of the beans in the can at the beginning of the computation?

How did you reason? Would your reasoning scale?

# Guarded-command program

Program CAN

$w, b : \mathbb{N}$

initially  $w + b > 0$

WW:  $w > 1 \quad \rightarrow \quad w := w-1$

BB:  $b > 1 \quad \rightarrow \quad b := b-2; w := w+1$

WB:  $w > 0 \wedge b > 0 \quad \rightarrow \quad w := w-1$



# TLA+ modeling

≡ beans.tla > ...

```
1  ----- MODULE beans -----
2  EXTENDS Integers
3  CONSTANTS W, B
4  ASSUME /\ W \in 0..100
5          /\ B \in 0..100
6          /\ W+B > 0
7
8  VARIABLES w,b
9
10  Init == /\ w = W
11          /\ b = B
12
```

## TLA+: actions

```
13  WW == /\ w > 1  \* same color and white
14         /\ w' = w-1 /\ UNCHANGED b
15
16  BB == /\ b > 1  \* same color and black
17         /\ b' = b - 2
18         /\ w' = w + 1
19
20  WB == /\ w > 0 /\ b > 0  \* different color
21         /\ w' = w - 1 /\ UNCHANGED b
22
23  Next == \/ WW
24         \/ BB
25         \/ WB
```

# TLA+: invariants

```
28  vars == <<w,b>>
29  Spec == Init /\ [] [Next]_vars
30          /\ WF_vars(Next) \* Weak Fairness
31
32  TypeOK ==  w+b > 0
33
34  Termination == <> (w+b < 2)
35
36  DecreasingN == [] [w'+b'< w+b]_vars
37
38  EvenB == [] [b%2=0 => b'%2=0]_vars
39
40  OddB == [] [b%2=1 => b'%2=1]_vars
```

# TLA+ config file

≡ beans.cfg

```
1  CONSTANTS
2      W = 5
3      B = 3
4
5  SPECIFICATION Spec
6
7  INVARIANTS
8      TypeOK
9
10 PROPERTIES
11     Termination
12     DecreasingN
13     EvenB
```

# PlusCal modeling

```
3  CONSTANTS W, B
4  ASSUME /\ W \in 0..100 /\ B \in 0..100
5         /\ W+B > 0
6
7  (* --fair algorithm beansAlg {
8      variable w=W, b=B;
9      {S:while (TRUE)
10         { either
11             {await (w>1); \* \* same color and white
12 WW:      w:=w-1;};
13         or
14             {await (b>1); \* \* same color and black
15 BB:      b:= b-2; w:= w+1;};
16         or
17             {await (w>0 /\ b>0); \* \* different color
18 WB:      w:= w-1;};
19         }
20     }
21 }
22 *)
```

# Pluscal translates to TLA+

```
24  \* BEGIN TRANSLATION (chksum(pcal) = "7c28162a" /\ chksum(tla) = "c
25  VARIABLES w, b, pc
26
27  vars == << w, b, pc >>
28
29  Init == (* Global variables *)
30          /\ w = W
31          /\ b = B
32          /\ pc = "S"
33
34  S == /\ pc = "S"
35        /\ \ / /\ (w>1)
36            /\ pc' = "WW"
37        \ / /\ (b>1)
38            /\ pc' = "BB"
39        \ / /\ (w>0 /\ b>0)
40            /\ pc' = "WB"
41        /\ UNCHANGED << w, b >>
42
```

# Pluscal translates to TLA+

```
43  WW == /\ pc = "WW"
44         /\ w' = w-1
45         /\ pc' = "S"
46         /\ b' = b
47
48  BB == /\ pc = "BB"
49         /\ b' = b-2
50         /\ w' = w+1
51         /\ pc' = "S"
52
53  WB == /\ pc = "WB"
54         /\ w' = w-1
55         /\ pc' = "S"
56         /\ b' = b
57
58  Next == S \/ WW \/ BB \/ WB
59
60  Spec == /\ Init /\ [] [Next]_vars
61         /\ WF_vars(Next)
```

# TLA+ & invariant-based reasoning

TLA helps you find invariants by testing your candidates

Then the invariants are used for improving over the system without breaking it

For invariant-based reasoning, it is enough to consider each action/procedure once (instead of several times in a trace for operational reasoning) and prove that the procedure preserves the invariant

- After proving each procedure preserves the invariant, we are guaranteed by induction that regardless of execution sequence of the procedures the invariant continues to hold in the system





Hands on:  
3 color can problem

# 3 color can problem

Consider a can of beans. The beans come in 3 different colors: red, green, and blue. The can is initially nonempty ( $r + g + b > 0$ ).

Choose two beans from the can;

- if they are same color, toss'em out

- if they are different colors, toss'em out & add a bean of the third color

This rule is repeated



# Discussion

# Discussion

Give examples to safety vs liveness properties from your work/services

Can we model any system property as a safety or liveness property? What about if we restrict to correctness properties only?

Give examples to the importance of being able to jump to abstract plane for reasoning

When was the last time you struggled with an unfamiliar hard problem, and overcome it? What was your thinking process?

# Discussion

What do you think of modeling distributed/concurrent in guarded-command lang? Is it too simple to be useful? Or is it a good abstraction suitable for distributed/concurrent system modeling?

What is the mapping between guarded-command to TLA+?

What is the mapping between PlusCal and TLA+?

How do labels play into the PlusCal to TLA+ / guarded-command?