INTRODUCTION TO TLA+

Presented by : Kevin Yeh

What is TLA+

- Specification Language for modelling complex or concurrent systems
- TLA+ toolbox performs model checks to check for correctness
- PlusCAL

What can TLA+ do for you?

- Modelling of ALGORITHMS prior to implementation
- Meant as a supplement to traditional test/ verification
- Very powerful bug detection

What can TLA+ do for you?

- Been used successfully at Amazon, HP, and Intel
- Two weeks before value was added

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.

Intangibles

- Requires up-front system understanding
- Adds value even after production release

TLA+ an Overview

- 4 parts to a specification
  - Initial predicate
  - Possible “Next” states
  - Safety Properties
  - Liveness Properties

Alternating One-bit Clock

- Initial Predicate
  - (b = 0) V (b = 1)
- Next States
  - ((b = 0) /\ (b’ = 1)) V ((b = 1) /\ (b’ = 0))

Alternating One-bit Clock

- Initial Predicate
  - (b = 0) V (b = 1)
- Next States
  - ((b = 0) /\ (b’ = 1)) V ((b = 1) /\ (b’ = 0))

```
VARIABLE b

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

Next ==  \/ /\ b= 0
            /\ b’ = 1
         \/ /\ b = 1
            /\ b’ = 0
```

Die Hard Problem

- What you have: 3-gallon jug, 5-gallon jug, and a faucet
- Goal: Measure 4 gallons

Die Hard Problem

```
VARIABLES big, small


Init == /\ big = 0

        /\ small = 0



Next == \/ FillSmall
        \/ FillBig
        \/ EmptySmall
        \/ EmptyBig
        \/ SmallToBig
        \/ BigToSmall
```

Die Hard Problem

```
VARIABLES big, small


Init == /\ big = 0

        /\ small = 0



Next == \/ FillSmall
        \/ FillBig
        \/ EmptySmall
        \/ EmptyBig
        \/ SmallToBig
        \/ BigToSmall
```

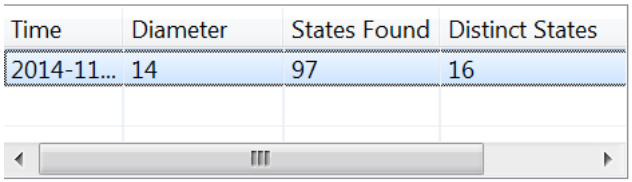
FillSmall == /\ small' = 3
/\ big' = big

SmallToBig == \/ /\ big+ small >5
/\ big' = 5
/\ small' = small – (5-big)
\/ /\ big + small <= 5
/\ big' = big + small
/\ small' = 0

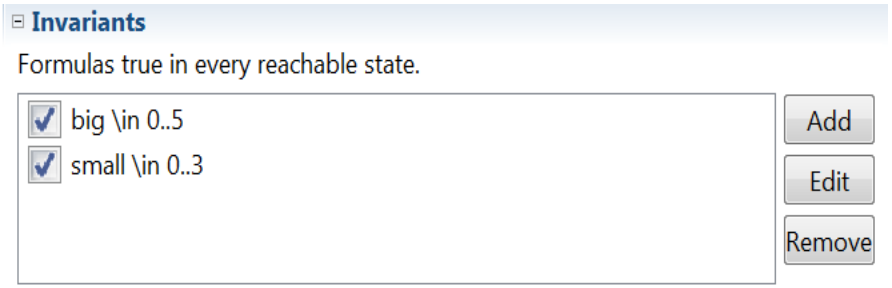
Model Checker

- Builds up a Directed Graph of all *possible* states.

State Statistics



Invariant Checker



Die Hard - Solution

Invariants

Formulas true in every reachable state.

big # 4

Add

Edit

Remove

Error-Trace

Name	Value
<Initial predicate>	State (num = 1)
big	0
small	0
<Action line 15, col 12	State (num = 2)
big	5
small	0
<Action line 35, col 7	State (num = 3)
big	2
small	3
<Action line 18, col 15	State (num = 4)
big	2
small	0
<Action line 35, col 7	State (num = 5)
big	0
small	2
<Action line 15, col 12	State (num = 6)
big	5

Select line in Error Trace to show its value here.

Safety/Liveness Properties

- Safety Property – Define a correct behavior of your procedure
  - Partial Correctness : (terminated) => (Correct_Output)
- Liveness Property – Define a correct behavior that must *eventually* hold
  - Termination

Euclid's Algorithm – a high level view

- Find the Greatest Common Divisor of two numbers
- General Procedure:
  - PlusCAL -> TLA+
  - Write the definition of GCD using set logic: GCD(m,n)
  - Use definition to write Safety/Liveness Properties
- This is how TLA+ is used in industry

Euclid's Algorithm

- PlusCAL code:

```
--algorithm Euclid{
variables x = M, y= N;
{ while (x # y) { if(x<y) { y:= y-x }
                    else    { x:= x - y}

                 };
}
}
```

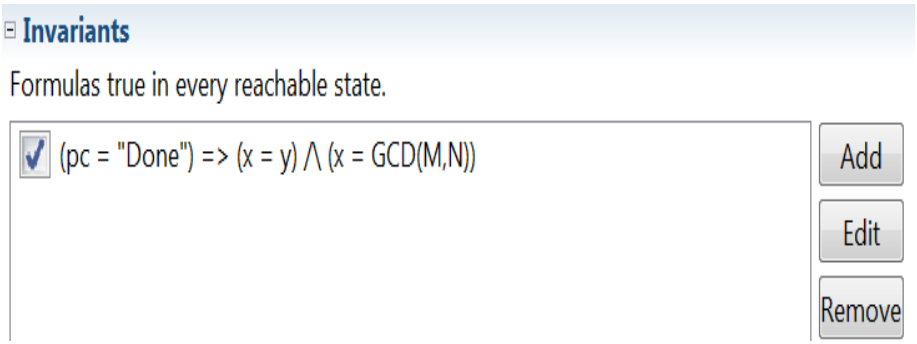
```
\* BEGIN TRANSLATION
VARIABLES x, y, pc

vars == << x, y, pc >>

Init == (* Global variables *)
        /\ x = M
        /\ y = N
        /\ pc = "Lbl_1"
```

```
Lbl_1 == /\ pc = "Lbl_1"
         /\ IF x # y
               THEN /\ IF x<y
                          THEN /\ y' = y-x
                               /\ x' = x
                          ELSE /\ x' = x - y
                               /\ y' = y
                    /\ pc' = "Lbl_1"
               ELSE /\ pc' = "Done"
                    /\ UNCHANGED << x, y >>
```

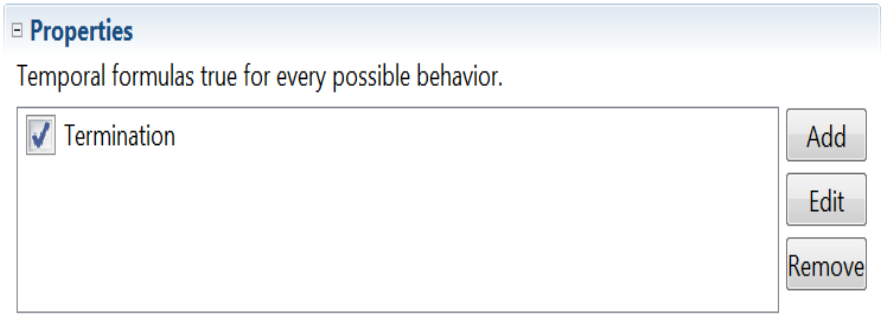
```
Next == Lbl_1
           \/ (* Disjunct to prevent deadlock on termination *)
              (pc = "Done" /\ UNCHANGED vars)
```

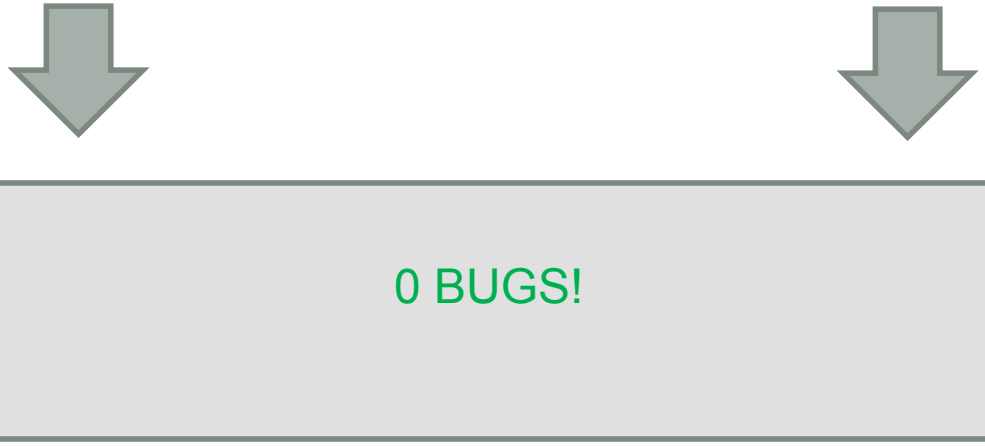
Model Checking

Safety



Liveness





Questions?