

# TLA<sup>+2</sup>

Kaustuv Chaudhuri

*Damien Doligez*

Leslie Lamport

Stephan Merz

Simon Zambrovski



January 28, 2009

# Outline

1 TLA<sup>+</sup>

2 TLA<sup>+2</sup>

3 Demo

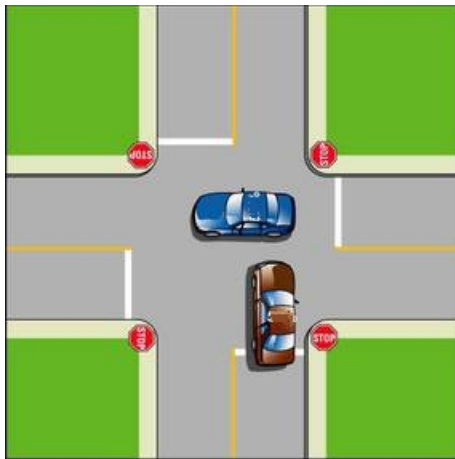
4 Conclusion and Future work

- A specification language based on :
  - ▶ First-order logic
  - ▶ Set theory
  - ▶ (Linear) temporal logic
- Especially suited for specifying and describing concurrent and distributed systems
  - ▶ Distributed fault-tolerant consensus (Paxos)
  - ▶ Shared virtual memory hardware (WildFire)
  - ▶ Garbage collection software (Concurrent Caml Light)
  - ▶ Web services protocols

### Tools :

- TLAsany : parser and static checker
- TLC : model-checker
- PlusCal compiler

# Peterson's algorithm



## PlusCal example

```
algorithm Peterson {
  variables flag = [i \in {0, 1} |-> FALSE], turn = 0;
  process (proc \in {0,1}) {
    a0: while (TRUE) {
      a1:  flag[self] := TRUE;
      a2:  turn := Not(self);
      a3:  while (flag[Not(self)] /\ turn = Not(self)) {
            skip };
      cs:  skip; /* critical section
      a4:  flag[self] := FALSE;
    } /* end while
  } /* end process
}
```

## TLA<sup>+</sup> example

```
cs(self) == /\ pc[self] = "cs"  
            /\ TRUE  
            /\ pc' = [pc EXCEPT ![self] = "a4"]  
            /\ flag' = flag  
            /\ turn' = turn
```

```
a4(self) == /\ pc[self] = "a4"  
            /\ flag' = [flag EXCEPT ![self] = FALSE]  
            /\ pc' = [pc EXCEPT ![self] = "a0"]  
            /\ turn' = turn
```

```
proc(self) == a0(self) \/ a1(self) \/ a2(self) \/ a3(self)  
              \/ cs(self) \/ a4(self)
```

```
Next == (\E self \in {0,1}: proc(self))
```

```
Spec == Init /\ [] [Next]_vars
```

# Outline

1 TLA<sup>+</sup>

2 TLA<sup>+2</sup>

3 Demo

4 Conclusion and Future work

TLA<sup>+</sup>2 extends TLA<sup>+</sup> to add proofs.

We use a declarative proof language with explicit structure.

Cooperation with Georges Gonthier.

Tools :

- Proof Manager : translates proofs into Isabelle
- Zenon : automatic theorem prover
- Isabelle/TLA<sup>+</sup> : formalization of the TLA<sup>+</sup>2 logic in Isabelle
- GUI

## TLA<sup>+</sup> example

```
THEOREM Inductive == TypeOK /\ I /\ Next => I'  
  <1>1. ASSUME TypeOK, I, NEW i \in {0,1}, proc(i)  
    PROVE I'  
  <2>0. TypeOK'  
    BY TypeOK, Next, TypeCorrect  
  <2>1. ASSUME NEW j \in {0,1},  
    pc'[j] \in {"a2", "a3", "cs", "a4"}  
    PROVE flag'[j]  
  <3>a0. CASE a0(i)  
  <3>a1. CASE a1(i)  
  <3>a2. CASE a2(i)  
  <3>9. QED  
  <2>9. QED  
  <1>9. QED  
  BY <1>1 DEF Inductive, Next
```

# Outline

1 TLA<sup>+</sup>

2 TLA<sup>+2</sup>

**3 Demo**

4 Conclusion and Future work

# Demo

# Outline

- 1 TLA<sup>+</sup>
- 2 TLA<sup>+2</sup>
- 3 Demo
- 4 Conclusion and Future work

# Conclusion

- TLA<sup>+</sup>2 proof language
- Proof Manager prototype
- Zenon : Isabelle back-end and TLA proof rules
- GUI embryo

## Work in progress

- Handle strings, records, arithmetic
- Make Zenon more powerful
- Handle temporal logic
- Interface with veriT
- Develop the GUI into a full-fledged IPE

Thanks for your attention