TLA+

TLC

Presented by
Rui Fan
Stanislav Funiac
Mandana Vaziri

6.897 Spring 2001

Outline

Overview of TLA/TLA+
Subset of TLA+ supported by TLC
Alternating Bit Protocol example
Model checking
Demo

L. Lamport, *Specifying Concurrent Systems with TLA+*, Dec. 2000.

Overview

TLA+
- stylized way of expressing TLA models

TLC
- takes subset of TLA+ & configuration file
- invariance, step simulation, temporal checking
- outputs no error, or counterexample trace

TLA

TLA model

ϕ = init $\wedge$ $[M]_{<vars>} \wedge$ Temporal

$M = A1 \vee A2 \vee \ldots \vee An$

Semantics

$<s_0, s_1, \ldots >[[\phi]] = s_0 [[\text{init}]] \wedge$

$\forall n: s_n [[M \vee (vars' = vars)]] s_{n+1} \wedge$

$<s_0, s_1, \ldots >[[\text{Temporal}]]$

TLA (cont.)

Property to check P

$\phi \Rightarrow$ P

Refinement Mapping ψ

$\phi \Rightarrow \psi$

TLA+

```
----------Module M ----------
EXTENDS     M1, ... ,Mn
CONSTANTS C1, ... , Cn
VARIABLES   x1, ... , xn
ASSUME      P
F (x1, ... , xn) == exp
f [x \in S]      == exp
-----------------------------------------
THEOREM P
=========================
```

Small Example

```
---------- MODULE HourClock ----------
EXTENDS  Naturals
VARIABLE hr
HCini  ==  hr \in (1 .. 12)
HCnxt ==  hr'= IF hr # 12 THEN hr + 1 ELSE 1
HC     ==  HCini /\   [HCnxt]_hr
------------------------------------------
THEOREM HC =>   Hcini
==========================================
```


Acceptable Subset of TLA+: Values

TLC values:

- bool, int, string, model values
- finite set of comparable TLC values
- function with TLC domain and range

Acceptable Subset of TLA+: Expressions

Left to right evaluation:

- $(x \# <>) \land (x[1] = 0)$ valid
- $(x[1] = 0) \land (x \# <>)$ not valid

Sets:

- $\{i \in 0 .. 5 : i < 4\}$ valid
- $\{i \in Nat : i < 4\}$ not valid

Functions:

- $[n \in Nat \rightarrow n * (n+1)][3]$ valid
- $[n \in Nat \rightarrow n * (n+1)]$ not valid

Acceptable Subset of TLA+: Actions

Left to right evaluation

$A \equiv x' \in 1 .. Len(y) \land$
$\quad y' = Append(Tail(y), x')$ valid

$A \equiv y' = Append(Tail(y), x') \land$
$\quad x' \in 1 .. Len(y)$ not valid

Actions (cont.)

Evaluation of all branches:

$A \equiv ((x = <>) \lor (x[1] = 0))$
$\quad \land x' = Append(x,0)$ not valid

Evaluated as 2 actions:

$(x = <>)$ $(x[1] = 0)$
$\land x' = Append(x,0)$ $\land x' = Append(x,0)$

$A \equiv ((x = <>) \lor$
$\quad (x \# <>) \land (x[1] = 0))$
$\quad \land x' = Append(x,0)$ valid

Examples

Alternating Bit Protocol

- Chapter 13
- tlatk/examples/alternatingbit/

Caching Memory

- Chapter 5
- tlatk/examples/fm99/

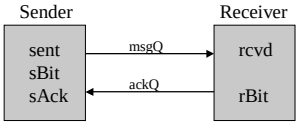
Alternating Bit Protocol

Protocol description
TLA+ specification
Configuration file

13

Alternating Bit Protocol

Sending messages over lossy channel



14

ABP: protocol description

Sender A, receiver B
A repeatedly sends the current message tagged with a protocol bit (sBit)
B acks with the same bit
A moves on to the next message once it receives sAck=sBit and flips sBit

15

ABP: TLA+ specification

MODULE AlternatingBit
CONSTANT Data
VARIABLES msgQ, ackQ, sBit, sAck, rBit, sent, rcvd

ABInit (initial condition)
TypeInv (type-correctness invariant)
ABNext (next-state relation)
Inv (invariant)

16

ABP: configuration file

Specifies the names of
- the initial predicate
- next-state relation
- invariants
- the temporal formula

Assigns values to the constants
Further constrains the model to make it finite

17

ABP: configuration file (cont.)

MCAlternatingBit.tla file
MCAlternatingBit.cfg file:

```
CONSTANTS  Data    = {d1, d2}
           msgQLen = 2
           ackQLen = 2
           sentLen = 3
INIT       ABInit
NEXT       ABNext
INVARIANTS Inv
           TypeInv
CONSTRAINT SeqConstraint
```

18

TLC Capabilities

Can check these types of conditions:

- initial condition: Init => ImpliedInit
- invariance: $Init \wedge [Next]_{<vars>} \Rightarrow Invariant$
- step simulation: $Init \wedge [Next]_{<vars>} \Rightarrow [ImpliedAction]_{<vars>'}$
- temporal: $Init \wedge [Next]_{<vars>} \wedge Temporal \Rightarrow ImpliedTemporal$

19

Model Checking

A *state* is an assignment of values to variables.

First generate all initial states by enumerating all possible initial assignments.

Do BFS for new states.

- For each state *s*, action *A*, check if *A* on *s* leads to a new state.
- Keep track of states already seen.
- If invariant or temporal properties is violated, TLC stops and reports the error.

Stop when all states lead to seen states.

20

Internals

Memory management

- Bit-state hashing
- Symmetry wrt variable permutations
- Writes states to disk, uses prefetch for access

Written in Java, multi-threaded

Models are interpreted, not compiled

21

TLC Limitations

Cannot check some temporal formulas.

e.g. $EvenSpec \equiv (x=0) \wedge [x'=x+2]_x \wedge WF_x(x'=x+2)$

$OddProp \equiv WF_x(x'=x+2) \Rightarrow \Diamond(x=1)$

OddProp is false, but TLC doesn't report an error.

TLC only generates finite behaviors, so $WF_x(x'=x+2)$ is false, and OddProp is true.

Note EvenSpec isn't finite state.

22

Conclusion

Advantages:

- TLA+ expressive, relatively simple
- multiple levels of abstraction, refinement mapping
- infinite state models

http://theory.lcs.mit.edu/~rfan/tlc.tar

Disadvantages:

- no uniform way of making TLA+ models suitable for TLC
  - configuration files
  - order of evaluation
  - evaluation of all branches
- sublanguage bound to analysis method

23