Reasoning with TLA+

Gowtham Kaki
Praseem Banzal
Suvidha Kancharla

PURDUE
UNIVERSITY

Written Assignment-1 Question

- 3. Take the Causal Atomic Broadcast seen in class
  - a. Does it implement Uniform Causal Order? Why (not)?
  - b. Suppose the underlying FIFO Atomic Broadcast algorithm provides Uniform Agreement instead of Agreement (i.e., I, II, III, V, IX, XIII instead of I, II, III, IV, IX, XIII). Does the resulting algorithm also provide Uniform Agreement? Can we simplify it?

Written Assignment-1 Question

- 3. Take the Causal Atomic Broadcast seen in class
  - a. Does it implement Uniform Causal Order? Why (not)?
  - b. Suppose the underlying FIFO Atomic Broadcast algorithm provides Uniform Agreement instead of Agreement (i.e., I, II, III, V, IX, XIII instead of I, II, III, IV, IX, XIII). Does the resulting algorithm also provide Uniform Agreement? Can we simplify it?
- Ans : No. There exists a counter-example.
- Ok, but how do we arrive at this conclusion? Trial and error?

TLA

- TLA - Temporal Logic of Actions (Lamport, 1993).
- A logic that lets us formally (i.e., methodically) reason about temporal (i.e., time-related) properties of actions (i.e, state-changing operations).
- In other words, it is a logic that lets us
  - Abstractly describe the behaviour of concurrent algorithms (eg: FIFO reliable broadcast) as a function of time.
  - Precisely state its properties, such as FIFO order and validity, whose English descriptions contain words like "before", "after", "never" and "eventually".
  - Formally prove that the algorithm satisfies its specification.

TLA+ and Paraphernalia

- Writing specifications, let alone algorithms, directly in a logic is very tedious for humans. Fortunately, tools exist:
  - TLA+ : A specification language built on TLA+. Lets us *name* formulas, as programmers name expressions in a programming language.
  - PlusCal (written CAL+) : An *algorithm language*. Lets us describe concurrent algorithms as pseudo-code. CAL+ compiles to TLA+, sparing us the burden of describing algorithms in logic.
- Checking/proving specifications is even harder. Again, tools exist:
  - TLC: A model checker for TLA+ specifications. Tries to find counter-examples to disprove specifications by exhaustive search.

Example - Specifying Hour Clock

```
---------------------- 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
==============================================================
```

- Observe that:
  - The behaviour of our system - hour clock, is succinctly captured by the formula HC
  - HC has the form : initial-state-predicate $\wedge$ next-state-predicate
  - The theorem asserts that behaviour of hour-clock guarantees the property that $hr \in \{1 .. 12\}$

Checking Hour-Clock with TLC

```
gowtham@tycon:~/git/tla-tools/tla/examples/SpecifyingSystems/HourClock$ java tlc2.TLC HourClock.tla
TLC2 Version 2.05 of 23 July 2013
Running in Model-Checking mode.
Parsing file HourClock.tla
Parsing file /Users/gowtham/git/tla-tools/tla/tla2sany/StandardModules/Naturals.tla
Semantic processing of module Naturals
Semantic processing of module HourClock
Starting... (2014-04-01 00:30:38)
Computing initial states...
Finished computing initial states: 12 distinct states generated.
Model checking completed. No error has been found.
  Estimates of the probability that TLC did not check all reachable states
  because two distinct states had the same fingerprint:
  calculated (optimistic):  val = 7.8E-18
  based on the actual fingerprints:  val = 5.1E-18
24 states generated, 12 distinct states found, 0 states left on queue.
The depth of the complete state graph search is 1.
Finished. (2014-04-01 00:30:43)
```

Hour Clock - 2

```
---------------------- MODULE HourClock ----------------------
EXTENDS Naturals
VARIABLE hr
HCini  ==  hr \in (1 .. 12)
HCnxt  ==  hr' = IF hr # 23 THEN hr + 1 ELSE 0
HC  ==  HCini /\ [][HCnxt]_hr
--------------------------------------------------------------
THEOREM  HC => []HCini
==============================================================
```

- Observe that:
  - Hour-clock now behaves as a 24-hour clock.
  - Yet, the specification requires that $hr \in \{1 \,..\, 12\}$
  - Theorem is invalid.

Checking Hour Clock-2

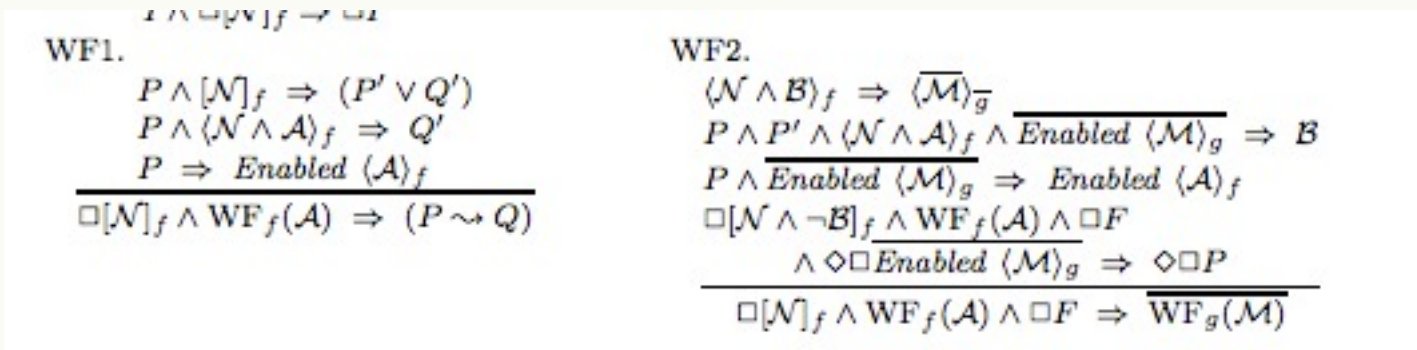
```
Computing initial states...
Finished computing initial states: 12 distinct states generated.
Error: Invariant HCini is violated.
Error: The behavior up to this point is:
State 1: <Initial predicate>
hr = 12

State 2: <Action line 5, col 12 to line 5, col 46 of module HourClock>
hr = 13

24 states generated, 13 distinct states found, 1 states left on queue.
```

Proving TLA+ Specification

- The scope of model-checking is limited. Our systems, in general, have infinite state-space. We may have to prove things in the logic of TLA.
- TLA proof rules are complex! Also, not algorithmic.



WF1.

$$\frac{\begin{array}{l} P \wedge [\mathcal{N}]_f \Rightarrow (P' \vee Q') \\ P \wedge \langle \mathcal{N} \wedge \mathcal{A} \rangle_f \Rightarrow Q' \\ P \Rightarrow \mathit{Enabled}\ \langle \mathcal{A} \rangle_f \end{array}}{\Box[\mathcal{N}]_f \wedge \mathrm{WF}_f(\mathcal{A}) \Rightarrow (P \rightsquigarrow Q)}$$

WF2.

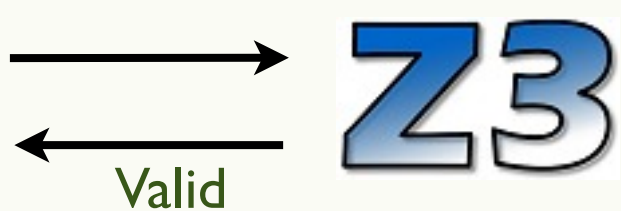
$$\frac{\begin{array}{l} \langle \mathcal{N} \wedge \mathcal{B} \rangle_f \Rightarrow \langle \overline{\mathcal{M}} \rangle_{\overline{g}} \\ P \wedge P' \wedge \langle \mathcal{N} \wedge \mathcal{A} \rangle_f \wedge \overline{\mathit{Enabled}\ \langle \mathcal{M} \rangle_g} \Rightarrow \mathcal{B} \\ P \wedge \overline{\mathit{Enabled}\ \langle \mathcal{M} \rangle_g} \Rightarrow \mathit{Enabled}\ \langle \mathcal{A} \rangle_f \\ \Box[\mathcal{N} \wedge \neg \mathcal{B}]_f \wedge \mathrm{WF}_f(\mathcal{A}) \wedge \Box F \\ \qquad \wedge \Diamond\Box \overline{\mathit{Enabled}\ \langle \mathcal{M} \rangle_g} \Rightarrow \Diamond\Box P \end{array}}{\Box[\mathcal{N}]_f \wedge \mathrm{WF}_f(\mathcal{A}) \wedge \Box F \Rightarrow \overline{\mathrm{WF}_g(\mathcal{M})}}$$

- Several other rules like above.
- TLAPS (TLA Proof System) is a tool that would assist us in writing such proofs, but it gives only basic support.

Can we automate?

```
---------------------- 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
==============================================================
```

```
hr[0] ∈ {0 .. 12}
∀t.  (hr[t] ≠ 12 ⇒ hr[t+1] = hr[t]+1)
   ∧ (hr[t] = 12 ⇒ hr[t+1]=1)
------------------------------------
∀t. hr[t] ∈ {0 .. 12} ??
------------------------------------
```



How far can we stretch this approach?

Need for Proof - Most Wanted?

- More fundamental question - For distributed algorithms, such as FIFO broadcast, is a proof really needed to answer such questions as written assignment problem?
- Doesn't model-checking suffice to answer these questions?
- Remember that formally proving properties, even when automated, is still very hard; this time for computers. Decidable subset of first-order logic is NEXPTIME complete.
- Cost of proving things should justify benefits.

Course Project

- Write CAL+ implementations, and the specifications of their interfaces for three distributed algorithms:
  - Reliable Broadcast
  - FIFO Reliable Broadcast
  - Causal-Order Reliable Broadcast
- Refine specifications until TLC cannot find bugs for finite-instances of the problem. This step involves constructing right models for TLC to work on.
- See if model-checking can find answers to questions such as : Does adding uniform agreement to reliable broadcast provide uniform agreement at FIFO reliable broadcast?
- Provide an experience report quantifying the effort and identifying subtle problems in writing TLA+ specs of distributed algorithms.

Course Project

- Only if time permits:
  - For one algorithm, say reliable broadcast, identify those properties which can be proved automatically (using an SMT solver) by reasoning in first-order logic. Give SMT-LIB encoding of such properties, and statistics on amount of time that solver took to discharge proof obligations.