Using Reachability Logic to Verify Distributed Systems

Stephen Skeirik, Andrei Stefanescu,
and José Meseguer

ACC Weekly Research Seminar
Nov. 2, 2016

Outline

- Building complex cloud-based systems is challenging
  - many more *failure modes* than traditional servers
  - often *combinatorial explosion* in size of state space
- Testing *alone* is *inadequate* for verifying cloud-based systems
  - can only cover *finite* part of the state space
  - testing requires *essentially complete* artifacts, yet early design mistakes most costly
  - requirements *evolve* with system

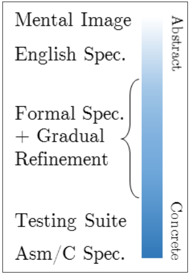
Q: What methods can assist designing/verifying cloud systems?

Q: What methods can assist designing/verifying cloud systems?

A: We propose *formal specification/verification methods*

Goal: a *precise*, *readable*, *testable*, *refinable* version of our mental model

Essentially, a kind of logic/high-level *declarative* programming language



Q: What *frameworks* to use for formal specification/verification? How to use formal methods to *specify/verify* systems?

Q: What frameworks to use for formal specification/verification?

A: One possible option is *Rewriting Logic* (RWL), because it is:

- *Expressive*—models complex systems with few K lines
- *Modular*—supports parameterization by theories
- *Executable*—can be tested; several interpreters available

Q: How to use formal methods to specify/verify systems?

A: Specify via *declarative programming*; verify via either
Model checking: automatable, finiteness requirements or
Deductive methods: less automatable, admits most systems

Previous ACC projects include analysis/verification of...

- Apache Zookeeper-based Group Key Management
- Google Megastore and extension Megastore-CGC
- Apache Cassandra
- RAMP (Read-Atomic-Multi-Partition) Systems

Q: What do the above four projects have in *common*?

A: (1) They are all *distributed databases*

(2) Analysis framework was *Rewriting Logic*

(3) They were all analyzed via *model checking*

Q: These systems are clearly infinite state.
How can they be model-checked?

A: A few simplifications were necessary:

- Bound all non-determinism
- Abstract away non-essential behaviors
- Consider a finite set of initial configurations

Q: Model checking already provides high assurance;
However, it cannot cover all possible configurations.
What methods might provide even greater assurance?

A: We believe *Reachability Logic* (RL) is one such method.

Outline

Rewriting Logic

Q: What is Rewriting Logic?

A: A logic of *distributed states* and *concurrent interactions*

Ex: A mutual exclusion alg. QLOCK specified by 3 rewrite rules:

$$
\begin{aligned}
n2w &: < n\ i \mid w \mid c \mid q > \;\to\; < n \mid w\ i \mid c \mid q ; i > \\
w2c &: < n \mid w\ i \mid c \mid i ; q > \;\to\; < n \mid w \mid c\ i \mid i ; q > \\
c2n &: \underbrace{\underbrace{< n \mid w \mid c\ i \mid i ; q >}_{\textit{State/Term}} \;\to\; < n\ i \mid w \mid c \mid q >}_{\textit{Concurrent Interaction/Rule}}
\end{aligned}
$$

QLOCK can be formalized by set of *rewrite rules* $R = \{l_i \to r_i\}_{i \in I}$

Concrete instances of rewrite rules are called *states/terms*
Rules induce *transition system* over concrete instances

Example transition (*rewrite step*):

$$n2w : < n\ \ i \mid w \quad \mid c \quad \mid q \quad\quad > \ \to\ < n \quad \mid w\ \ i \mid c \quad \mid q\ ;\ i >$$

$$\searrow \qquad\qquad\qquad\qquad\qquad \searrow$$

$$< 1\ 3\ 2 \mid mt \mid mt \mid nil > \to < 1\ 2 \mid 3 \mid mt \mid 3 >$$

A *rewrite path* is a sequence $s_0 \to s_1 \to \cdots \to s_{i-1} \to s_i$

Q: How to define symbolic states?

A: Recall set-builder notation from mathematics. Ex:

$$Even = \{x \in \mathbb{N} \mid x \ mod \ 2 = 0\}$$

We can also *constrain* terms. Let $dupl(s \ s \ s_0) = tt$. Ex:

$$\underbrace{< n \mid w \mid c \mid q >}_{\textit{term}} \ \Big| \ \underbrace{dupl(n \ w \ c) \neq tt}_{\textit{constraint}}$$

This constrained term refers to states *without* duplicate ids

Reachability Logic (RL) is:

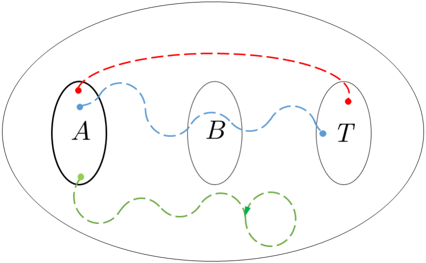
- parameterized over an underlying *rewrite theory* $\mathcal{R}$
- considers formulas $A \longrightarrow^{\circledast} B$ between *constrained terms* A, B
- a generalization of Hoare Logic *partial correctness*,
  i.e. can view sequents $A \longrightarrow^{\circledast} B$ as $\{A\}\mathcal{R}\{B\}$
- directly captures *circular* behavior in *any* theory $\mathcal{R}$,
  unlike Hoare Logic, special rules for loops, etc, *unnecessary*

Q: What does the relation $A \longrightarrow^{\circledast} B$ mean?

A: Suppose we have:

(1) a rewrite theory $\mathcal{R}$
(2) constrained terms A, B
(3) and terminating states T

Then $A \longrightarrow^{\circledast} B$ means:
for each concrete t in A,
for each t rewrite path p,
either: (1) p crosses B
(2) p is infinite



- - - indicates counterex.
- - - satisfies $A \rightarrow^{\circledast} B$
- - - *vacuously* satisfies

Q: Then given RWL theory $\mathcal{R}$, how do we prove $A \longrightarrow^{\circledast} B$?

A: Perhaps surprisingly, two proof rules are enough

- A rule that traces *rewrite steps* of *symbolic* states in $\mathcal{R}$
- A rule that captures *circular behavior* of $\mathcal{R}$

We call these two rules *Step+Subsumption* and *Axiom* resp.

Reachability Logic

Proof Rules

$$\frac{\displaystyle\bigwedge_{(j,\alpha)\in\textsc{unify}(u|\varphi',\ R)} [\mathcal{A}\cup\mathcal{C},\ \emptyset]\ \vdash_T\ (r_j \mid \varphi' \wedge \phi_j)\alpha \longrightarrow^{\circledast} \bigvee_i (v_i \mid \psi_i)\alpha}{[\mathcal{A},\ \mathcal{C}]\ \vdash_T\ u \mid \varphi \longrightarrow^{\circledast} \bigvee_i v_i \mid \psi_i}$$

$$\frac{\displaystyle\bigwedge_j [\{u' \mid \varphi' \longrightarrow^{\circledast} \bigvee_j v'_j \mid \psi'_j\} \cup \mathcal{A},\ \emptyset]\ \vdash_T\ v'_j\alpha \mid \varphi \wedge \psi'_j\alpha \longrightarrow^{\circledast} \bigvee_i v_i \mid \psi_i}{[\{u' \mid \varphi' \longrightarrow^{\circledast} \bigvee_j v'_j \mid \psi'_j\} \cup \mathcal{A},\ \emptyset]\ \vdash_T\ u \mid \varphi \longrightarrow^{\circledast} \bigvee_i v_i \mid \psi_i}$$

The *Step+Subsumption* and *Axiom* Rules

Outline

Current Progress

Q: So what work has been done already?

A: A substantial RL framework is already in place with:

- full semantics for RL developed in terms of RWL
- soundness proof for proof system and semantics
- working Maude prototype of proof system
- a small but growing collection of case studies

Current Progress

Case Studies

Example Name	# Goals/Lemmas
Thermostat	4
Bounded Retransmission Protocol	1
Dijkstra's Mutual Ex. Alg.	4
Fault-Tolerant Comm. Protocol	6
QLOCK Mutual Ex. Alg.	3
Readers/Writers	3
Fixed-Size Token Ring	3
Unbounded Lamport's Bakery	11
IMP Swap Function	2
IMP Min Function	2
IMP Max Function	2

Outline

Future Directions

Q: So, how to scale up to more complex cloud-based systems?

- No one right answer; by way of example...
  let's intuitively sketch a fragment of Apache Cassandra

Q: So what is Apache Cassandra?

Future Directions

Apache Cassandra

Q: So what is Apache Cassandra?

A: A distributed key value store where:

- peers are *equal*; no master/slave
- data is copied among user-configurable # of *replicas*
- writes are *cheap*; inconsistencies handled during reads
- data tagged with timestamp; merges by *last-write-wins*

Q: Does Cassandra satisfy *eventual consistency*?
How to RL to *verify* it satisfies eventual consistency?

Two rewrite rules: request generation/server request consumption

$$< wr\ [ts, k, v] \mid m \mid \{i \mid ks\}\ s \mid c > \rightarrow$$
$$< wr \mid i \leftarrow [c + d \mid ts, k, v]\ m \mid \{i \mid ks\}\ s \mid c >$$

$$< nil \mid i \leftarrow [c \mid ts, k, v]\ m \mid \{i \mid ks\}\ s \mid c > \rightarrow$$
$$< nil \mid fwd(s, ts, k, v) \quad m \mid \{i \mid ins(ts, k, v, ks)\}\ s \mid c >$$

- Assume an initial state $A =< wr \mid mt \mid s \mid c >$
- A consistency violation can be specified by
  $cv(\{i \mid k = v : ks\}\ \{j \mid k = v' : ks\}\ s) = tt$
- To show eventual consistency, it is enough to prove
  $A \longrightarrow^{\circledast} < nil \mid mt \mid s' \mid c' > \Big| \ cv(s') \neq tt$

Future Directions

There are several concrete *action steps* in the pipeline

- Support constrained terms over *undecidable theories*
- Increase automation by integrating *inductive theorem prover*
- Implement *optimizations* for built-in satisfiability methods
- Further *exploit* above techniques in analyzing cloud systems

The End

Questions?

Example
Verifying QLOCK

$$
\begin{array}{rl}
n2w: & < n\ i \mid w \quad \mid c \quad \mid q \quad > \;\rightarrow\; < n \quad \mid w\ i \mid c \quad \mid q\ ;\ i > \\
w2c: & < n \quad \mid w\ i \mid c \quad \mid i\ ;\ q > \;\rightarrow\; < n \quad \mid w \quad \mid c\ i \mid i\ ;\ q > \\
c2n: & < n \quad \mid w \quad \mid c\ i \mid i\ ;\ q > \;\rightarrow\; < n\ i \mid w \quad \mid c \quad \mid q \quad >
\end{array}
$$

Mutual exclusion property $P = < n \mid w \mid c \mid q > \mid size(c) \leq 1$
Would like to show $< n \mid mt \mid mt \mid nil > \longrightarrow^{\circledast} P$

But since QLOCK *never* terminates, it is impossible to verify...

Q: What can be done?

Example

Verifying QLOCK

Q: What can be done?

A: A simple theory transformation shown below

$$
\begin{array}{rl}
n2w : & < n\ i \mid w \mid c \mid q > \;\to\; < n \mid w\ i \mid c \mid q\ ;\ i > \\
w2c : & < n \mid w\ i \mid c \mid i\ ;\ q > \;\to\; < n \mid w \mid c\ i \mid i\ ;\ q > \\
c2n : & < n \mid w \mid c\ i \mid i\ ;\ q > \;\to\; < n\ i \mid w \mid c \mid q > \\
term : & < n \mid w \mid c \mid q > \;\to\; [\, n \mid w \mid c \mid q \,]
\end{array}
$$

New mutual ex. property $[P] = [n \mid w \mid c \mid q] \mid size(c) \leq 1$
Then prove $< n \mid mt \mid mt \mid nil > \mid dupl(n) \neq tt \longrightarrow^{\circledast} [P]$
Needed lemma $< n \mid w \mid c \mid q > \mid size(c) \leq 1 \longrightarrow^{\circledast} [P]$