Using Reachability Logic to Verify Distributed Systems

Stephen Skeirik, Andrei Stefanescu, and José Meseguer

ACC Weekly Research Seminar
Nov. 2, 2016
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* (R WL), 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.
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:

\[
\begin{align*}
  n2w &: < n \ | \ i \ | \ w \ | \ c \ | \ q > \rightarrow < n \ | \ w \ | i \ | \ c \ | \ q \ ; \ i > \\
  w2c &: < n \ | \ w \ | i \ | \ c \ | \ i \ ; \ q > \rightarrow < n \ | \ w \ | c \ | i \ | i \ ; \ q > \\
  c2n &: < n \ | \ w \ | c \ | i \ | i \ ; \ q > \rightarrow < n \ | \ i \ | w \ | c \ | q > 
\end{align*}
\]
QLOCK can be formalized by set of *rewrite rules* \( R = \{ l_i \rightarrow r_i \}_{i \in I} \)

Concrete instances of rewrite rules are called *states/terms*

Rules induce *transition system* over concrete instances

Example transition (*rewrite step)*:

\[
\begin{align*}
\text{n2w}: & < n \ i \ | \ w \ | \ c \ | \ q > \rightarrow < n \ | \ w \ i \ | \ c \ | \ q \ ; \ i > \\
& \downarrow \downarrow \\
& < 1 \ 3 \ 2 \ | \ mt \ | \ mt \ | \ nil > \rightarrow < 1 \ 2 \ | \ 3 \ | \ mt \ | \ 3 >
\end{align*}
\]

A *rewrite path* is a sequence \( s_0 \rightarrow s_1 \rightarrow \cdots \rightarrow s_{i-1} \rightarrow s_i \)
Q: How to define symbolic states?

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

\[ \text{Even} = \{ x \in \mathbb{N} \mid x \mod 2 = 0 \} \]

We can also constrain terms. Let \( \text{dupl}(s \ s \ s_0) = \text{tt} \). Ex:

\[
\begin{align*}
\langle n \mid w \mid c \mid q \rangle & \quad \text{term} \\
\text{dupl}(n \ w \ c) & \neq \text{tt} \quad \text{constraint}
\end{align*}
\]

This constrained term refers to states \textit{without} duplicate ids.
Reachability Logic (RL) is:

- parameterized over an underlying *rewrite theory* $\mathcal{R}$
- considers formulas $A \rightarrow^\ast B$ between *constrained terms* $A, B$
- a generalization of Hoare Logic *partial correctness*, i.e. can view sequents $A \rightarrow^\ast B$ 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 $A \rightarrow^\ast B$ mean?

A: Suppose we have:

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

Then $A \rightarrow^\ast B$ 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 $A \rightarrow^\ast B$
- - - vacuously satisfies
Q: Then given RWL theory $\mathcal{R}$, how do we prove $A \rightarrow^* B$?

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

\[ \bigwedge_{(j,\alpha) \in \text{UNIFY}(u|\varphi', R)} [A \cup C, \emptyset] \vdash_T (r_j \mid \varphi' \land \phi_j)\alpha \rightarrow^* \bigvee_i (v_i \mid \psi_i)\alpha \]

\[ [A, C] \vdash_T u \mid \varphi \rightarrow^* \bigvee_i v_i \mid \psi_i \]

\[ \bigwedge_{j} \{ u' \mid \varphi' \rightarrow^* \bigvee v'_j \mid \psi'_j \} \cup A, \emptyset \vdash_T v'_j\alpha \mid \varphi \land \psi'_j\alpha \rightarrow^* \bigvee_i v_i \mid \psi_i \]

\[ \{ u' \mid \varphi' \rightarrow^* \bigvee v'_j \mid \psi'_j \} \cup A, \emptyset \vdash_T u \mid \varphi \rightarrow^* \bigvee_i v_i \mid \psi_i \]

The \textit{Step+Subsumption} and \textit{Axiom} Rules
Outline

1. Introduction
2. Reachability Logic
3. Current Progress
4. Future Directions
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
<table>
<thead>
<tr>
<th>Example Name</th>
<th># Goals/Lemmas</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thermostat</td>
<td>4</td>
</tr>
<tr>
<td>Bounded Retransmission Protocol</td>
<td>1</td>
</tr>
<tr>
<td>Dijkstra’s Mutual Ex. Alg.</td>
<td>4</td>
</tr>
<tr>
<td>Fault-Tolerant Comm. Protocol</td>
<td>6</td>
</tr>
<tr>
<td>QLOCK Mutual Ex. Alg.</td>
<td>3</td>
</tr>
<tr>
<td>Readers/Writers</td>
<td>3</td>
</tr>
<tr>
<td>Fixed-Size Token Ring</td>
<td>3</td>
</tr>
<tr>
<td>Unbounded Lamport’s Bakery</td>
<td>11</td>
</tr>
<tr>
<td>IMP Swap Function</td>
<td>2</td>
</tr>
<tr>
<td>IMP Min Function</td>
<td>2</td>
</tr>
<tr>
<td>IMP Max Function</td>
<td>2</td>
</tr>
</tbody>
</table>
Outline

1. Introduction
2. Reachability Logic
3. Current Progress
4. 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?
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?
Future Directions
Modelling Apache Cassandra

Two rewrite rules: request generation/server request consumption

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

\[
< \text{nil} \ | \ i \leftarrow [c \ | \ ts, k, v] \ m \ | \ \{i \ | \ ks\} \ s \ | \ c > \rightarrow \\
< \text{nil} \ | \ \text{fwd}(s, ts, k, v) \ m \ | \ \{i \ | \ \text{ins}(ts, k, v, ks)\} \ s \ | \ c >
\]
Assume an initial state $A = < wr | mt | s | c >$

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

$$A \xrightarrow{⊗} < nil | mt | s' | c' > \mid cv(s') \neq tt$$
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
Questions?
Example
Verifying QLOCK

\begin{align*}
n2w & : < n \mid w \mid c \mid q > \rightarrow < n \mid w_1 \mid c \mid q ; i > \\
w2c & : < n \mid w \mid c \mid i ; q > \rightarrow < n \mid w \mid c_i \mid i ; q > \\
c2n & : < n \mid w \mid c_i \mid i ; q > \rightarrow < n_i \mid w \mid c \mid q > \\
\end{align*}

Mutual exclusion property \( P = < n \mid w \mid c \mid q > \mid \text{size}(c) \leq 1 \)

Would like to show \( < n \mid mt \mid mt \mid nil > \rightarrow^* P \)

But since QLOCK \textit{never} terminates, it is impossible to verify...

\textbf{Q}: What can be done?
Q: What can be done?

A: A simple theory transformation shown below

\[
\begin{align*}
n2w &: < n \mid w \mid c \mid q > \rightarrow < n \mid w i \mid c \mid q ; i > \\
w2c &: < n \mid w i \mid c \mid i ; q > \rightarrow < n \mid w \mid c i \mid i ; q > \\
c2n &: < n \mid w \mid c i \mid i ; q > \rightarrow < n i \mid w \mid c \mid q > \\
term &: < n \mid w \mid c \mid q > \rightarrow [ n \mid w \mid c \mid q ]
\end{align*}
\]

New mutual ex. property \([P] = [ n \mid w \mid c \mid q ] \mid \text{size}(c) \leq 1\)

Then prove \(< n \mid mt \mid mt \mid nil > \mid \text{dupl}(n) \neq tt \rightarrow^{\star} [P]\)

Needed lemma \(< n \mid w \mid c \mid q > \mid \text{size}(c) \leq 1 \rightarrow^{\star} [P]\)