AFFIRM

Quarterly Meeting

February 15, 2017

Year 3 Plans

- Build a prototype SAL/Sally backend for the ADSL:
  - translation of expression language and message passing semantics
  - configurable hybrid fault model
  - generation of framework specific lemmas (e.g. calendar lemmas)
  - specification of properties
  - generation of observers and abstract state machines
- Specify our case studies in terms of the prototype ADSL and translator
  - OM(1)
  - WBS
  - Multi-level system: BRAIN, TTE, ...
- ...

Progress this Quarter

Configurable hybrid fault model:

Built-in options:

- no faults
- fixed fault mapping
- hybrid fault model with user specified maximum fault assumption

Progress this Quarter

New Channel API:

- ```
  channel :: Name
          -> Type
          -> Atom (ChanInput, ChanOutput)
  ```
- `writeChannel :: ChanInput -> E a -> Atom ()`
- `readChannel :: ChanOutput -> E a`
- `fullChannel :: ChanOutput -> E Bool`
- `consumeChannel :: ChanOutput -> Atom ()`

These calls are now translated into calendar automata actions.
  ```

Progress this Quarter

We added and/or improved synthesis of:

- clock module
- calendar entries
- fault model
- maximum fault assumption

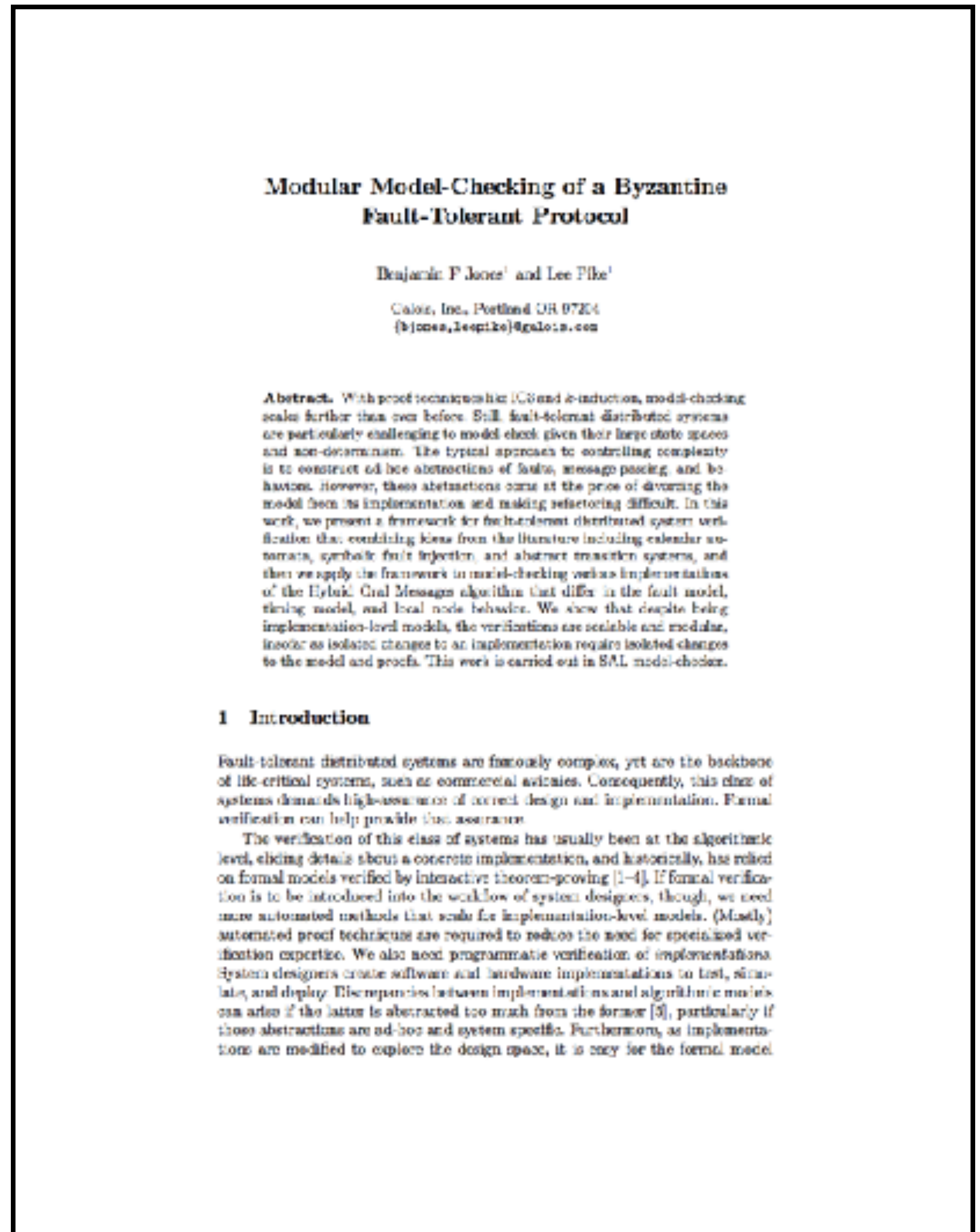
Progress this Quarter

We added a few user-facing features:

- Add simple debugging option to Sally models
- Arithmetic term rewriting to generate smaller models

Progress this Quarter

To appear at
NASA FM, 2017



Modular Model-Checking of a Byzantine Fault-Tolerant Protocol

Benjamin F Jones[1] and Lee Pike[1]

Galois, Inc., Portland OR 97204
{bjones,leepike}@galois.com

Abstract. With proof techniques like IC3 and k-induction, model-checking scales further than ever before. Still, fault-tolerant distributed systems are particularly challenging to model check given their large state spaces and non-determinism. The typical approach to controlling complexity is to construct ad hoc abstractions of faults, message passing, and behaviors. However, these abstractions come at the price of divorcing the model from its implementation and making refactoring difficult. In this work, we present a framework for fault-tolerant distributed system verification that combines ideas from the literature including calendar automata, synthetic fault injection, and abstract transition systems, and then we apply the framework to model-checking various implementations of the Hybrid Oral Messages algorithm that differ in the fault model, timing model, and local node behavior. We show that despite being implementation-level models, the verifications are scalable and modular, insofar as isolated changes to an implementation require isolated changes to the model and proofs. This work is carried out in SAL model-checker.

1 Introduction

Fault-tolerant distributed systems are famously complex, yet are the backbone of life-critical systems, such as commercial avionics. Consequently, this class of systems demands high-assurance of correct design and implementation. Formal verification can help provide that assurance.

The verification of this class of systems has usually been at the algorithmic level, eliding details about a concrete implementation, and historically, has relied on formal models verified by interactive theorem-proving [1–4]. If formal verification is to be introduced into the workflow of system designers, though, we need more automated methods that scale for implementation-level models. (Mostly) automated proof techniques are required to reduce the need for specialized verification expertise. We also need programmatic verification of implementations. System designers create software and hardware implementations to test, simulate, and deploy. Discrepancies between implementations and algorithmic models can arise if the latter is abstracted too much from the former [5], particularly if those abstractions are ad-hoc and system specific. Furthermore, as implementations are modified to explore the design space, it is easy for the formal model

Next Quarter

- specification of <u>temporal properties</u>
- generation of <u>framework lemmas</u> (e.g. calendar lemmas)
- generation of <u>synchronous observers</u> and <u>abstract state machines</u>
- refine our case studies in the ADSL:
  - WBS
  - Multi-level system: BRAIN, TTE, ...