Current Focus

How to synthesize from the ADSL generic model-checking models for distributed systems? That are

- Efficient (small number of state variables)
- Same model for synchronous and partially-synchronous systems
- Verification is mostly automatic

Progress Since Review

- New synthesizable OM(1) model, with faults
- Generic timeout-automata proofs
- Approach for “push-button” model

Efficient Calendar Based OM(1)

- the number of nodes is fixed at 3 relays and 3 receivers (minimum needed to tolerate 1 byzantine fault)

- there are only as many channel variables as there are actual channels (reduces system state size)

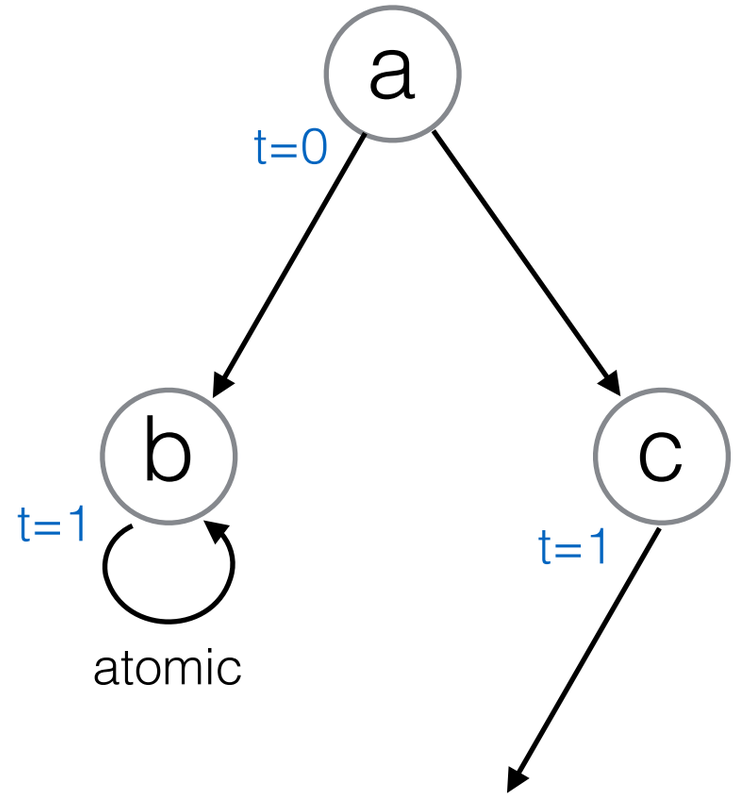
- the "ivote" module from the v3 model has been *flattened* into the receiver modules (could be accomplished automatically during translation)

- a notion of "atomic" transition block was introduced in the calendar automata framework (greatly reduces interleaving of transitions)

Atomic Transition Blocks

- an atomic transition block is a sequence of coordinating transitions that occur **instantaneously**

- we can model local, stateful computations this way and avoid interleaving with network communication and remote computation



Model Checking

- we believe the new model is targetable by automatic translation
- verification is tractable, it takes less than half the time to verify validity and agreement as it did in the previous model with fewer nodes

	v1	v4 current
3 relays 2 recv	279 s	--
3 relays 3 recv	> 2 hrs	91 s

Time for verification

Verification Approach

- Now infinite-state for
  - uninterpreted functions
  - real-time
- Means: need to prove invariants

SAL Fault Model Implementation

- we are experimenting with fault model implementations in SAL

- **first draft**:
  - fault type is assigned to each node at init
  - functions that read messages from the calendar are patched when the sender is faulty
  - messages from faulty nodes are *uninterpreted constants*

Model-Generic Invariants

```
% Global time is positive (or "atomic")
global_time_not_null(t: TIME): BOOLEAN = t /= -1;

% Global time is monotonic
monotonic_time(cal: CALENDAR, t: TIME): BOOLEAN =
 FORALL (s: TIME): is_next_time?(cal, s) => s = atomic_time OR s >= t;

% Messages are handled at their timeout
missing_cal(cal: CALENDAR): BOOLEAN =
  FORALL (i: CHANNEL):
    cal.msg[i] = missing <=> cal.time[i] = invalid_time;

% Only one component executes atomic actions at a time
unique_atomic(cal: CALENDAR): BOOLEAN =
  FORALL (i,j: CHANNEL): cal.time[i] = -2 AND cal.time[j] = -2 => i = j;
```