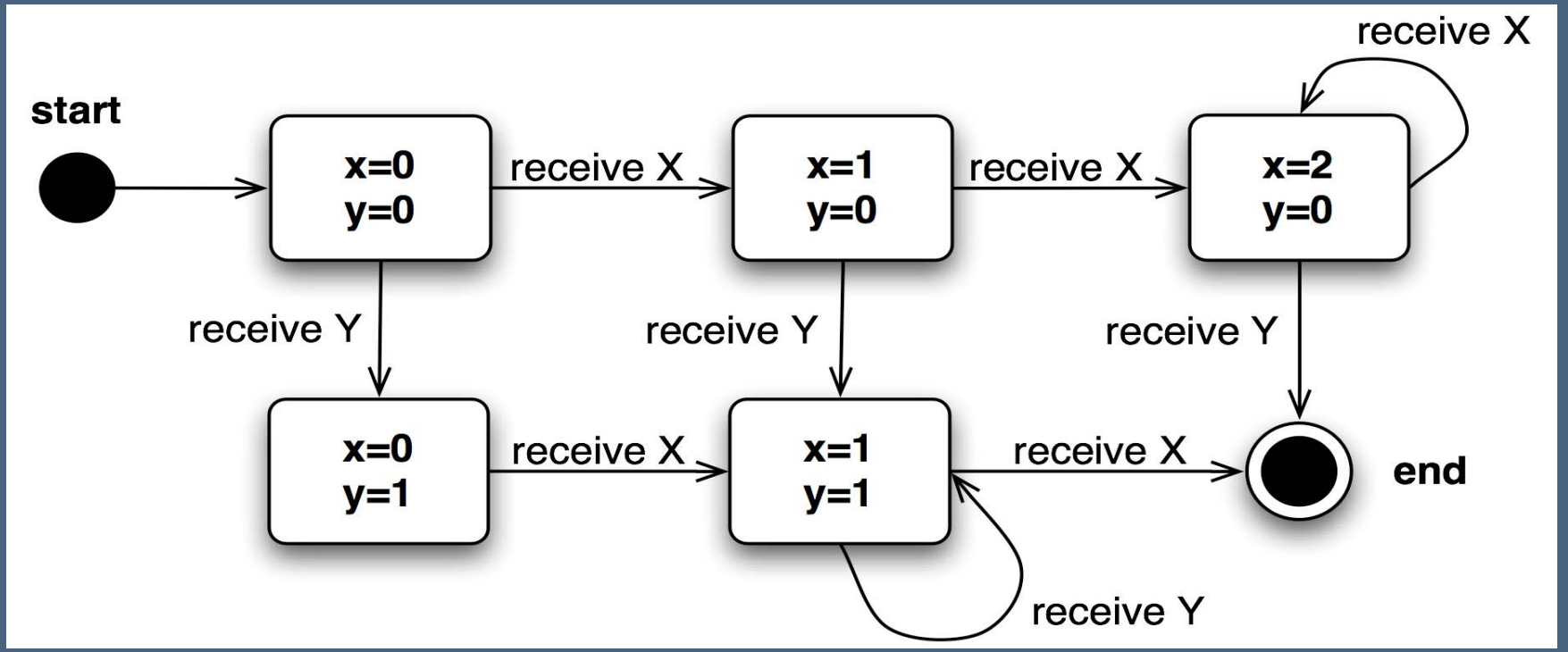
Generating a Family of Byzantine-Fault-Tolerant Protocol Implementations Using a Meta-Model Architecture

Graham Kirby, Alan Dearle & Stuart Norcross

School of Computer Science, University of St Andrews

A Finite State Machine



Problem

- Apply a FSM formulation to an algorithm whose generality precludes its expression as a single state machine
  - algorithm is characterised as a *family* of related state machines
    - each corresponding to particular values of some parameters to the general algorithm
- Family members:
  - differ in their individual states and transitions
  - share a common structure dictated by the general algorithm

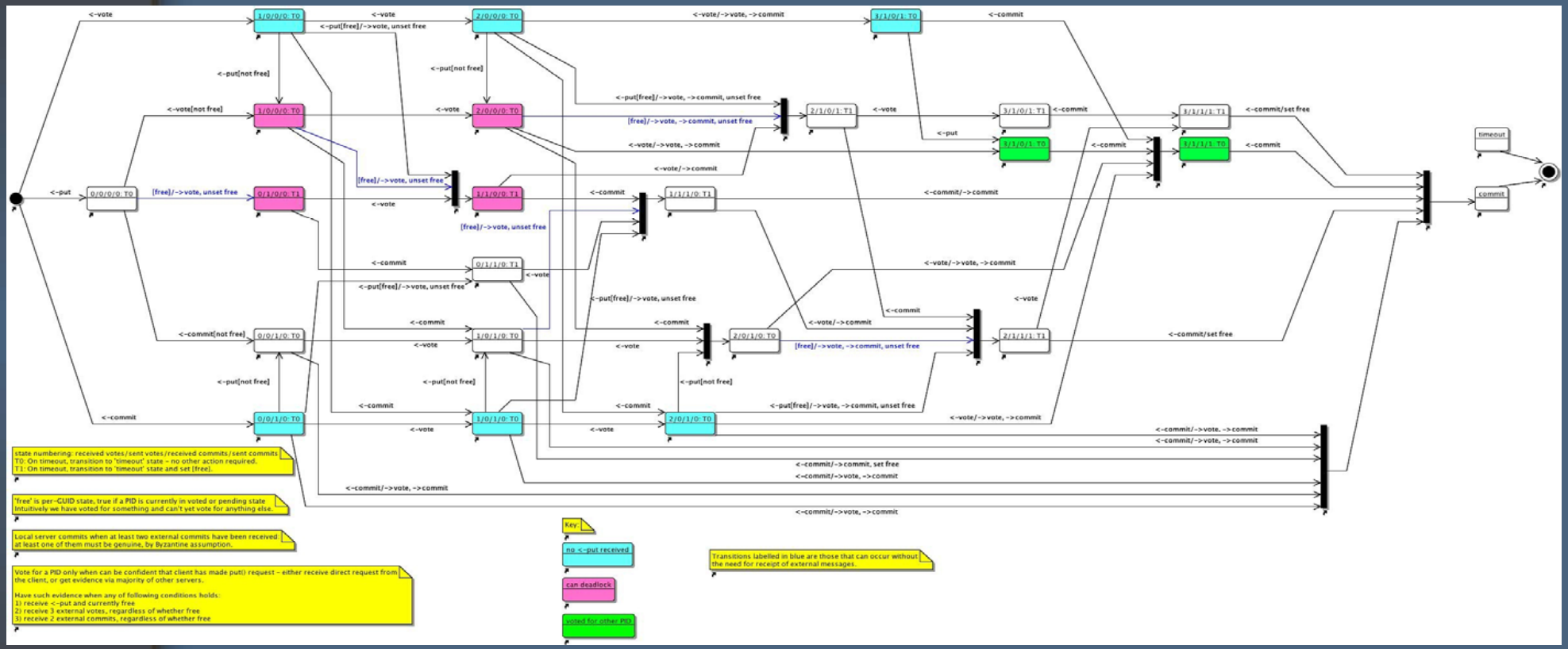
Motivating Example

- Distributed update algorithm
  - each data item replicated on a set of *n* servers (4 for basic Byzantine-fault-tolerance)
  - servers agree global ordering of updates
    - potentially concurrent
    - symmetric algorithm: no server is special

Approach

- Designed single generic algorithm
  - quorum-based
    - 'enough' servers must agree to each update
  - parameterised by replication factor *n*
  - about 500 lines pseudo-code
- Developed FSM model for selected replication factor (*n*=4)
  - 33 states
    - 5 boolean variables, 2 integers ranging 1..n

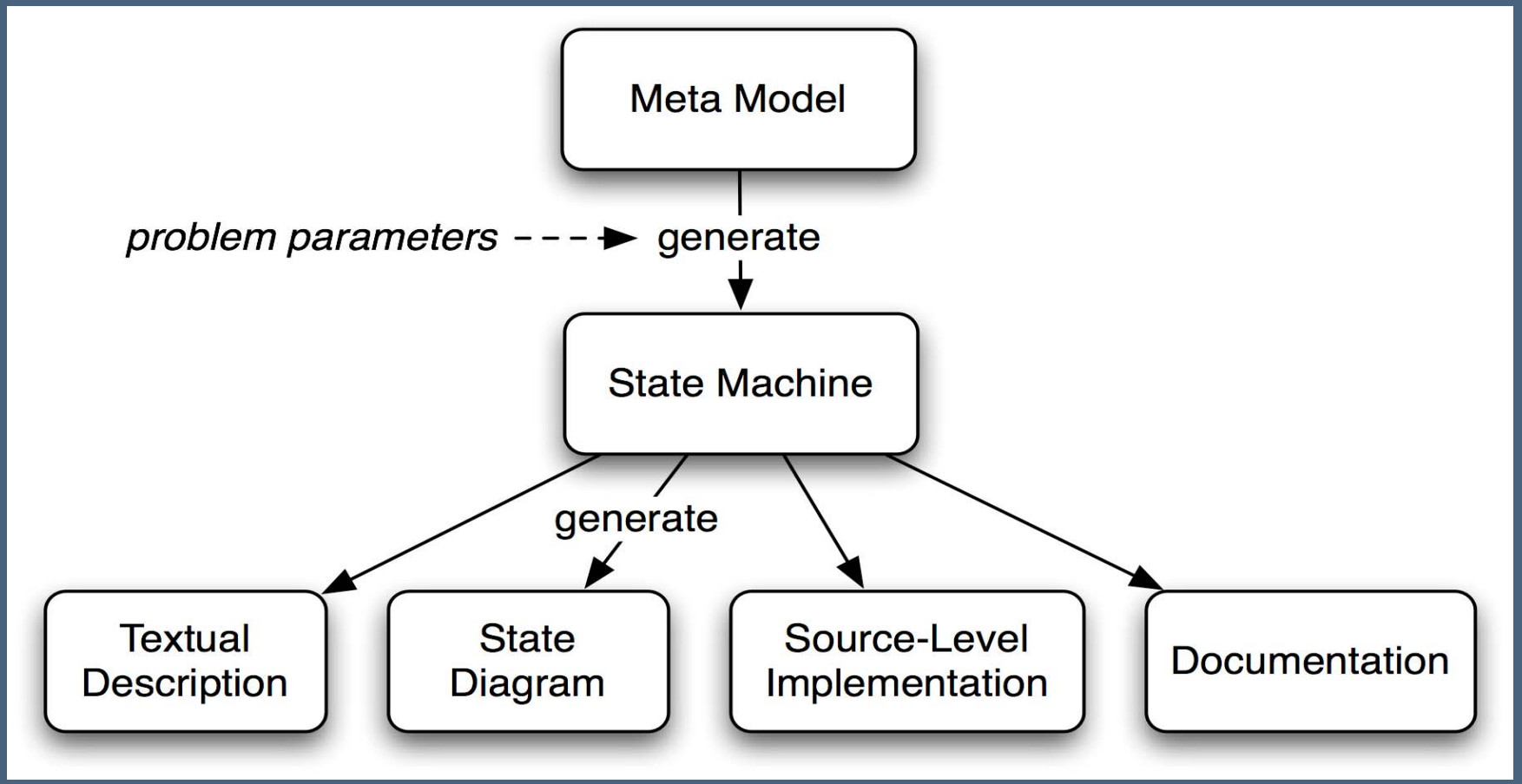
FSM for Replication Factor 4



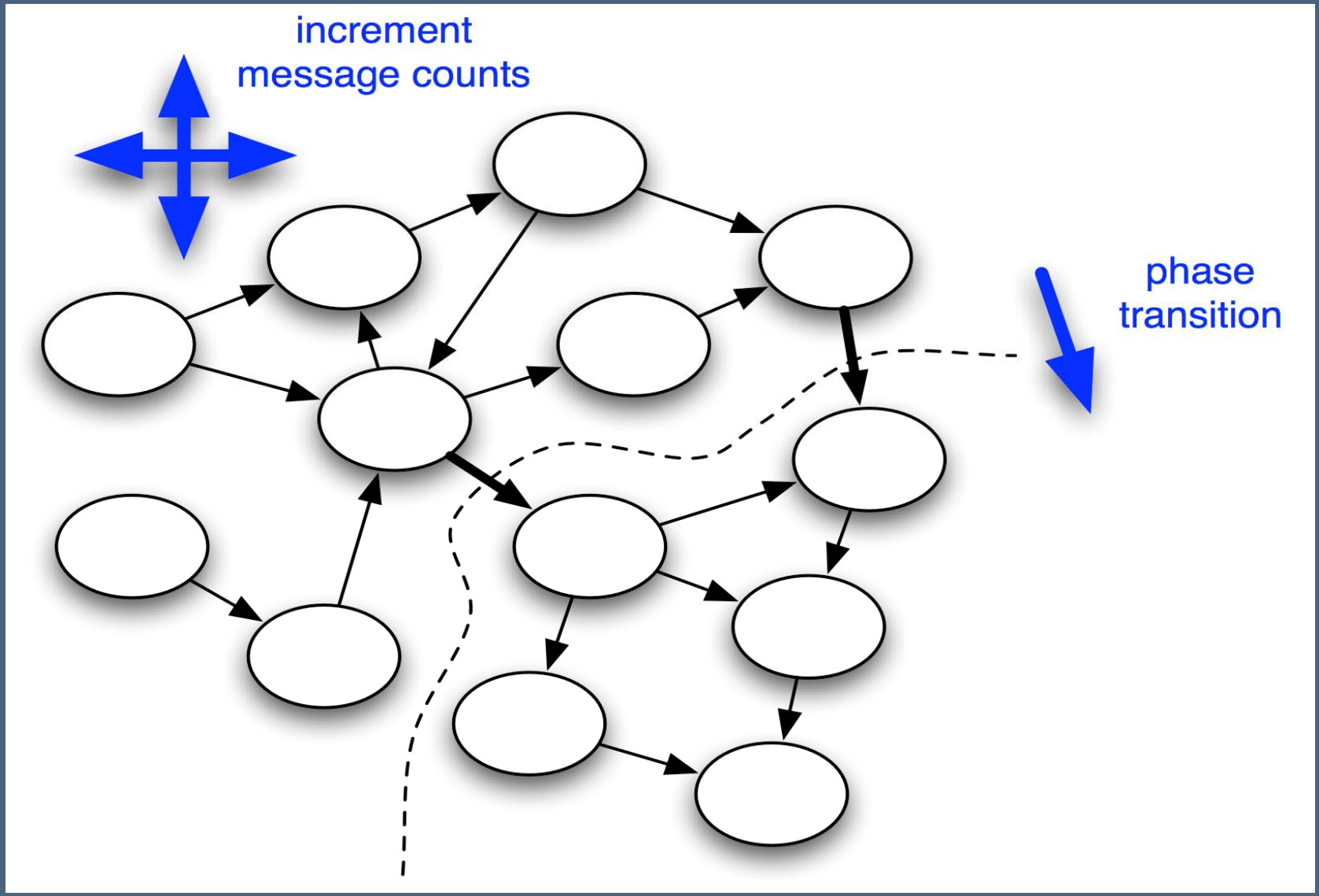
Did the FSM Help?

- No strong correlation between code and state machine
  - algorithm is generic
  - FSM is specific to replication factor
    - states in FSM correspond to message counts
    - so can't construct single FSM for algorithm
- Wish to unify FSM and algorithm
  - solution: define meta-model

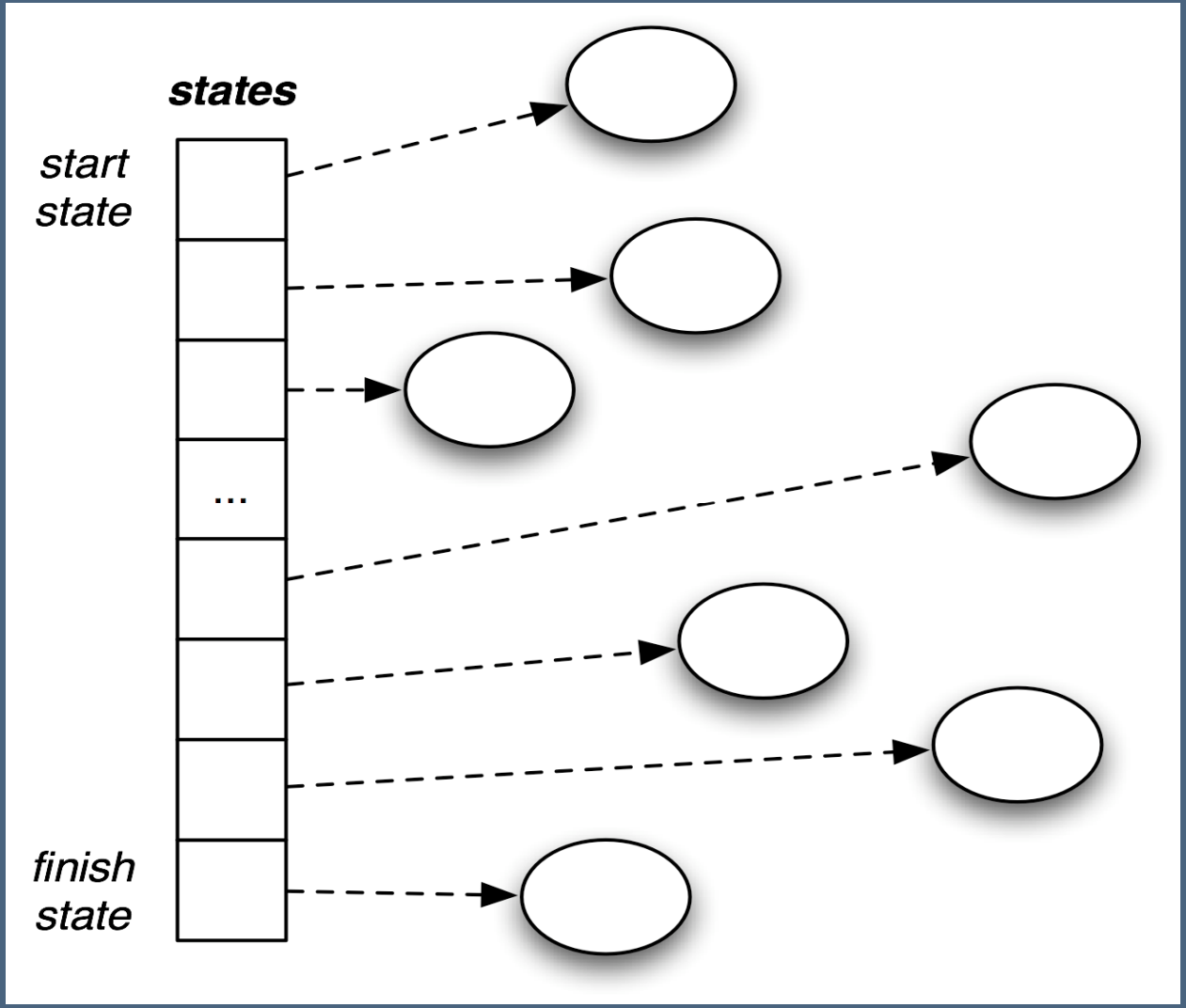
Generation Scheme



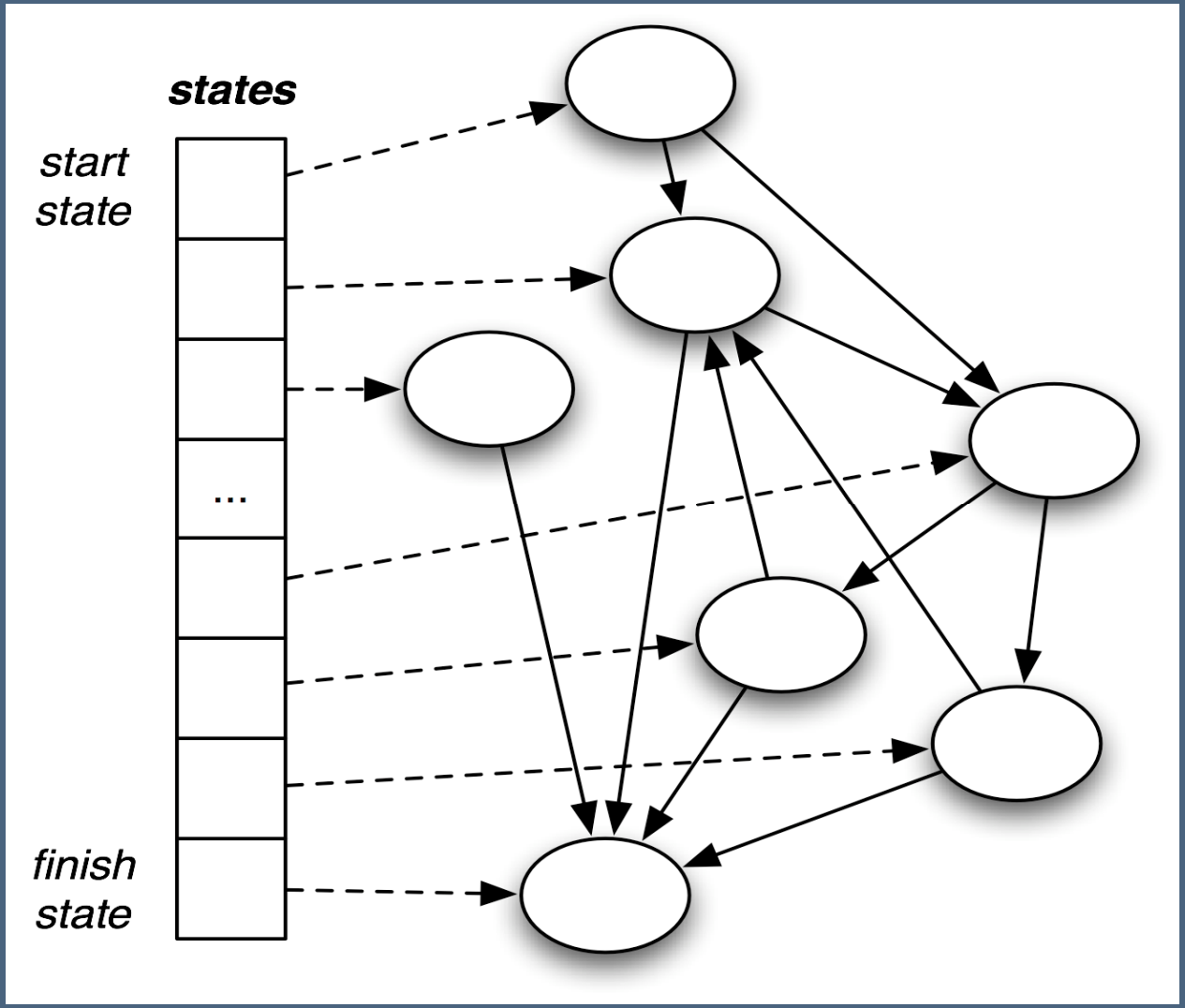
State Transitions



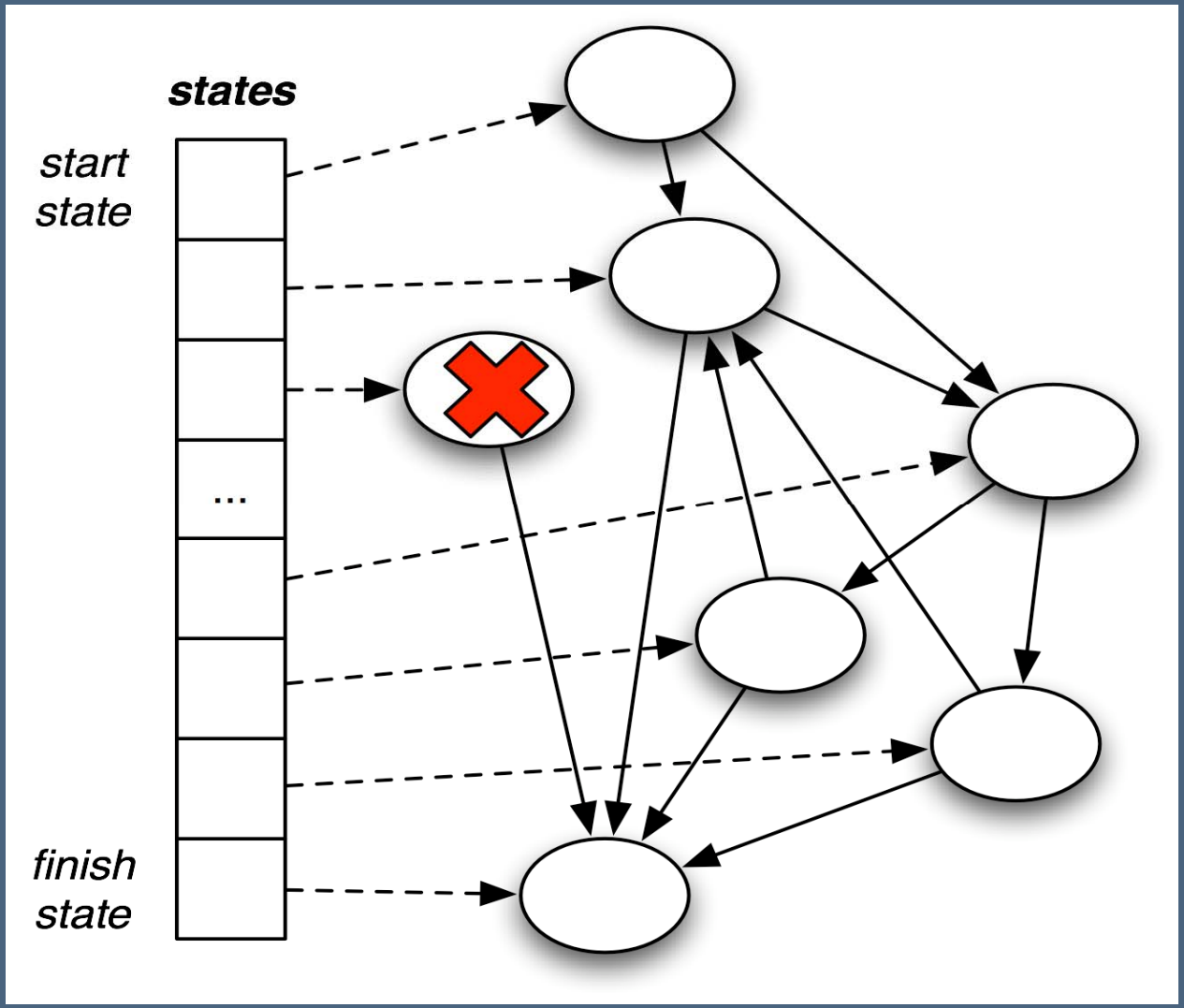
FSM Generation: All States



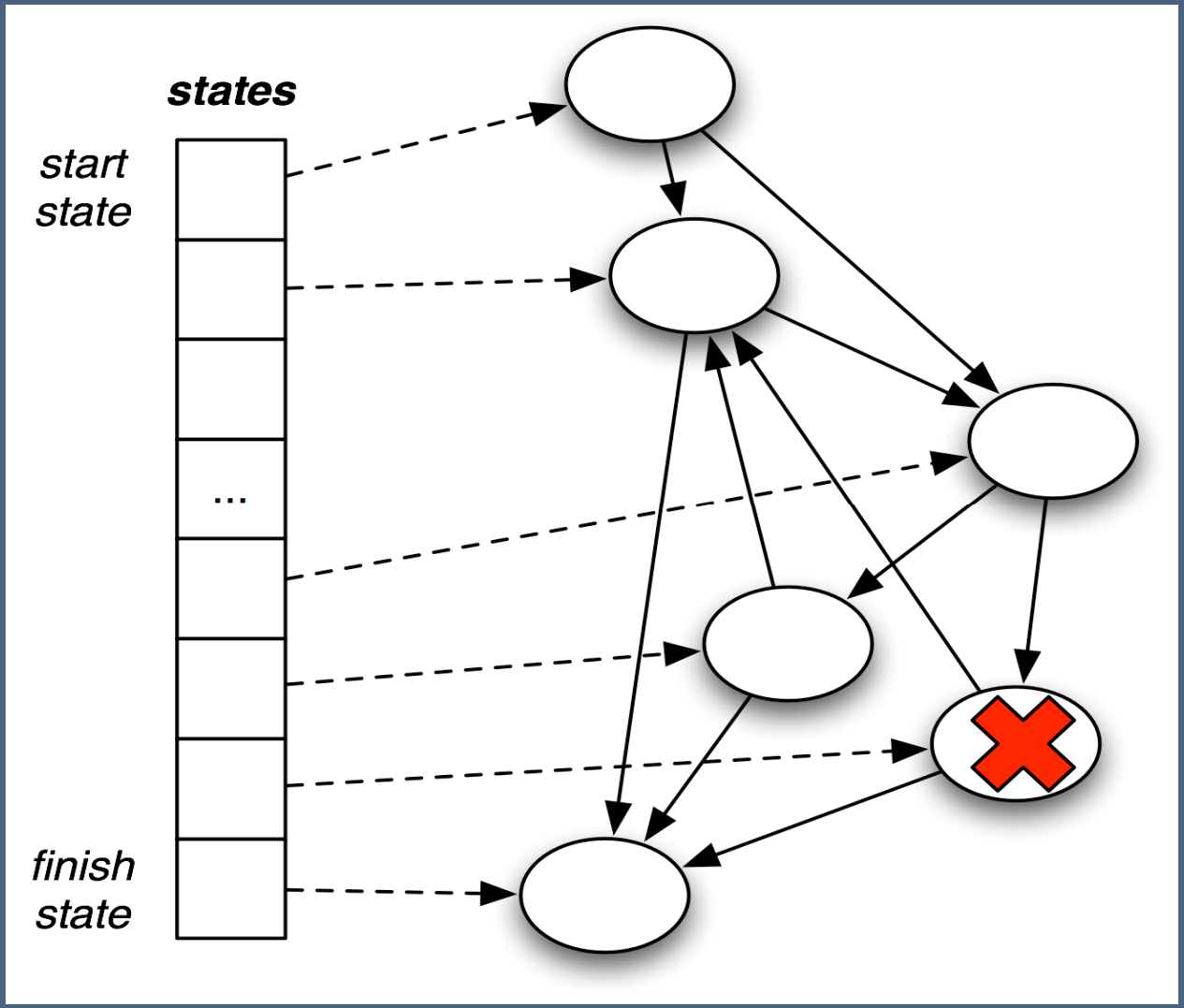
FSM Generation: Transitions



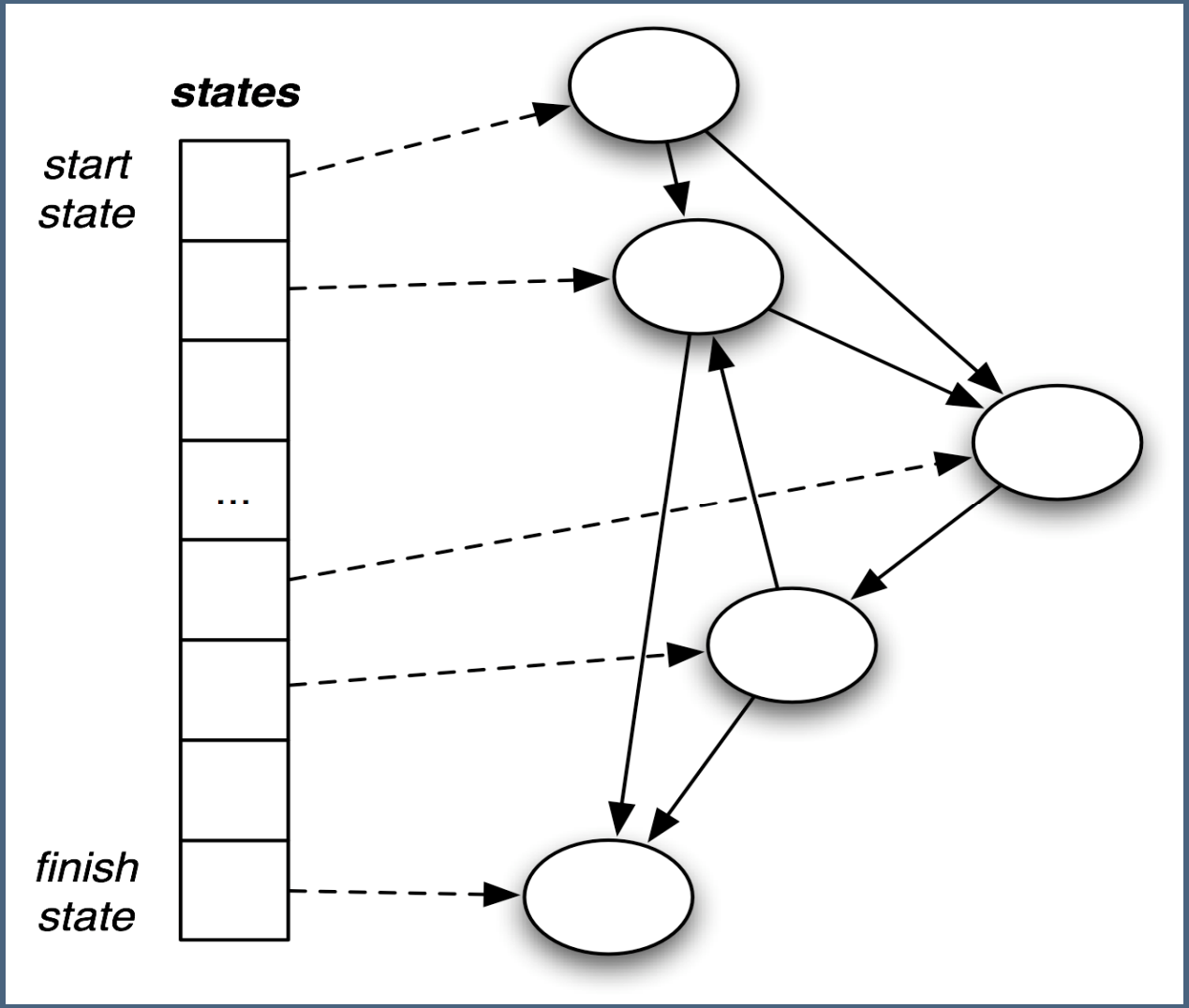
Pruning Unreachable States



Combining Equivalent States



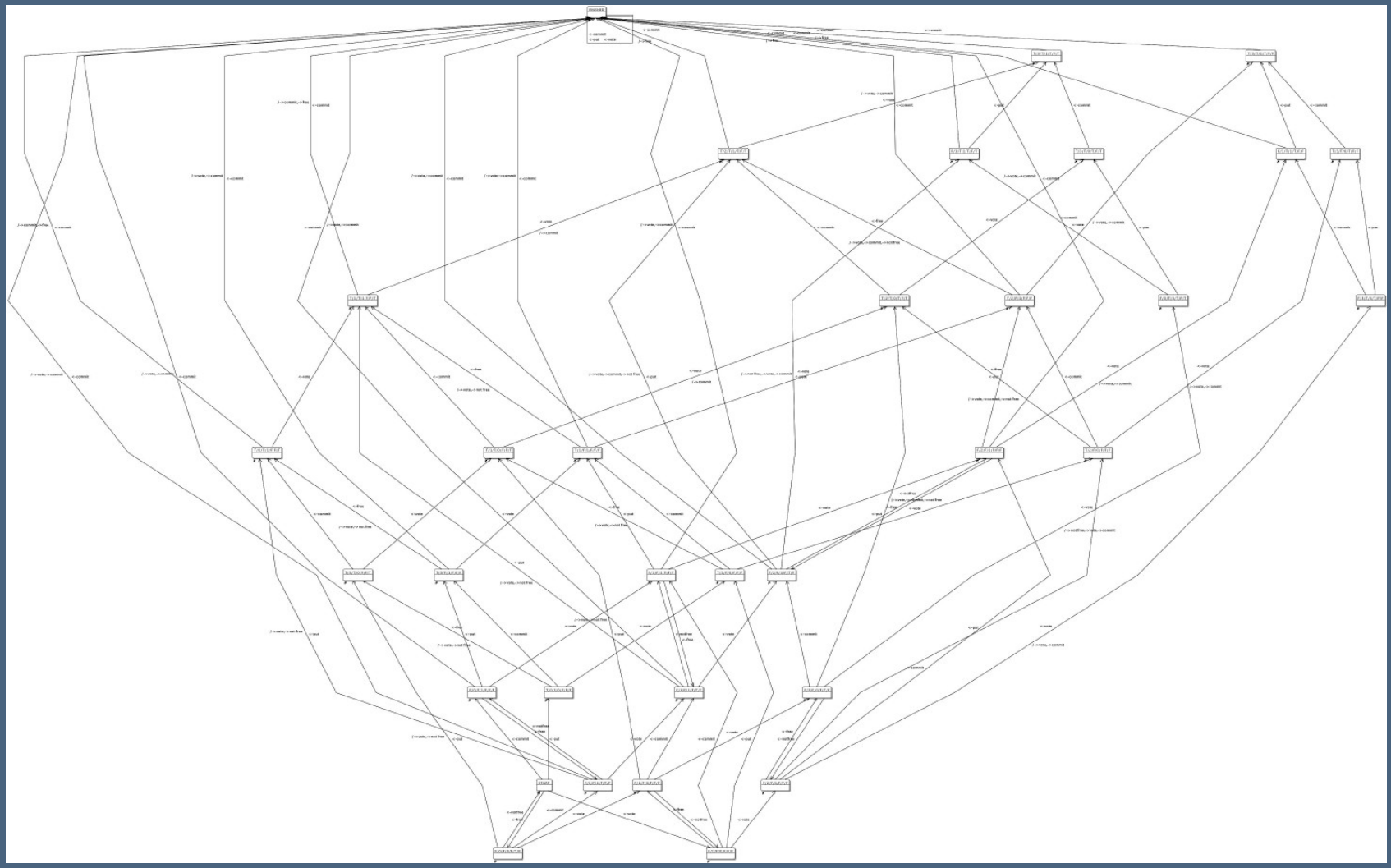
Final FSM Representation



Example Generated State

```
state: T/2/F/0/F/F/F

Have received initial ‘put’ from client. Have not voted since another update has
   already been voted for. Have received 2 votes and no commits. Have not sent a
   ‘commit’ since neither the vote threshold (3) nor the external commit
   threshold (2) has been reached. May not choose since another ongoing update
   has been voted for. Have not chosen this update since another ongoing update
   has been chosen. Waiting for 1 further vote (including local vote if any)
   before sending ‘commit’. Waiting for 2 further external commits to finish.

Transitions:

   message: VOTE
         action: send vote message
         action: send commit message
         transition to: T/3/T/0/T/F/F

   message: COMMIT
         transition to: T/2/F/1/F/F/F

   message: FREE
         action: send vote message
         action: send commit message
         action: send not free message
         transition to: T/2/T/0/T/T/T
```

Example Generated FSM



Example Generated Code

```
void receiveVote() {

  switch (getState()) {

    case (F-0-F-0-F-F-F) : {
      setState(F-1-F-0-F-F-F);
    }

    case (F-0-F-0-F-F-T) : {
      setState(F-1-F-0-F-F-F);
    }
    ...
    case (T-1-T-1-F-T-T) : {
      sendCommit();
      setState(T-2-T-1-T-T-T);
    }
    ...
}
```

handler for 'vote' message

switch on current state

variable representing state

action

state transition

Conclusions

- Generative meta-model approach
  - allows closer coupling of generic algorithm and specific FSMs
  - lead to discovery of several errors in original algorithm
  - may be applicable to other protocols for critical infrastructure

- Links
  - ASA project
    - asa.cs.st-andrews.ac.uk/
  - Algorithm details
    - asa.cs.st-andrews.ac.uk/metamodel/

Meta-Model

```
generateTransitionOnVote(State s) {
   initialise state variables from s
   increment votes_received
   if total votes >= threshold(r):
        if !vote_sent:
                if could_choose:
                        set has_chosen
                        record action:
                                send not free message
                record action: send vote message
                set vote_sent
                unset could_choose,
        if commit_sent:
                record action: send commit message
                set commit_sent
   derive new state s1 from state variables
   record transition s->s1 in data structure
}
```

Generation Times

f	r	initial states	final states	generation time (s)
1	4	512	33	0.10
2	7	1568	85	0.12
4	13	5408	261	0.38
8	25	20000	901	2.2
15	46	67712	2945	19.1