A Verified Model of Fault-Tolerance

John Rushby

Computer Science Laboratory
SRI International

Transient Faults are Common and Important

NASA-LaRC 1988 FCDS Technology Workshop:

- A number of DFCS are highly susceptible to radiated EM energy (composite materials provide less shielding, densely packed VLSI more susceptible to SEU)

- Designers must *prove* that their design will always recover from any and all non-hard faults reasonably quickly

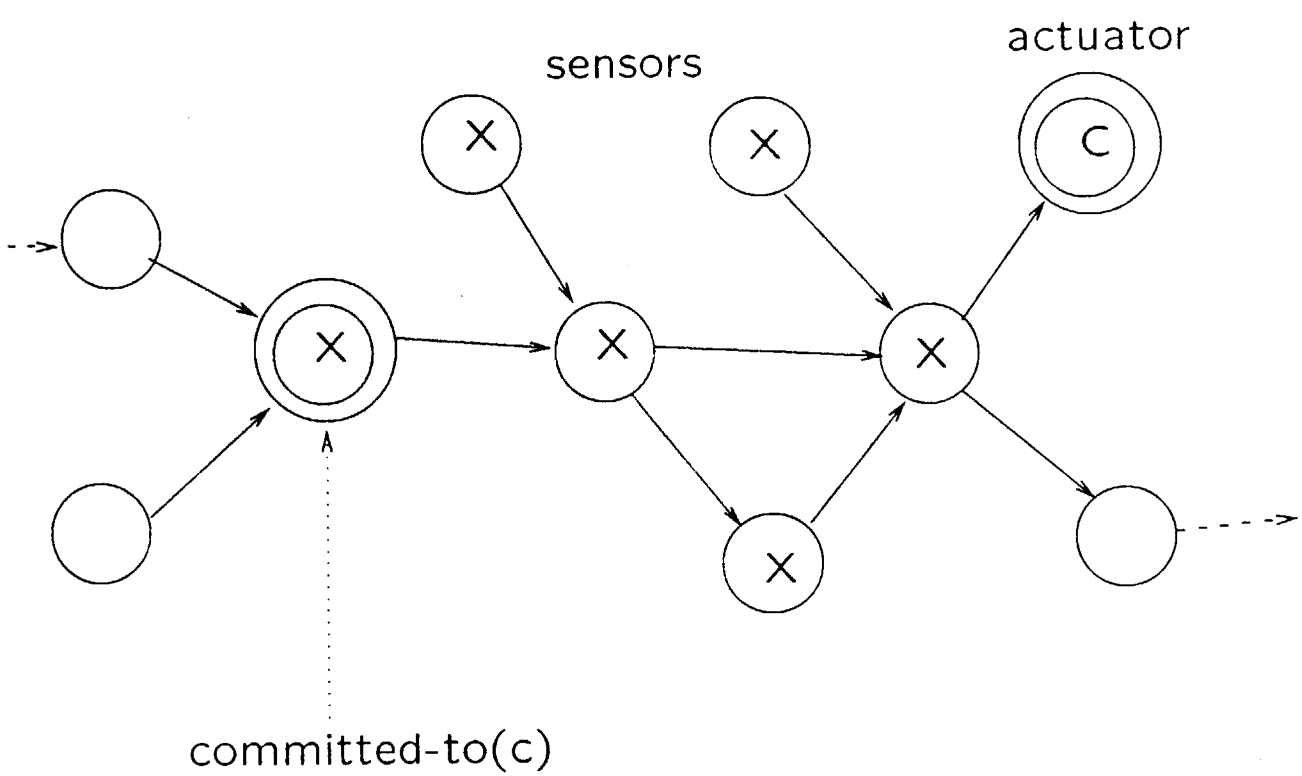
Goal

- A model of a replicated system with exact-match voting
- A fault model that includes transients
- A theorem that establishes the conditions under which the system provides fault tolerance
- A formal specification of the model and a mechanically checked verification of the theorem that is consonant with the journal-level presentation

Status

- Model based closely on that developed by Butler, Caldwell, and DeVito at LaRC, but simplified and more abstract
  - Does not model frames and cycles
  - Does not model sensor failure or loss of frame counter
- Model and theorem described in draft journal-level report
- Specification and verification in Ehdm completed (Jim Caldwell provided stimulation and help in the proof)
- Currently reconciling the two
- Next step is to address the (over) simplifications

General Idea



x = members of foundation(c)

Sets

```
sets: MODULE [T: TYPE]
EXPORTING ALL
THEORY

  set: TYPE IS function[T -> bool]

  x, y, z: VAR T
  a, b, c: VAR set

  union: function[set, set -> set] ==
    (LAMBDA a, b : (LAMBDA x : a(x) OR b(x)))

  subset: function[set, set -> bool] =
    (LAMBDA a, b : (FORALL z : a(z) IMPLIES b(z)))

  member: function[T, set -> bool] == (LAMBDA x, b : b(x))

  empty: function[set -> bool] = (LAMBDA a : (FORALL x : NOT a(x)))

  emptyset: set == (LAMBDA x : false)

  fullset: set == (LAMBDA x : true)

  extensionality: AXIOM
    (FORALL x : member(x, a) = member(x, b)) IMPLIES (a = b)
```

Cardinality

```
m, n, p: VAR nat

card: function[set -> nat]

card_ax: AXIOM
  card(union(a, b)) + card(intersection(a, b)) = card(a) + card(b)

card_subset: AXIOM subset(a, b) IMPLIES card(a) <= card(b)

card_empty: AXIOM card(a) = 0 IFF empty(a)

empty_prop: LEMMA card(a) > 0 IMPLIES (EXISTS x : member(x, a))

card_prop: LEMMA
  subset(a, c)
      AND subset(b, c)
        AND 2 * card(a) > card(c) AND 2 * card(b) > card(c)
    IMPLIES card(intersection(a, b)) > 0
```

Sensors etc.

```
C: TYPE

a, c: VAR C

cell_types: TYPE = (sensor_cell, actuator_cell, task_cell)

cell_type: function[C -> cell_types]

sensors: TYPE FROM C WITH (LAMBDA c : cell_type(c) = sensor_cell)

actuators: TYPE FROM C WITH (LAMBDA c : cell_type(c) = actuator_cell)

active_tasks: TYPE FROM C WITH
  (LAMBDA c : cell_type(c) /= sensor_cell)

voted: TYPE FROM C

voted_ax: AXIOM
  (c IN actuators IMPLIES c IN voted)
    AND (c IN voted IMPLIES NOT (c IN sensors))

Gbar: function[C, C -> bool]

sensor_ax: AXIOM (EXISTS a : Gbar(a, c)) IFF NOT (c IN sensors)
```

Simple Machine

$$step(\sigma, c, n) = \sigma \textbf{ with } [c := \textbf{if } \; c \in C\_S \textbf{ then } \; sensor(c)(n) \textbf{ else } \; task(c)(\sigma)]$$

$$\begin{aligned} run(0) &= (\lambda c : \perp) \\ run(n+1) &= step(run(n), sched(n+1), n+1). \end{aligned}$$

```
step: function[state, C, M -> state] =
  (LAMBDA s, c, m : s
     WITH [c :=
           IF c IN sensors THEN sensor(c)(m) ELSE task(c)(s) END IF])

identity: function[M -> nat] == (LAMBDA m : m)

run: RECURSIVE function[M -> state] =
  (LAMBDA m :
     IF m = 0 THEN undef ELSE step(run(m - 1), sched(m), m) END IF)
  BY identity
```

TCC's

```
(* Subtype TCC generated for the first argument to task in dependency *)

  dependency_TCC1: FORMULA
    (c IN active_tasks AND (FORALL a : Gbar(a, c) IMPLIES s(a) = t(a)))
      IMPLIES (cell_type(c) /= sensor_cell)

(* Subtype TCC generated for the first argument to sensor in step *)

  step_TCC1: FORMULA (c IN sensors) IMPLIES (cell_type(c) = sensor_cell)

(* Subtype TCC generated for the first argument to task in step *)

  step_TCC2: FORMULA
    (NOT (c IN sensors)) IMPLIES (cell_type(c) /= sensor_cell)

(* Subtype TCC generated for the first argument to run in run *)

  run_TCC1: FORMULA (m >= 0) IMPLIES (NOT (m = 0)) IMPLIES (m - 1 >= 0)

(* Termination TCC generated for run *)

  run_TCC2: FORMULA
    (m >= 0) IMPLIES (NOT (m = 0)) IMPLIES identity(m) > identity(m - 1)
```

Replicated Machine

$$\neg\mathcal{F}(i)(n) \supset sstep(\rho, c, n)(i) = step(\rho(i), c, n),$$

$$\neg\mathcal{F}(i)(n) \supset vote(\rho, c, n)(i) \;=\; \begin{array}{l}\textbf{if}\ \ c \in C\_V\ \ \textbf{then}\ \ \rho(i)\ \textbf{with}\ [c := maj\{\rho(j)(c) | j \in R\}] \\ \textbf{else}\ \ \rho(i)\end{array}$$

$$rstep(\rho, c, n) = vote(sstep(\rho, c, n), c, n).$$

Replicated Machine

```
sstep_ax: AXIOM
  NOT (F(i)(m)) IMPLIES sstep(rs, c, m)(i) = step(rs(i), c, m)

maj_ax: AXIOM
  (EXISTS A :
       2 * card(A) > card(fullset[R])
         AND (FORALL i : member(i, A) IMPLIES rs(i)(c) = x))
    IMPLIES maj(rs, c) = x

vote_ax: AXIOM
  NOT (F(i)(m))
    IMPLIES vote(rs, c, m)
      = IF c IN voted
           THEN rs WITH [(i)(c) := maj(rs, c)]
        ELSE rs END IF

rstep: function[rstate, C, M -> rstate] ==
  (LAMBDA rs, c, m : vote(sstep(rs, c, m), c, m))

rrun: RECURSIVE function[M -> rstate] =
  (LAMBDA m :
     IF m = 0
       THEN (LAMBDA i : undef)
     ELSE rstep(rrun(m - 1), sched(m), m) END IF)
  BY identity
```

Foundation etc.

$$foundation(c) = \begin{cases} \{c\} & \textbf{if} \quad c \in (C\_S \cup C\_V) \\ \{c\} \cup \bigcup_{(a,c) \in \overline{G}} foundation(a) & \textbf{otherwise} \end{cases}$$

$$support(c) = \begin{cases} \{c\} \cup \bigcup_{(a,c) \in \overline{G}} foundation(a) & \textbf{if} \quad c \in C\_V \\ foundation(c) & \textbf{otherwise}. \end{cases}$$

$$\textsf{committed-to}(c) = min\{when(a) | a \in support(c)\}.$$

Foundation etc.

```
foundation: RECURSIVE function[C -> set[C]] =
  (LAMBDA c :
     (LAMBDA a :
        c = a
          OR (NOT (c IN voted OR c IN sensors)
               AND (EXISTS b :
                  Gbar(b, c) AND member(a, foundation(b))))))
  BY dowhen

backup: function[C -> set[C]] =
  (LAMBDA c :
     (LAMBDA a :
        (EXISTS b : Gbar(b, c) AND member(a, foundation(b)))))

support: function[C -> set[C]] =
  (LAMBDA c :
     (LAMBDA a :
        member(a, foundation(c))
          OR (c IN voted AND member(a, backup(c)))))

critical_times: function[C -> set[M]] ==
  (LAMBDA c : (LAMBDA t : member(sched(t), support(c))))

committed_to: function[C -> M] == (LAMBDA c : min(critical_times(c)))
```

OK and MOK

$$OK(i)(c) = (\forall n : \textit{committed-to}(c) \leq n \leq when(c) \supset \neg\mathcal{F}(i)(n)).$$

$$MOK(c) = \exists\Theta \subseteq R, |\Theta| > r/2, i \in \Theta \supset OK(i)(c).$$

```
OK: function[R -> set[C]] =
  (LAMBDA i :
     (LAMBDA c :
        (FORALL m :
           committed_to(c) <= m AND m <= dowhen(c)
             IMPLIES NOT F(i)(m))))

working: function[C -> set[R]] == (LAMBDA c : (LAMBDA i : OK(i)(c)))

MOK: function[C -> bool] =
  (LAMBDA c : 2 * card(working(c)) > card(fullset[R]))
```

Theorem

If

$$\forall a : when(a) \leq when(c) \supset MOK(a),$$

then

$$\forall j : OK(j)(c) \supset rrunto(c)(j)(c) = runto(c)(c).$$

```
safe: RECURSIVE function[C -> bool] =
  (LAMBDA c : MOK(c) AND (FORALL a : Gbar(a, c) IMPLIES safe(a)))
  BY dowhen

correct: function[C -> bool] =
  (LAMBDA c :
     (FORALL j : OK(j)(c) IMPLIES rrunto(c)(j)(c) = runto(c)(c)))

the_result: THEOREM safe(c) IMPLIES correct(c)
```

Noetherian Induction

```
noetherian: MODULE [dom: TYPE, <: function[dom, dom -> bool]]
ASSUMING
  measure: VAR function[dom -> nat]
  a, b: VAR dom

  well_founded: FORMULA
    (EXISTS measure : a < b IMPLIES measure(a) < measure(b))
THEORY
  p, A, B: VAR function[dom -> bool]
  d, d1, d2, d3, d4: VAR dom

  general_induction: AXIOM
    (FORALL d1 : (FORALL d2 : d2 < d1 IMPLIES p(d2)) IMPLIES p(d1))
      IMPLIES (FORALL d : p(d))

  mod_induction: THEOREM
    (FORALL d3, d4 : d4 < d3 IMPLIES A(d3) IMPLIES A(d4))
        AND (FORALL d1 :
            (FORALL d2 : d2 < d1 IMPLIES (A(d1) AND B(d2)))
              IMPLIES B(d1))
      IMPLIES (FORALL d : A(d) IMPLIES B(d))
PROOF
  mod_proof: PROVE mod_induction d1 <- d1@p1, d3 <- d1@p1, d4 <- d2
    FROM general_induction p <- (LAMBDA d : A(d) IMPLIES B(d))
END noetherian
```

The Proof

```
correctness_proof: MODULE
USING correctness, voted_step, nonvoted_step, sensor_step,
  noetherian[C, Gbar]
PROOF
  a, c: VAR C

  discharge_well_founded: PROVE well_founded measure <- dowhen FROM
    Gbar_when c <- b

  inductive_step: LEMMA
    (FORALL a : Gbar(a, c) IMPLIES safe(c) AND correct(a))
      IMPLIES correct(c)

  almost_final_proof: PROVE inductive_step a <- a@p7 FROM
    sensor_inductive_step,  voted_inductive_step,  nonvoted_inductive_step,
    induction_body a <- a@p1,  induction_body a <- a@p2,
    induction_body a <- a@p3,  induction_body

  final_proof: PROVE the_result FROM
    mod_induction A <- safe, B <- correct, d <- c, d2 <- a@p3,
    safe a <- d4@p1, c <- d3@p1,
    inductive_step c <- d1@p1

END correctness_proof
```

Summary

- Formal specification and verification revealed typos in the original report
- Exposed omission in original proof
- Led to stronger theorem and more elegant proof (using Noetherian rather than ordinary induction)
- Confirmed that Ehdm has the capability to specify interesting and useful properties in a direct, natural, and readable manner
- Proofs were hard (three intensive man-weeks, 92 lemmas); I haven't yet gone back to see why that was so
- We have the beginnings of a formally verified model for a fault tolerant operating system