Formal Methods for Safety Assessment (ctd)

Model Checking

- complement simulation and testing
- system is modeled as state transition system
- check automatically, that specifications are met (exhaustively explore state space)
- finite models ⇒guaranteed termination
- counterexample is produced, when failure state is found
- check a *Kripke*-structure ⇒may be exponential in number of components
- *explicit model checking* SPIN: explicitly compute states
- *symbolic model checking* SMV, NuSMV, VIS⇒manipulate sets of states and transitions
- *binary decision diagrams* SMV: representation for logic formulae
- *bounded model checking* (symbolic) bound steps to a value k, uses modern SAT solvers
- often run bdd and bmc in parallel, the first one to finishes, wins
- *timed model checking* UPAAL, KRONOS
- *probabilistic model checking* PRISM, MRMC
- *partial order reduction* ⇒reduce state space
- *symmetry reduction* exploit symmetry in model

Using Model Checking for Requirements Validation

- validate a set of requirements (assumes, that requirements meet end-user expectations)
- RAT tool provides way to asses quality of requirements
  - *property simulation* ⇒behavior associated with each requirement
  - *property assurance* check for logical consistency (=freedom from contradictions)
    - if requirements are logically inconsistent ⇒find subset, that is explanation for inconsistency

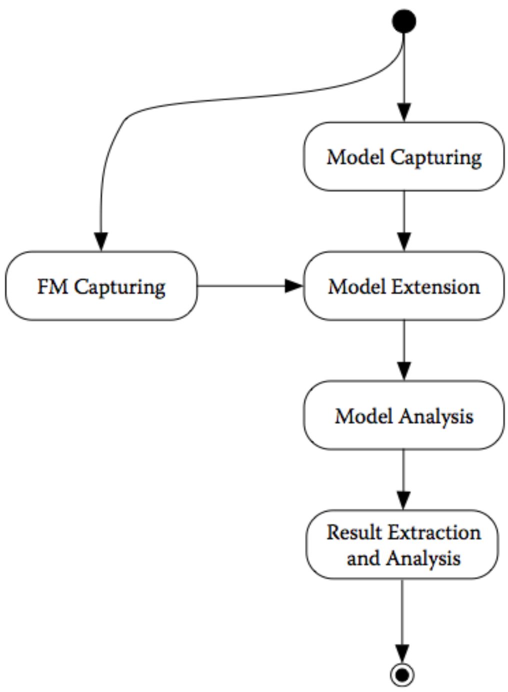
Using Model Checking for Property Verification

- build formal model for system (state transition system)
- checked by model checker
- FSAP tool for formal verification and safety analysis, GUI on top of NuSMV

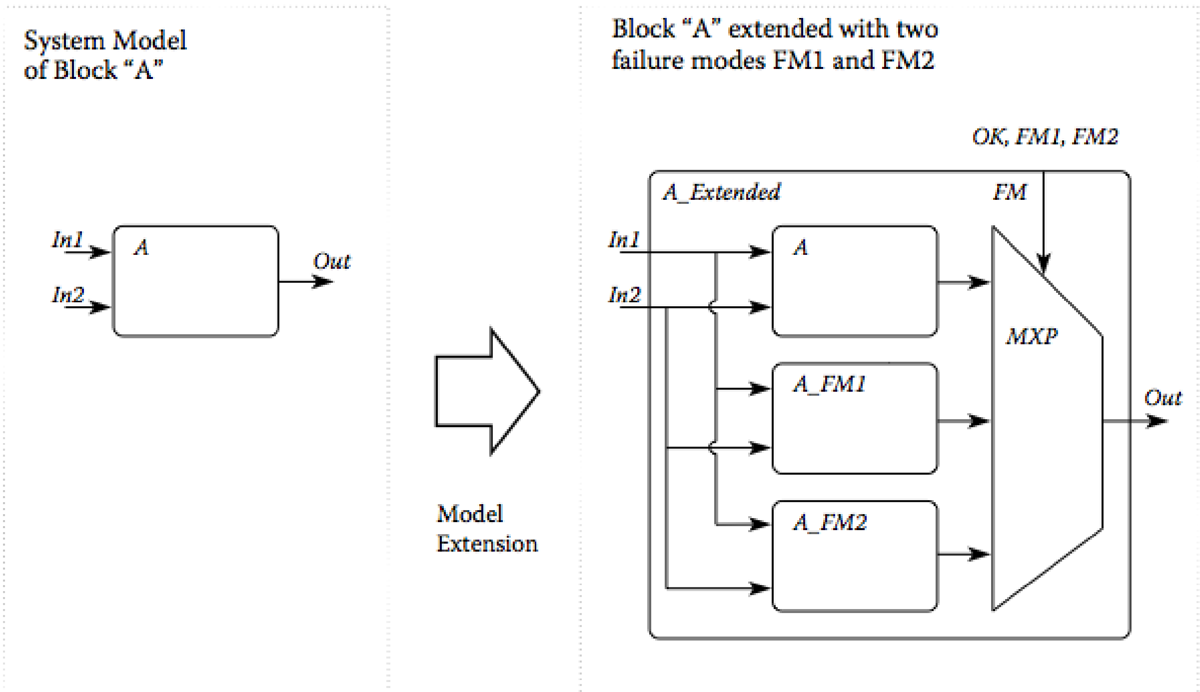
Formal Safety Analysis

- ESACS and ISAAC ⇒ESACS methodology supported by formal methods, to assist system development and safety analysis
- shared formal notations between design and safety analysis
- decoupling between nominal model and fault model
  1. formal model (design or safety engineer) ⇒*nominal system model* (only nominal behavior)

2. *fault injection*: put nominal model into failure modes ⇒extended model through model extension



Fault Injection



Fault Models and Model Extension

- FSAP generic failure mode library: library of typical failure modes to be injected
- model extension takes specification of fault to be added
- *fault configuration* subset of failure mode variables

Fault Tree Generation

- fault tree can be represented as collection: of minimal cut sets TLE and and conjunction of corresponding basic faults

- generation of fault tree:
  - need to remember, if fault has been activated during iterative traversal

Table 5.4 A Symbolic Algorithm for Fault Tree Generation

```
function FTA-Forward (M, Tle)
    M := Extend(M, R°);
    Reach := I ∩ (o = f);
    Front := I ∩ (o = f);
    while (Front ≠ ∅) do
        temp := Reach;
        Reach := Reach ∪ fwd_img(M, Front);
        Front := Reach\temp;
    end while;
    CS := Project(o, Reach ∩ Tle);
    MCS := Minimize(CS);
    return Map_{o→f}(MCS);
```

FMEA Table Generation

- input: set of failure modes + set of events to be analyzed
- FSAP does this somehow (magic?)

Industrial Application of Formal Methods

IBMs Customer Information Control System

- more than 750 000 lines of code
- more than 2 000 pages of formal specifications produced
- reduction in development cost of about 9%

Conclusion

- formal methods in design and development become more pervasive
- CounterExample Guided Abstraction Refinement (CEGAR) builds set of abstraction from set of predicates
- nontrivial: systems with both digital and analogue parts (*hybrid systems*)