From One to Many: Checking A Set Of Models

Iowa State University

Rohit Dureja and Kristin Yvonne Rozier

Laboratory for Temporal Logic

Motivation

- The design of a complex system often requires analyzing several models of the system under development for:
  - narrowing in on the final system design,
  - check capabilities of systems with varying features, or
  - regression verification to make a design more robust.
- The **set of models** constitute the system's **design space**.
- Models in a set represent different design options for the system, features, or bug fixes.
- Different models differ in terms of core capabilities, implementations, and component configurations.

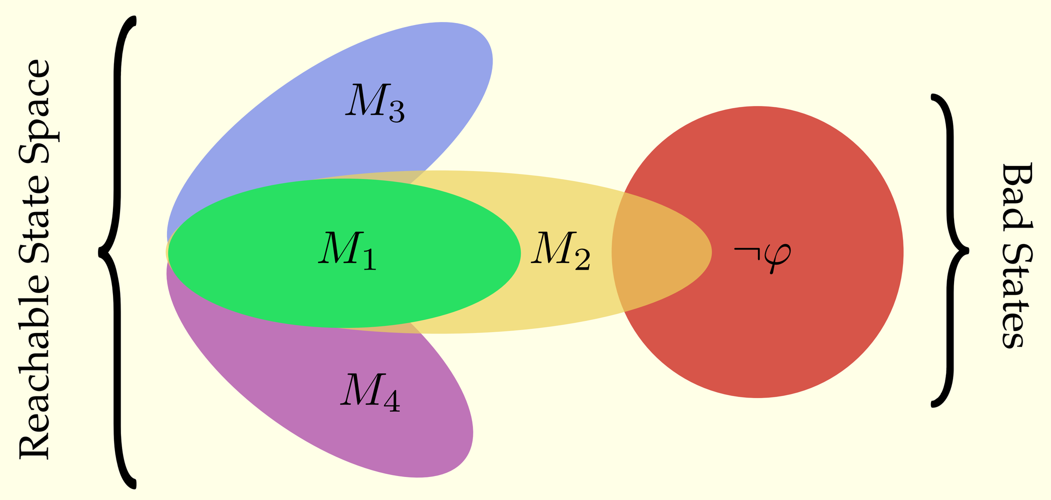
Model checking aids system development via a thorough comparison of the set of models.

Classical Approach Check each model in the set one-by-one against the set of properties representing system requirements.

- Inefficient for large design spaces; may not scale to handle combinatorial size of the design space.

Important Observation Different models in the design space are related, i.e., have overlapping state spaces.

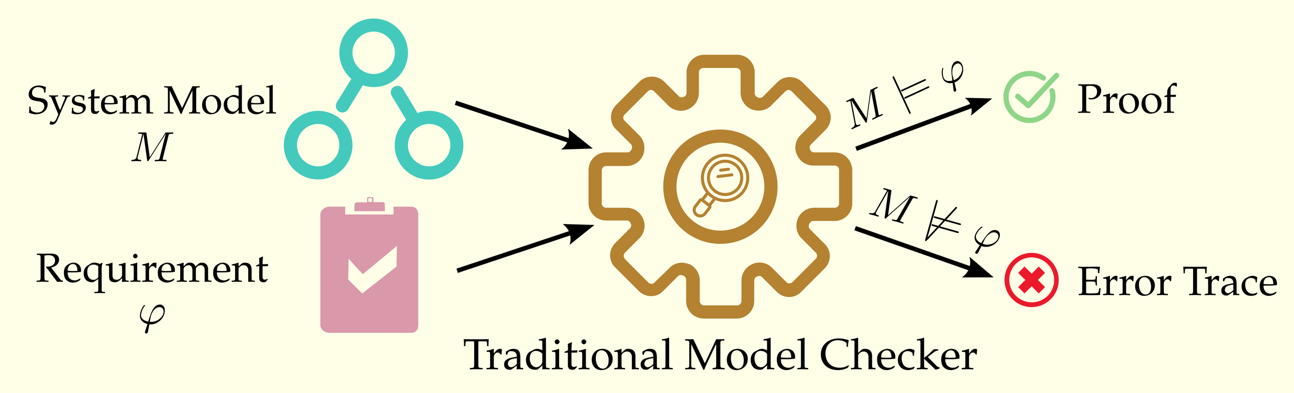
- Traditional checking does not take advantage of this information; wasteful for large models.
- Information from earlier model-checking runs can speed-up future checking efforts; like reusing variable ordering in BDD-based model checking.



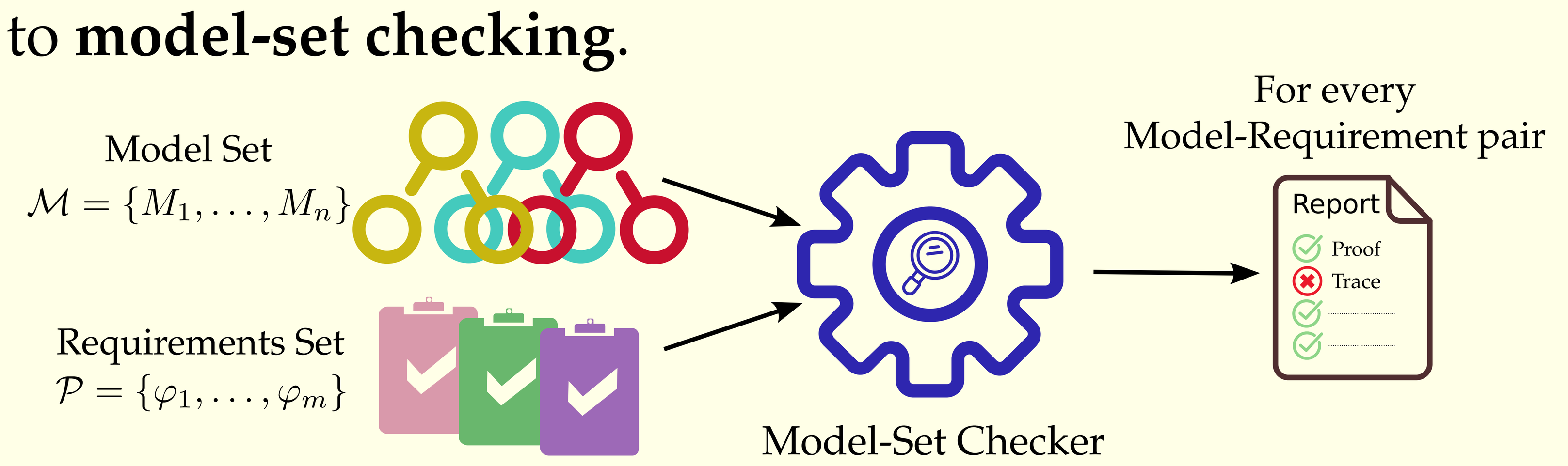
Venn diagram representation of reachable states for a set of models $M = \{M_1, M_2, M_3, M_4\}$ and bad states $\neg\varphi$. Model $M_2 \not\models \varphi$ since reachable states of M_2 intersect bad states $\neg\varphi$. If M_1 is checked before M_2, the checker run for M_2 can reuse the already explored and verified state space of M_1.

Problem Statement

Lift **traditional model checking**



to **model-set checking**.



Our Solution

FuseIC3 (from **F**rame Re**use**)

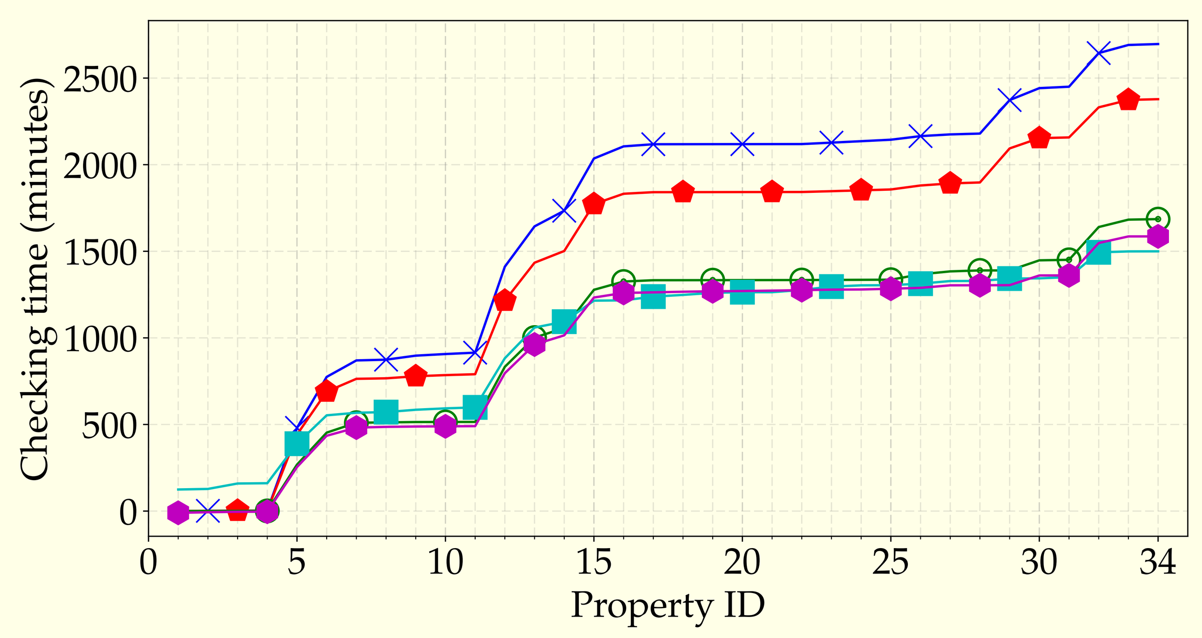
- Extends IC3/PDR to deal with a set of models.
- Checks each model in the set by reusing information: state approximations, counterexamples, and invariants.
- Repairs and patches unusable information using a SAT-query efficient algorithm.

FuseIC3 is a median 1.75× (average 5.48×) faster than checking each model individually on real-life industrial benchmarks.

Heuristic #1: Model Ordering

- Every model-pair is assigned a score based on
  1. number of "same" assignments to variables
  2. overlap between assignments to the same variable
- Both metrics computed in linear time.
- Models organized as a weighted and complete undirected graph; polynomial-time solution to the *greedy Hamiltonian Path Problem* gives a possible model ordering.

Preliminary Results and Ongoing Work



Cumulative model checking time per algorithm. 1,620 models are checked for each property ID; total of 34 properties.

- FuseIC3 is on average 3.67× faster than Incremental IC3.
- Heuristic #1: overhead, Heuristic #2: marginally faster.
- Model preprocessing heuristics may improve performance.
- Development of a tool for checking model-sets that incorporates theoretical advances.

Heursitic #2: Information Ranking

- Unusable clauses are scored based on
  1. number of overlapping literals with the bad state
  2. number of frames that contain the clause
- Both metrics computed in constant time (delta encoding)
- Only repair clauses that block a bad state and have the highest score in the future, instead of repairing them prematurely (lazy).

Publications

1) "FuseIC3: An algorithm for checking large design spaces," in FMCAD, 2017.
2) "Combinatorial model checking reduction," (under submission).
3) "Nexus: A model checker for large design spaces," (ongoing work).

Supported by NSF CAREER Award CNS-1552934.