Model Checking

Static analysis techniques for finite-state models and design representations

A note on terminology

• “Model checking” often means “temporal logic model checking”
  - And recently, often just “Symbolic model checking with OBDD models”
• Related terms:
  - Finite-state verification (of concurrent programs)
  - Reachability analysis, concurrency analysis
• Closely related to flow analysis of sequential and concurrent programs
Models and Formulae

• An object may be a model of a formula
  - i.e., the models of a specification are objects that satisfy it; an inconsistent specification has no models
• Model checking: Given an object and a formula (specification), determine whether the object is a model of the formula
• Models derived from programs or designs, formulas express desired properties

Models & Formulae: Examples

• Models
  - Control flow graphs, data flow graphs
  - Reachability graphs (of Petri nets, process graphs, etc.)
• Formulae & other specs
  - Logics: Propositional or first-order, ordinary or temporal, real-time, authentication, . . .
  - Languages: Regular expressions, context-free languages
  - Particular properties of interest, e.g., freedom from deadlock
Temporal Logic

- Like a standard (first order or propositional) logic with additional connectives
  - first-order: with quantifiers; propositional: without
  - “eventually” (“future,” “sometime”) Abbrev: F
  - “always” (“henceforth,” “globally”) Abbrev: G
  - “until” Abbrev: U
  - “next” (seldom desirable at spec level) X

Meaning of “Eventually”

- Interpret propositional temporal logic as first-order statements about a sequence of program states S₀, S₁, ...
- Si |- p iff p is true in Si
- Si |- F p iff Sj |- p, for some j ≥ i

![Diagram of time and states](image-url)
Alternate definition of “eventually”

- $S \models p$ iff $S_0 \models p$
- $S \models Fp$ iff $S \models p$ or $XS \models Ep$
  - This latter definition is the basis of model-checking algorithms

Other temporal connectives

- Eventually $q$: $q$ in this state, or eventually $q$ in the next state
- Always $p$: $p$ in this state, and always $p$ in the next state
- $p$ Until $q$: $q$ in this state, or $p$ in this state and $p$ Until $q$ in the next state
- Next $p$: $p$ in the next state
Why temporal logic?

- To say:
  “Eventually the call gets through”
  “Race conditions never occur”
  “N/S green does not come on until E/W light is red”
  “If scheduler is fair, all processes eventually run”
- Properties of progress, but not of metric time
- Especially for eventuality; safety (never, always) can be specified in other ways

Why use logic at all?

vs. operational spec or model

- Twin dangers of over and under-specification
  - Logic specs often say too little
  - Operational models often say too much
- Combination appears to be attractive
  - Say a few simple things with an appropriate logic
  - If the logic gets messy, move part of it into another kind of spec
- Example: Lamport’s transition axiom method
  - State machine with invariants for safety properties, temporal logic for liveness properties
Temporal logic model checking

• Given a graph model of a program
  – State machine in which the propositional variables can be evaluated
• Given a propositional temporal logic formula
• Determine whether the model satisfies (“is a model of”) the formula

CTL: Restricted branching-time logic

• Branching time: Quantification over paths
  – A graph of possible execution histories, not a single path through the program
  – A: All paths (from here)
  – E: Some path (from here)
• Restriction: Require quantifier with each temporal connective (for efficient checking)
  – AF, EF (inevitably, potentially)
  – AG, EG (always)
  – AU, EU (until)
Checking AFp

- Evaluate $p$ in every state
- Initialize AFp to false in every state
- Apply inductive definition in each state until no values change
  - actual algorithm is a depth-first search, 1 pass over the graph

Model checking algorithm

- Decompose specification formula into a tree
- Each node $\Rightarrow$ one pass over the graph
- Example: $a$ and $b$:
  - Evaluate $a$ at each node
  - Evaluate $b$ at each node
  - Combine $a$ and $b$ at each node
- For temporal connectives, node values propogate along edges; order of evaluation is important for 1-pass evaluation
Fixed points

- A fixed point of a function $f$ is a value $x$ such that $f(x) = x$
- A set of equations (constraints) may have a set of solutions (fixed points), among them a least fixed point
- Inductive definitions of temporal connectives can be formulated as finding a least fixed point solution

Temporal logic & fixed points

- $\text{AF } p \equiv p \text{ or } \text{AX } \text{AF } p$
- $\text{EF } p \equiv p \text{ or } \text{EX } \text{EF } p$
- $\text{AG } p \equiv p \text{ and } \text{AX } \text{AG } p$
- $p \text{ AU } q \equiv q \text{ or } ( p \text{ and } \text{AX } ( p \text{ AU } q ))$

“AX” and “EX” mean: Look at (all, any) of the edges from this node to its successors. The inductive definitions become a set of constraints, and a fixed point solution gives the value of the temporal formulae at each node.
Expressiveness of CTL

• There is no CTL equivalent for
  $\text{GF } p \Rightarrow \text{GF } q$
  • And this does come up in practice!
    - Example: If at least some packets get through, the
      protocol will eventually deliver a message
  • Solution: Hack the algorithm
    • Hard-wire the fairness property into the model checking
      algorithm
    • See Clarke, Emerson, Sistla 85 (Toplas) for details

Complexity and Expressiveness

• Restricted branching time logics: CTL, LTAC
  - linear time checking procedures: $|f| \times |M|$
• Linear time logic: PTL
  - $2^{|f|} \times |M|$
    • Why? Because formula is evaluated (in the worst case) on all
      paths.
• Cheap extensions:
  - arbitrary state machines as temporal connectives
  - PTL to CTL* (linear time to unrestricted branching
    time)
Symbolic Model Checking

• The model (graph) could be very large.
• Q: Can we do better than explicitly evaluating formulae in every state?
• A: Not always, but sometimes symbolic representations and lazy evaluation help
• Represent graph as next-state function (symbolically), represent formula as evaluation