Formal Methods - State of the Art

from Edmund Clarke and Jeannette Wing, "Formal Methods: Sate of the Art and Future Directions", ACM Computing Surveys, December 1996.

from John Rushby and N. Shankar, "Theorem Proving and Model Checking for Software", FSE tutorial, October 1996.

these will be used throughout the semester.

Model-checking

Specify
- The system as a labeled transition system (interacting finite state machines)
- A required property as a formula in a temporal logic. Establish that the system is a model of the formula

system $\models$ *requirement*

Accomplished by a search over the state space
Guaranteed to terminate since the model is finite
in CTL - linear in # of states and length of formula

Usually no background theories (sometimes limited arithmetic)
Usually, the only assumptions concern interpretation of liveness (branching vs. linear time, fairness)
- Built in to choice of logic and selection of operators

Model-checking (Cont'd)

- Dominant approach uses Computation Tree Logic (CTL)
  - Mechanized by SMV
- A branching time logic: characterizes properties of states in the tree that represents possible paths in a computation
- Linear time logic (mechanized by SPIN - Bell Labs) characterizes properties of individual computations (i.e., sequences of states)
  - Model-checking is exponential in length of formula

Issues in Model-checking

Which logic?
- Model checking for branching time can be linear in length of the formula and the size of the state space
- For linear time, exponential in length of formula
- Many liveness properties cannot be stated in branching time logics

Varieties of approach:
- Temporal model checking - building a model and checking that temporal logic property holds
- Comparing two automatons (spec and system) for the same behavior

Issues in Model-checking (Cont'd)

Main issue is the size of state space: the "state explosion problem"

- Can use partial order methods to avoid enumerating computations that differ only in order of interleaving of independent operations
- Can use symmetry reductions to avoid enumerating states that differ only in irrelevant ways (e.g., which processor has failed when all that matters is how many have failed)
- Explicit state enumeration systems can generally build state space "on the fly" and halt when error is found
- Can typically explore millions of states in a few hours

Pros and Cons

- Often cannot express full requirements
  - instead, check several smaller properties
- Few real systems have sufficiently small state space to allow direct checking
  - must generally abstract them or "downscale" them
- Largely automatic and fast (modulo BDD orderings)
- Produces counterexamples
- Generally used for debugging
- Usually, find more problems by exploring *all* the behaviors of a downscaled system than by testing only *some* of the behaviors of full system

Symbolic Model Checking

- (e.g. SMV) represents states symbolically using Ordered Binary Decision Diagrams (OBDDs)
- Counterexamples can be still produced
- Many explicit states may have same symbolic representation
- Can allow huge state spaces to be examined (10^{120})
- Abstraction may enable checking systems with virtually unlimited number of states
- Can handle systems with 100-200 state variables
- Finding good variable ordering is crucial (can take hours and maybe there is no good ordering)

Model-checking Examples

IEEE Futurebus+ (1992, Clarke, CMU)

- Used SMV to verify the cache coherence protocol described in IEEE Futurebus+ Standard 896.1-1991
- Discovered a number of previously undetected errors in the design
- First attempt to find errors in an IEEE standard

IEEE SCI (1992, Dill, Stanford)

- Murphi verification system
- Verified cache coherence protocol of Scalable Coherent Interface, IEEE Standard 1596-1992
- Based their model directly on the C code given as definition of the standard
- Verified just small instances of the system
- Found several errors, ranging from omissions of variable initializations ti subtle logical errors
- The protocol has already been simulated and implemented!!

Model-checking Examples (Cont'd)

Stereo Components (1994-96)

- 1994. Proved (manually) correctness of a control protocol used in Philips stereo components
- 1995. Verified an abstraction of protocol using symbolic model checker HyTech and inferred more efficient timing of the protocol
- 1996. Model checked the entire protocol

ISDN/ISUP (1989-92)

- NewCoRe Project - first full-scale application of formal verification methods in routine software design project within AT&T
- 5 people formalized 145 requirements in temporal logic and did proofs with a special-purpose model checker
- 7500 lines of Specification and Description Language (SDL) source code verified
- 112 errors were revealed in high-level designs

Model-checking Examples (Cont'd)

PowerScale (1995, Bull with Berimag Laboratory)

- Used LOTOS to describe processors, memory controller, and bus arbiter of the multiprocessor architecture PowerScale
- Architecture is based on IBM's Power PC microprocessor
- Identified four correctness properties expressing essential requirements for proper functioning of the arbitration algorithm
- Established correctness automatically in a few minutes

Model-checking Examples (Cont'd)

Buildings (1995, North Caroline State)

- Used Concurrency Workbench to analyze timing properties of a distributed active structural control system
- System was designed to make buildings more resistant to earthquakes
- First coded in a timed version of the CCS. Resulting model contained over $2.12 * 10^{19}$ states and was not analyzable
- Applied semantic minimization feature of Concurrency Workbench
- Analyzed properties and uncovered an error in a timer setting
- If undetected, this error would cause active structural control component to worsen, rather than dampen, the vibration experienced by buildings during earthquakes

State of the Art Model Checkers

Temporal logic model checkers

- EMC - Clarke and Emerson 81-86
- CESAR - Queille and Sifakis 1982
- SMV - McMillan 1993 - first to use BDDs
- Spin - Gerth et al. 1995 - uses partial order reduction to reduce the state explosion problem
- Murphi - Dill et al. 1992 is based on Unity programming language
- Concurrency Workbench - Cleaveland et al. 1993 - verifies CCS processes for properties expressed as mu-calculus formulas
- HyTech - Alur et al. 1996 - for hybrid systems
- Kronos - Daws and Yovine 1995 - for real-time systems
- FORMAT - Damm et al. 1996, SVE - Filkorn et al. 1994 and CV - Deharbe and Borrione 1995 - focus on hardware verification

State of the Art Model Checkers (Cont'd)

Behavior conformance checkers

- Cospan/FormalCheck system - DePalma and Glaser 1996 - based on showing inclusion between omega automata
- FDR - Roscoe 1994 - checks refinement between CSP programs
- Concurrency Workbench checks a similar notion of refinement between CCS programs
- Concurrency Workbench and Auto - Roy and de Simone 1990 - can be used to minimize systems with respect to observational equivalence and to determine if two systems are observably correct

State of the Art Model Checkers (Cont'd)

Combination checkers

- Berkeley's HSIS - Hojati et al. 1993 - combines model checking with language inclusion
- Stanford's STep - Bjorner et al. 1996 - with deductive methods
- VIS - Vrayton et al. 1996 - with logic synthesis
- PVS Theorem prover - Owre, Rushby, Shankar 1992 - has a model checker for the model mu-calculus
- METAFrame - Steffen et al. 1996 - an environment that supports model checking in the entire software development process.

Theorem Proving Approaches

Specify
- The system (at some suitable level of abstraction)
- A required property
- The assumptions
- Necessary background theories as formulas in a single logic

Prove that

background + *assumptions* + *system* $\models$ *requirement*

Variation
- Prove that an implementation is a refinement of a specification

Classical commutative diagram, or theory interpretation

Techniques and Issues

How rich a logic?
- Easier to automate restricted logics (e.g., unquantified)
- But can make for awkward specifications

Interactive guidance vs. automation
- Really, need both; the issue is *balance*

Automation: decision procedures.
- Propositional calculus (NP complete at least)
- Equality over uninterpreted function symbols
- Linear arithmetic over rationals and integers
- Functional updates (arrays, stores), i.e.

f **with** $[(x := y](z) \stackrel{\text{def}}{=}$ if $z = x$ **then** y **else** $f(z)$

Techniques and Issues

Automation: Rewriting
- Provides decision procedures if rules are finite terminating
- More often used heuristically
- Hard to tell when theorem is wrong rather than we failed to find a proof

Automation: Heuristics
- First order: resolution (like in Prolog), etc.
- Induction can help in proofs over infinite domains
- Difficult to interpret failure

Pros and Cons of Theorem Proving

- Specialized methods can be very effective on a limited domain (e.g., boolean equivalence checking in CAD tools)
- General methods can state and prove any true property given enough time, skill and patience
- Often require too much time, skill and patience
- Many theorem provers are poorly matched to the requirements of formal methods
  1) lack adequate automation
  2) do not support civilized specification language, nor theories needed for computer science
  3) do not fail gracefully
     - false conjecture or inadequate heuristic?
     - do not help pinpoint error

Theorem Proving - Examples

SRT division algorithm (1995, Clarke, German and Zhao)
- Used automatic theorem-proving techniques based on symbolic algebraic manipulation to prove the correctness of an SRT division algorithm line the one in the Pentium
- Verification method runs automatically and count have detected the error in the Pentium, caused by a faulty quotient digit selection table
- SRI's PVS was also used on this same example (1996)

Theorem Proving - Examples (Cont'd)

Processor Designs
- Verity verification tool is widely used within IBM in design of PowerPC and System/390
- Tool can handle entire processor designs containing millions of transistors (when applied in hierarchical manner)
- Can model the functional behavior of a hardware system as a boolean state transition function at register transfer level, gate level, or transistor level
- Use BDDs to check the equivalence of the state transition functions at different design levels

Theorem Proving - Examples (Cont'd)

Motorola 68020 (Boyer and Yu, 1991-95)

- Constructed an Nqthm specification of Motorola 68020 microprocessor
- Specification included 80 - Used specification to prove correctness of many binary machine code programs produced by commercial compilers from Lisp, C, Ada
- Verified MC68020 binary code produced by gcc compiler for 21 of 22 programs in the Berkeley string library

AAMP5 (1993-95, Srivas, Stanford and Miller, Rockwell)

- specified and verified Collins Commercial Avionics AAMP5 microprocessor
- used PVS to specify 108 of the 209 AAMP5 instructions and verified the microcode for 11 representative instructions

Theorem Proving - Examples (Cont'd)

AMD5K86 (1995, Moore and Kaufmann of Computational Logic, Inc. and Lynch of Advanced Micro Devices, Inc)

- Proved correctness of Lynch's microcode for floating point division on the AMDK86
- Started from informal proof of correctness
- Formalized proof in the ACL2 logic and checked with ACL2 mechanical theorem prover
- Erros were found in informal "proof" but the microcode was correct
- Effort took nine weeks

Theorem Proving - Examples (Cont'd)

Motorola CAP (1992-96 Brock of Computational Logic, Inc.)

- Developed an ACL2 specification of the entire Motorola Complex Arithmetic Processor (CAP)
- Most complicated microprocessor yet formalized
- Three state pipeline, six independent memories, four multiplier-accumulators, over 250 programmer-visible registers
- Instruction set allows simultaneous modification of over 100 registers in a single instruction
- Used ACL2 to verify binary microcode for several digital signal processing algorithms

State of the Art Theorem Provers

User-guided automatic deduction tools

- Systems are guided by a sequence of lemmas and definitions but each theorem is proved automatically using builtin heuristics for induction, lemma-driven rewriting and simplification

ACL2 [Kaufmann and Moore 1995], Eves [Cragen et al 1988], LP [Garland and Guttag 1988], Nqthm [Boyer and Moore 1979], Reve [Lescanne 1983], RRL [Kapur and Musser 1987]

- Nqthm was used to check a proof of Godel's first incompleteness theorem

Proof checkers

- Used to formalize and verify hard problems in mathematics and in program verification
- Coq [Cornes et al. 1995], HOL [Gordon 1987], LEGO [Luo and Pollack 1992], LCF [Gordon et al. 1979], Nuprl [Constable et al. 1986]

State of the Art Theorem Provers

Combination provers

- Analytica [Clarke and Zhao 1993] - combines theorem proving with symbolic algebra system Mathematica. Proved some hard number-theoretic problems
- PVS and STep combine decision procedures and model checking with interactive proof
- PVS was used to verify a number of hardware designs and reactive, real-time and fault-tolerant algorithms

food for slide eater

food for slide eater

food for slide eater