Formal Verification

Lecture 10

Formal Verification

- Formal verification relies on
  - Descriptions of the properties or requirements of interest
  - Descriptions of systems to be analyzed, and
    - rely on underlying mathematical logic system and the proof theory of that system
- Two general categories
  - Inductive techniques
  - Model checking techniques

Verification techniques

- Proof-based vs model-based
  - Proof: Formula define premises / conclusions
    - Proof shows how to reach conclusions from premises
  - Model-based:
    - Premises and conclusions have same truth tables
- Degree of automation
  - may be manual or have tool support
- Full verification vs property verification
    - Does methodology model full system?
    - Or just prove certain key properties?
- Intended domain of application
  - HW/SW, reactive, concurrent
- Predevelopment vs post development
  - As design aid or after design

Inductive verification

- Typically more general
- Uses theorem provers
  - E.g., uses predicate calculus
  - A sequence of proof steps starting with premises of the formula and eventually reaching a conclusion
- May be used
  - To find flaws in design
  - To verify the properties of computer programs

Model-checking

- Systems modeled as state transition systems
  - Formula may be true in some states and false in others
  - Formulas may change values as systems evolve
- Properties are formulas in temporal logic
  - Truth values are dynamic
- Model and the desired properties are semantically equivalent
  - Model and properties express the same truth table
- Often used after development is complete but before a product is release to the general market

Formal Verification: Components

- Formal Specification
  - defined in unambiguous (mathematical) language
  - Restricted syntax, and well-defined semantics based on established mathematical concepts
  - Example: security policy models (Take-Grant, BLP)
- Implementation Language
  - Generally somewhat constrained
- Formal Semantics relating the two
- Methodology to ensure implementation ensures specifications met

Specification Languages

- Specify WHAT, not HOW
  - Valid states of system
  - Postconditions of operations
- Non-Procedural
- Typical Examples:
  - Propositional / Predicate Logic
  - Temporal Logic (supports before/after conditions)
  - Set-based models (e.g., formal Bell-LaPadula)

Specification Languages

- Must support machine processing
  - Strong typing
  - Model input/output/errors
- Example: SPECIAL
  - First order logic base
  - Strongly typed
  - VFUN: describes variables (state)
  - OFUN/OVFUN: describe state transitions

Example: SPECIAL

- MODULE Bell_LaPadula_Model Give_read
- Types
- Subject_ID: DESIGNATOR;
- Object_ID: DESIGNATOR;
- Access_Model: {READ, APPEND, WRITE};
- Access: STRUCT_OF(Subject_ID subject; Object_ID object; Access_Mode mode);
- Functions
- VFUN active (Object_ID object) -> BOOLEAN active: HIDDEN; INITIALLY TRUE;
- VFUN access_matrix() -> Access accesses: HIDDEN; INITIALLY FORALL Access a: a INSERT accesses => active(a.object);
- OFUN give_access(Subject_ID giver; Access access); ASSERTIONS active(access.object) = TRUE; EFFECTS `access_matrix() = access_matrix() UNION (access);
- END_MODULE

Example: Enhanced Hierarchical Development Methodology

- Based on HDM
  - A general purpose design and implementation methodology
  - Goal was
    - To mechanize and formalize the entire development process
    - Design specification and verification + implementation specification and verification
- Proof-based method
  - Uses Boyer-Moore Theorem Prover

Example: Enhanced Hierarchical Development Methodology

- Hierarchical approach
  - *Abstract Machines* defined at each level
    - specification written in SPECIAL
  - *Mapping Specifications* define functionality in terms of machines at higher layers
  - *Consistency Checker* validates mappings “match”
- Compiler that maps a program into a theorem-prover understood form
- Successfully used on MLS systems
  - Few formal policy specifications outside MLS domain

Alternate Approach: Combine Specifications and Language

- Gypsy verification environment (GVE)
- Specifications defined on procedures
  - Entry conditions
  - Exit conditions
  - Assertions
- Proof techniques ensure exit conditions / assertions met given entry conditions
  - Also run-time checking
- Examples:
  - Gypsy (in book) – uses theorem prover
  - CLU
  - Eiffel (and derivatives) – run-time checks

Other Examples

- Prototype Verification System (PVS)
  - Based on EHDM
  - Interactive theorem-prover
- Symbolic Model Verifier
  - Temporal logic based
  - Notion of “path” – program represented as tree
  - Statements that condition must hold at *a* future state, *all* future states, all states on one path, etc.

Other Examples

- Formal verification of protocols
  - Key management
  - Protocol development
- Verification of libraries
  - Entire system not verified
  - But components known okay
- High risk subsystems

Protocol Verification

- Generating protocols that meet security specifications
- Assumes cryptography secure
  - But cryptography not enough