Automated Theorem Proving and Proof Checking

Engler: Automatically Generating Malicious Disks using Symex

- IEEE Security and Privacy 2006
- Use CIL and Symbolic Execution on Linux FS code
- Special model of memory, makes theorem prover calls, aims to hit all paths, has trouble with loops
- New: transform program so that it combines concrete and symbolic execution (cf. RTCG)
- New: uses constraint solver to automatically generate test case (= FS image)
- Found 5 bugs (4 panic, 1 root)
- Special thanks to Wei Hu for noticing this ...

Cunning Plan

- There are full-semester courses on automated deduction; we will elide details.
- Logic Syntax
- Theories
- Satisfiability Procedures
  - Mixed Theories
- Theorem Proving
- Proof Checking
- SAT-based Theorem Provers (cf. Engler paper)

Motivation

- Can be viewed as “decidable AI”
  - Would be nice to have a procedure to automatically reason from premises to conclusions ...
- Used to rule out the exploration of infeasible paths (model checking, dataflow)
- Used to reason about the heap (McCarthy, symbolic execution)
- Used to automatically synthesize programs from specifications (e.g. Leroy, Engler optional papers)
- Used to discover proofs of conjectures (e.g., Tarski conjecture proved by machine in 1996, efficient geometry theorem provers)
- Generally under-utilized

History

- Automated deduction is logical deduction performed by a machine
- Involves logic and mathematics
- One of the oldest and technically deepest fields of computer science
  - Some results are as much as 75 years old
  - “Checking a Large Routine”, Turing 1949
  - Automation efforts are about 40 years old
  - Floyd-Hoare axiomatic semantics
- Still experimental (even after 40 years)

Standard Architecture
Logic Grammar

- We’ll use the following logic:
- Goals:  \[ G ::= L | \text{true} | G_1 \land G_2 | H \Rightarrow G | \forall x. G \]
- Hypotheses:  \[ H ::= L | \text{true} | H_1 \land H_2 \]
- Literals:  \[ L ::= p(E_1, \ldots, E_k) \]
- Expressions:  \[ E ::= n | f(E_1, \ldots, E_m) \]

- This is a subset of first-order logic
  - Intentionally restricted: no \( \lor \) so far
  - Predicate functions: \( p: <, =, \ldots \)
  - Expression functions: \( f: +, *, \text{sel}, \text{upd}, = \)

Theorem Proving Problem

- Write an algorithm “prove” such that:
- If \( \text{prove}(G) = \text{true} \) then \( \models G \)
  - Soundness (must have)
- If \( \models G \) then \( \text{prove}(G) = \text{true} \)
  - Completeness (nice to have, optional)
- \( \text{prove}(H, G) \) means prove \( H \Rightarrow G \)
- Architecture: Separation of Concerns
  - #1. Handle \( \land, \Rightarrow, \forall, = \)
  - #2. Handle \( /FL00B7h, *, \text{sel}, \text{upd}, = \)

Theorem Proving

- Want to prove true things
- Avoid proving false things
- We’ll do proof-checking later to rule out the “cat proof” shown here
- For now, let’s just get to the point where we can prove something

Basic Symbolic Theorem Prover

- Let’s define \( \text{prove}(H, G) \) ...
- \( \text{prove}(H, \text{true}) = \text{true} \)
- \( \text{prove}(H, G_1 \land G_2) = \text{prove}(H, G_1) \land \text{prove}(H, G_2) \)
- \( \text{prove}(H_1, H_2 \Rightarrow G) = \text{prove}(H_1, H_2 \land G) \)
- \( \text{prove}(H, \forall x. G) = \text{prove}(H, G[a/x]) \)
  - (\( a \) is “fresh”)
- \( \text{prove}(H, L) = ??? \)

Theorem Prover for Literals

- We have reduced the problem to \( \text{prove}(H, L) \)
- But \( H \) is a conjunction of literals \( L_1 \land \ldots \land L_k \)
- Thus we really have to prove that \( L_1 \land \ldots \land L_k \Rightarrow L \)
  - Equivalently, that \( L_1 \land \ldots \land L_k \land \sim L \) is unsatisfiable
    - For any assignment of values to variables the truth value of the conjunction is false
  - Now we can say \( \text{prove}(H, L) = \text{Unsat}(H \land \sim L) \)

Theory Terminology

- A theory consists of a set of functions and predicate symbols (syntax) and definitions for the meanings of those symbols (semantics)
- Examples:
  - 0, 1, -1, 2, -3, ..., +, -, =, < (usual meanings; “theory of integers with arithmetic” or “Presburger arithmetic”)
  - =, < (axioms of transitivity, anti-symmetry, and \( \forall x. \forall y. x \leq y \lor y \leq x \); “theory of total orders”)
  - sel, upd (McCarthy’s “theory of lists”)
Decision Procedures for Theories

- The Decision Problem
  - Decide whether a formula in a theory with first-order logic is true
- Example:
  - Decide “∀x. x>0 ⇒ (∃y. x=y+1)” in \([\mathbb{N}, +, =, >]\)
- A theory is **decidable** when there is an algorithm that solves the decision problem
  - This algorithm is the **decision procedure** for that theory

Satisfiability Procedures

- The Satisfiability Problem
  - Decide whether a conjunction of literals in the theory is satisfiable
  - Factors out the first-order logic part
  - The decision problem can be reduced to the satisfiability problem
    - Parameters for ∀, skolem functions for ∃, negate and convert to DNF (sorry; I won’t explain this here)
- “Easiest” Theory = Propositional Logic = **SAT**
  - A decision procedure for it is a “**SAT solver**”

Theory of Equality

- Theory of equality with uninterpreted functions
  - Symbols: =, ≠, f, g, ...
- Axiomatically defined (A,B,C ∈ Expressions):
  - \(A = A\)
  - \(B = A\)
  - \(A = B\)
  - \(A = C\)
  - \(f(A) = f(B)\)
- Example satisfiability problem:
  - \(g(g(g(x))) = x \land g(g(g(g(g(x))))) = x \land g(x) ≠ x\)

More Satisfying Examples

- Theory of Linear Arithmetic
  - Symbols: ≥, =, +, -, integers
  - Example: \(y > 2x + 1, x > 1, y < 0\) is unsat
  - Satisfiability problem is in P (loosely, no multiplication means no tricky encodings)
- Theory of Lists
  - Symbols: cons, head, tail, nil
    - Theorem: head(x) = head(y) \land tail(x) = tail(y) ⇒ x = y

Mixed Theories

- Often we have facts involving **symbols from multiple theories**
  - E’s symbols =, ≠, f, g, ...
  - R’s symbols =, ≠, +, -, ≤, 0, 1, ...
  - Running Example (and Fact):
    - \(x ≤ y \land y + z ≤ x \land 0 ≤ z \Rightarrow f(f(x) - f(y)) = f(z)\)
  - To prove this, we must decide:
    - Unsat(x ≤ y, y + z ≤ x, 0 ≤ z, f(f(x) - f(y)) ≠ f(z))
- We may have a sat procedure for each theory
  - E’s sat procedure by Ackermann in 1924
  - R’s proc by Fourier
- The sat proc for their combination is much harder
  - Only in 1979 did we get E+R

Satisfiability of Mixed Theories

- Can we just separate out the terms in Theory 1 from the terms in Theory 2 and see if they are separately satisfiable?
  - No, unsound, equi-sat ≠ equivalent.
- The problem is that the two satisfying assignments may be incompatible
- Idea (Nelson and Oppen): Each sat proc announces all equalities between variables that it discovers
Handling Multiple Theories

- We’ll use cooperating decision procedures
- Each sat proc works on the literals it understands
- Sat procs share information (equalities)

Consider Equality and Arith

\[
\begin{align*}
  f(f(x) - f(y)) &\neq f(z) \\
  x &< y \\
  y + z &< x \\
  0 &< z
\end{align*}
\]

- How can we do this in our prover?

Nelson-Oppen: The E-DAG

- Represent all terms in one Equivalence DAG
  - Node names act as variables shared between theories!
  \[
  f(f(x) - f(y)) \neq f(z) \land y \geq x \land x \geq y + z \land z \geq 0
  \]

Nelson-Oppen: Processing

- Run each sat proc
  - Report all contradictions (as usual)
  - Report all equalities between nodes (key idea)

Implementation details: Use union-find to track node equivalence classes in E-DAG. When merging \( A = B \), also merge \( f(A) = f(B) \).

Nelson-Oppen: Processing

- Broadcast all discovered equalities
  - Rerun sat procedures
  - Until no more equalities or a contradiction

Does It Work?

- If a contradiction is found, then unsat
  - This is sound if sat procs are sound
  - Because only sound equalities are ever found
- If there are no more equalities, then sat
  - Is this complete? Have they shared enough info?
  - Are the two satisfying assignments compatible?
  - Yes!
  - (Countable theories with infinite models admit isomorphic models, convex theories have necessary interpretations, etc.)
Proofs
“Checking proofs ain’t like dustin’ crops, boy!”

Proof Generation
- We want our theorem prover to emit proofs
  - No need to trust the prover
  - Can find bugs in the prover
  - Can be used for proof-carrying code
  - Can be used to extract invariants
  - Can be used to extract models
- Implements the soundness argument
  - On every run, a soundness proof is constructed

Proof Representation
- Proofs are trees
  - Leaves are hypotheses/axioms
  - Internal nodes are inference rules
- Axiom: “true introduction”
  - Constant: true : pf
  - pf is the type of proofs
- Inference: “conjunction introduction”
  - Constant: and : pf → pf → pf
- Inference: “conjunction elimination”
  - Constant: andl : pf → pf → pf
- Problem:
  - “andel truei : pf” but does not represent a valid proof
  - Need a more powerful type system that checks content

Dependent Types
- Make pf a family of types indexed by formulas
  - f : Type (type of encodings of formulas)
  - e : Type (type of encodings of expressions)
  - pf : f → Type (the type of proofs indexed by formulas: it is a proof that f is true)
- Examples:
  - true : f
  - and : f → f → f
  - truei : pf true
  - andi : pf A → pf B → pf (and A B)
- andi : Π A:f. Π B:f. pf A → pf B → pf (and A B)

Proof Checking
- Validate proof trees by type-checking them
- Given a proof tree X claiming to prove A ∧ B
- Must check X : pf (and A B)
- We use “expression tree equality”, so
  - andl (andi “1+2=3” “x=y”) does not have type pf (3=3)
  - This is already a proof system! If the proof-supplier wants to use the fact that 1+2=3 ⇔ 3=3, she can include a proof of it somewhere!
- Thus Type Checking = Proof Checking
  - And it’s quite easily decidable!

Parametric Judgment
- Universal Introduction Rule of Inference
  - \[ [a/x]A \quad (a \text{ is fresh}) \]
  - \[ \vdash \forall x. A \]
- We represent bound variables in the logic using bound variables in the meta-logic
  - all : (e → f) → f
  - Example: \( \forall x. x=x \) represented as (all (λ x.eq x x))
  - Note: \( \forall y. y=y \) has an \( \alpha \)-equivalent representation
  - Substitution is done by \( \beta \)-reduction in meta-logic
  - \( [E/x](x=x) \) is \( (\lambda x.eq x x) E \)
Parametric $\forall$ Proof Rules

\[ \vdash [a/x]A \quad (a \text{ is fresh}) \]
\[ \vdash \forall x. A \]

- Universal Introduction
  \[ \Pi A : (e \rightarrow f). \ (\Pi a : e. \ pf (A a)) \rightarrow pf (\forall A) \]

- Universal Elimination
  \[ \Pi A : (e \rightarrow f). \ Pi E : e. \ pf (\forall A) \rightarrow pf (A E) \]

Parametric $\exists$ Proof Rules

\[ \vdash [E/x]A \]
\[ \vdash \exists x. A \]

- Existential Introduction
  \[ \Pi A : (e \rightarrow f). \ Pi E : e. \ pf (\exists A) \rightarrow pf (exists A) \]

- Existential Elimination
  \[ \Pi A : (e \rightarrow f). \ Pi B : f. \ pf (\exists A) \rightarrow (\Pi a : e. \ pf (A a) \rightarrow pf B) \rightarrow pf B \]

SAT-Based Theorem Provers

- Recall separation of concerns:
  - #1 Prover handles connectives ($\forall$, $\land$, $\Rightarrow$)
  - #2 Sat procs handle literals ($+$, $\leq$, $0$, head)

- Idea: reduce proof obligation into propositional logic, feed to SAT solver (CVC)

  - To Prove: $3x=9 \Rightarrow (x = 7 \land x \leq 4)$
  - Becomes Prove: $A \Rightarrow (B \land C)$
  - Becomes Unsat: $A \land (\neg B \lor \neg C)$
  - Becomes Unsat: $A \land (\neg B \lor \neg C)$

SAT-Based Theorem Proving

- To Prove: $3x=9 \Rightarrow (x = 7 \land x \leq 4)$
  - Becomes Unsat: $A \land (\neg B \lor \neg C)$
  - SAT Solver Returns: $A=1, C=0$
  - Ask sat proc: unsat($3x=9, \neg x=7) = true$
  - Add constraint: $(\neg (A \land C))$
  - Becomes Unsat: $A \land (\neg B \lor \neg C) \land (\neg (A \land C))$
  - SAT Solver Returns: $A=1, B=0$
  - Ask sat proc: unsat($3x=9, \neg x=7) = false$
  - $(x=3$ is a satisfying assignment)
  - We’re done! (original to-prove goal is false)
  - If SAT Solver returns “no satisfying assignment” then original to-prove goal is true

Homework

- Project Status Update
- Project Due Tue Apr 25
  - You have ~21 days to complete it.
  - Need help? Stop by my office or send email.