Cunning Theorem-Proving 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

Program

Specification

Verification Condition Generation

Theorem In A Logic

Semantics

Validity

Provability

Meets Spec Or Found A Bug

Automated Theorem Proving
Logic Grammar

• We’ll use the following logic:

Goals: \[ G ::= L \mid \text{true} \mid G_1 \land G_2 \mid H \Rightarrow G \mid \forall x. G \]

Hypotheses: \[ H ::= L \mid \text{true} \mid H_1 \land H_2 \]

Literals: \[ L ::= p(E_1, \ldots, E_k) \]

Expressions: \[ E ::= n \mid f(E_1, \ldots, E_m) \]

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

\[ \text{prove}(H, \text{true}) = \text{true} \]
\[ \text{prove}(H, G_1 \land G_2) = \text{prove}(H, G_1) \land \text{prove}(H, G_2) \]
\[ \text{prove}(H_1, H_2 \Rightarrow G) = \text{prove}(H_1 \land H_2, G) \]
\[ \text{prove}(H, \forall x. G) = \text{prove}(H, G[a/x]) \quad (a \text{ is } \text{“fresh”}) \]
\[ \text{prove}(H, L) = ??? \]
Theorem Prover for Literals

- We have reduced the problem to 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 $L_1 \land \ldots \land L_k \land \neg L$ is unsatisfiable
  - For any assignment of values to variables the truth value of the conjunction is false
- Now we can say $\text{prove}(H, L) = \text{Unsat}(H \land \neg L)$
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 ∀x. ∀y. x ≤ y ∨ y ≤ 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 \( \forall x. x>0 \Rightarrow (\exists y. x=y+1) \) 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 $\forall$, skolem functions for $\exists$, 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):

  \[
  \begin{align*}
  & A=A \\
  & A=B \\
  & B=A \\
  & A=\text{B} \\
  & B=\text{C} \\
  & A=B \\
  & f(A) = f(B)
  \end{align*}
  \]

• Example satisfiability problem:

  \[
  g(g(g(x)))=x \land g(g(g(g(g(x)))))=x \land g(x)\neq x
  \]
More Satisfying Examples

• Theory of **Linear Arithmetic**
  - Symbols: $\geq$, $=$, $+$, $-$, 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: $\text{head}(\text{cons}(A,B)) = A$ and $\text{tail}(\text{cons}(A,B)) = B$
  - Theorem: $\text{head}(x) = \text{head}(y)$ and $\text{tail}(x) = \text{tail}(y) \Rightarrow x = y$
Mixed Theories

• Often we have facts involving symbols from multiple theories
  - E’s symbols $=$, $\neq$, $f$, $g$, ... (uninterp function equality)
  - R’s symbols $=$, $\neq$, $+$, $-$, $\leq$, $0$, $1$, ... (linear arithmetic)
  - Running Example (and Fact):
    $\vdash x \leq y \land y + z \leq x \land 0 \leq z \Rightarrow f(f(x) - f(y)) = f(z)$
  - To prove this, we must decide:
    $\text{Unsat}(x \leq y, y + z \leq x, 0 \leq z, f(f(x) - f(y)) \neq f(z))$

• 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 $\neq$ 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**

- $f(f(x) - f(y)) \neq f(z)$
- $x \leq y$
- $y + z \leq x$
- $0 \leq z$
- $x = y$
- $y \leq x$
- $0 = z$
- $f(x) = f(y)$
- $f(x) - f(y) = z$

**How can we do this in our prover?**

- $f(f(x) - f(y)) = f(z)$

**false**
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.)*
SAT-Based Theorem Provers

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

• Idea: reduce proof obligation into propositional logic, feed to SAT solver (CVC)
  - To Prove: $3*x=9 \Rightarrow (x = 7 \wedge x \leq 4)$
  - Becomes Prove: $A \Rightarrow (B \wedge C)$
  - Becomes Unsat: $A \wedge \neg(B \wedge C)$
  - Becomes Unsat: $A \wedge (\neg B \vee \neg C)$
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($3x=9$, $\neg x\leq4$) = true
  - Add constraint: $\neg(A \land \neg C)$
  - Becomes Unsat: $A \land (\neg B \lor \neg C) \land \neg(A \land \neg C)$
  - SAT Solver Returns: $A=1$, $B=0$, $C=1$
  - Ask sat proc: unsat($3x=9$, $\neg x=7$, $x\leq4$) = 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
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 (e.g., in SLAM)

• 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: \( \text{truei} : \text{pf} \)
  - \( \text{pf} \) is the type of proofs

• Inference: “conjunction introduction”
  - Constant: \( \text{andi} : \text{pf} \rightarrow \text{pf} \rightarrow \text{pf} \)

• Inference: “conjunction elimination”
  - Constant: \( \text{andel} : \text{pf} \rightarrow \text{Pf} \)

• Problem:
  - “andel truei : pf” but does not represent a valid proof
  - Need a more powerful \textit{type system that checks content}
Dependent Types

• Make \( \text{pf} \) a family of types indexed by formulas
  - \( f : \text{Type} \) (type of encodings of formulas)
  - \( e : \text{Type} \) (type of encodings of expressions)
  - \( \text{pf} : f \rightarrow \text{Type} \) (the type of proofs indexed by formulas: it is a proof \textit{that } \( f \text{ is true} \))

• Examples:
  - \( \text{true} : f \)
  - \( \text{and} : f \rightarrow f \rightarrow f \)
  - \( \text{truei} : \text{pf true} \)
  - \( \text{andi} : \text{pf A} \rightarrow \text{pf B} \rightarrow \text{pf (and A B)} \)
  - \( \text{andi} : \prod A:f. \prod B:f. \text{pf A} \rightarrow \text{pf B} \rightarrow \text{pf (and A B)} \)
Proof Checking

- Validate proof trees by **type-checking** them
- Given a proof tree X claiming to prove $A \land B$
- Must check $X: \text{pf} \ (\text{and} \ A \ B)$
- We use “expression tree equality”, so
  - andel (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 $1+2=3 \iff 3=3$, she can include a proof of it somewhere!

- **Thus** **Type Checking = Proof Checking**
  - And it’s quite easily **decidable**!
Parametric Judgment (Time?)

• Universal Introduction Rule of Inference

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

• We represent bound variables in the logic using bound variables in the meta-logic
  - all : (e \rightarrow f) \rightarrow f
  - Example: \( \forall x. x=x \) represented as \((\text{all} \ (\lambda x. \text{eq} \ x \ x))\)
  - Note: \( \forall y. y=y \) has an \( \alpha \)-equivalent representation
  - Substitution is done by \( \beta \)-reduction in meta-logic
    - \([E/x](x=x) \) is \((\lambda x. \text{eq} \ x \ x) \ E\)
Parametric $\forall$ Proof Rules (Time?)

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

\[
\vdash \forall x. \ A
\]

- **Universal Introduction**
  - alli: $\Pi A: (e \to f). (\Pi a: e. \ pf (A a)) \to pf (\text{all } A)$

\[
\vdash \forall x. \ A
\]

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

- **Universal Elimination**
  - alle: $\Pi A: (e \to f). \Pi E: e. \ pf (\text{all } A) \to pf (A E)$
Parametric \( \exists \) Proof Rules (Time?)

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

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

\[
\vdash [a/x]A \\
\vdots \\
\vdash \exists x. A \\
\vdash B
\]

- **Existential Elimination**
  - existe: \( \Pi A : (e \rightarrow f) \). \( \Pi B : f. \)
  
  pf \( (\text{exists } A) \rightarrow (\Pi a : e. \text{ pf } (A a) \rightarrow \text{ pf } B) \rightarrow \text{ pf } B \)
Homework

• Project
  - Need help? Stop by my office or send email.