### Theorem Proving for FOL

**Satisfiability Procedures**

**CS 294-8**

**Lecture 11**

---

#### Review

- Recall the tool we 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:**
  \[ L ::= p(E_1, ..., E_n) \]

  **Expressions:**
  \[ E ::= n \mid f(E_1, ..., E_n) \]

- This is sufficient for **VCGen** if:
  - The invariants, preconditions and postconditions are all from **H**

---

#### A Simple and Complete Prover

- Define the following symbolic "prove" algorithm:
  - **Prove**(*H*, \( \text{true} \)) \( \Rightarrow \text{true} \)
  - **Prove**(*H*, \( G_1 \land G_2 \)) \( \Rightarrow \text{Prove}(G_1, \text{true}) \land \text{Prove}(G_2, \text{true}) \)
  - **Prove**(*H*, \( H_1 \Rightarrow H_2 \)) \( \Rightarrow \text{Prove}(H_1, \text{true}) \Rightarrow \text{Prove}(H_2, \text{true}) \)
  - **Prove**(*H*, \( \forall x. G \)) \( \Rightarrow \text{Prove}([G/x], \text{true}) \)
  - **Prove**(*H*, \( G \land L \)) \( \Rightarrow \text{Prove}(G, \text{true}) \land \text{Prove}(L, \text{true}) \)

- We have a simple, sound and complete prover
  - If we have a way to check unsatisfiability of sets of literals

---

#### How Powerful is Our Prover?

- With **VCGen** in mind we must restrict invariants to
  \[ H ::= L \mid \text{true} \mid H_1 \land H_2 \]

- No disjunction, implication or quantification!
  - Is that bad?

- Consider the function:
  ```c
  void insert(LIST *a, LIST *b) {
    LIST *t = a->next; a->next = b; b->next = t;
  }
  ```

- And the problem is to verify that
  - It preserves linearity: all list cells are pointed to by at most one other list cell
  - Provided that \( b \) is non-NULL and not pointed to by any cell

---

#### Lists and Linearity

- A bit of formal notation (remember the sel/udp):
  - We write \( \text{sel}(n, a) \) to denote the value of \( a->next \) given the state of the "next" field is \( a \)
  - We write \( \text{udp}(n, a, b) \) to denote the new state of the "next" field after \( a->next = b \)

- Code:
  ```c
  void insert(LIST *a, LIST *b) {
    LIST *t = a->next; a->next = b; b->next = t;
  }
  ```

- **Pre** is
  \[ (q, q = 0 \Rightarrow \forall p, p = \text{sel}(n, a) \Rightarrow q \Rightarrow p_2 = p_3) \]

- **Post** is
  \[ (q, q = 0 \Rightarrow \forall p, p = \text{sel}(n, a) \Rightarrow q \Rightarrow p_2 = p_3) \]

- **VC** is
  \[ \text{Pre} \Rightarrow \text{Post} \text{(udp}(n, a, b), b, \text{sel}(n, a)) / n \]

- Not a G!}

---

#### Two Solutions

- So it is quite easy to want to step outside \( H \)
  - And then extend the prover

- So what can we do?
  1. Extend the language of \( H \)
     - And then extend the prover
  2. Push the complexity of invariants into literals
     - And then extend the unsatisfiability procedure
**Goal Directed Theorem Proving (1)**
- Finally we extend the use of quantifiers:
  \[ G = \text{true} \mid G_1 \land G_2 \mid H \Rightarrow G \mid \forall x. G \mid \exists x. G \]
  \[ H = \text{true} \mid H_1 \land H_2 \mid \forall x. H \]
- We have now introduced an existential choice
  - Both in \( H \Rightarrow G \) and \( \forall x. H \Rightarrow G \)
- Existential choices are propagated
  - Introduce unification variables \( = \) unification
    \[ \text{prove}(H, G, H) = \text{prove}(H, G, H, u) \] (\( u \) is a unif var)
    \[ \text{prove}(H, u) = \text{instantiates} u \text{ with } t \text{ if } u = V(t) \]
- Still sound and complete goal directed proof search!
  - Provided that one can handle unification variables!

---

**Goal Directed Theorem Proving (2)**
- We can add disjunction (but only to goals):
  \[ G := \text{true} \mid G_1 \land G_2 \mid H = G \mid \forall x. G \mid G_1 \lor G_2 \]
- Extend prover as follows:
  \[ \text{prove}(H, G, G_1 \lor G_2) = \text{prove}(H, G, G_1) \lor \text{prove}(H, G, G_2) \]
- This introduces a choice point in proof search
  - Called a "disjunctive choice"
  - Backtracking is complete for this choice selection
    - But only in intuitionistic logic!

---

**Goal Directed Theorem Proving (3)**
- Now we extend the language of hypotheses
  - Important since this adds flexibility for invariants and specs.
    \[ H := \text{true} \mid H_1 \land H_2 \mid G \Rightarrow H \]
- We extend the prover as follows:
  \[ \text{prove}(H, G, H_1 \Rightarrow H_2) \Rightarrow G \]
  \[ \text{prove}(H_1 \Rightarrow H_2, G) = \text{prove}(H_1 \Rightarrow H_2, G) \]
  \[ \text{prove}(H_1 \Rightarrow H_2, G) = \text{prove}(H_1 \Rightarrow H_2, G) \]
  - This adds another choice (clause choice in Prolog) expressed here as a disjunctive choice
  - Still complete with backtracking

---

**Goal Directed Theorem Proving (4)**
- The VC for linear lists can be proved in this logic!
  - This logic is called Hereditary Harrop Formulas
- But the prover is not complete in a classical sense
  - And this complications might arise with certain theories
- Still no way to have disjunctive hypotheses
  - The prover becomes incomplete even in intuitionistic logic
    - \( E_g \), cannot prove even that \( P \lor Q = Q \lor P \)
- Let's try the other method instead ...

---

**A Theory of Linear Lists**
- Push the complexity into literals
  - Define new literals:
    \[ \text{linear}(s) \equiv \forall s \in \mathbb{N}, q + 0 = q, \forall p, p = p, \forall s, s = s \]
    \[ \text{rc}(p, b) \equiv \exists b > 0 \implies \text{sel}(p, p) = b \]
- Now the predicates become:
  - Pred is \( \text{linear} \)
  - Post is \( \text{linear} \)
- VC is \( \text{linear} \lor \text{rc}(0, b), a = 0, b = 0 \)
- This is a 0!
- The hard work is now in the satisfiability procedure

---

**A Theory of Lists**
- In order to allow the prover to work with "linear" and "rc0" we must define their meaning:
  - Semantically (by giving the definitions from before)
  - Axiomatically (by giving a set of axioms that define them)
    \[ \text{linear}(s) \quad a = 0 \quad \text{rc}(s, b) \]
    \[ \text{linear}(\text{up}(s, 0, a, b)) \]
    \[ \text{linear}(\text{rc}(s, a, b)) \]
- Now we can prove the VC with just three uses of these axioms
- Is this set of axioms complete?
**Discussion**

- It makes sense to push hard work in literals:
  - Can be handled in a customized way within the Sat procedures
  - The hard-crafted inference rules guide the prover
  - The inference rules are useful lemmas
- Important technique #3
- Just like in type inference, or data flow analysis:

<table>
<thead>
<tr>
<th>Theorem Proving</th>
<th>Type Inference</th>
<th>Data Flow Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Literals</td>
<td>Type system</td>
<td>Lattice</td>
</tr>
<tr>
<td>Inference rules</td>
<td>Typing rules</td>
<td>Transfer functions</td>
</tr>
<tr>
<td>Sat. procedure</td>
<td>Inference algorithm</td>
<td>Iterative D.F.A.</td>
</tr>
</tbody>
</table>

**Theories**

- Now we turn to unsat \( (L_2, \ldots, L_n) \)

- A theory consists of:
  - A set of function and predicate symbols (syntax)
  - Definitions for the meaning of these symbols (semantics)
    - Semantic or axiomatic definitions

- Example:
  - Symbols: \( 0, 1, -1, 2, \ldots, \star, \star, \star \) (with the usual meaning)
  - Theory of integers with arithmetic (Presburger arithmetic)

**Decision Procedures for Theories**

- The Decision Problem:
  - Decide whether a formula in a theory + FOL is true

- Example:
  - Decide whether \( \forall x \cdot x \cdot 0 \rightarrow \forall y \cdot \forall x = y + 1 \) in \( \{ x, z, \star \} \)

- A theory is decidable when there is an algorithm that solves the decision problem for the theory
  - This algorithm is the decision procedure for the theory

**Satisfiability Procedures for Theories**

- The Satisfiability Problem
  - Decide whether a conjunction of literals in the theory is satisfiable
  - Factor out the FOL part of the decision problem

- This is what we need to solve in our simple prover

- We will explore a few useful theories and satisfiability procedures for them...

**Examples of Theories, Equality.**

- The theory of equality with uninterpreted functions

- Symbols: \( =, \neq, \preceq \)

- Axiomatically defined:

\[
E = E \\
E_1 = E_2 \\
E_2 = E_3 \\
E_3 = E_4 \\
E_4 = E_5 \\
f(E_5) = f(E_6)
\]

- Example of a satisfiability problem:

\[
g(g(x)) = x \land g(g(g(g(g(x))))) = x \land g(x) \neq x
\]

**A Satisfiability Procedure for Equality**

- Definitions:
  - Let \( R \) be a relation on terms
  - The equivalence closure of \( R \) is the smallest relation that is closed under reflexivity, symmetry and transitivity
    - An equivalence relation
  - Equivalence classes
    - Give a term \( \bar{t} \), we say that \( \bar{t} \) is its representative
    - Two terms \( \bar{t}_1 \) and \( \bar{t}_2 \) are equivalent iff \( \bar{t}_1 = \bar{t}_2 \)
    - Computable in near-linear time (union-find)
  - The congruence closure of a relation is the smallest relation that is closed under equivalence and congruence
A Representation for Symbolic Terms

- We represent terms as DAGs
  - Share common subexpressions
  - E.g. \( f((a, b), b) \)

- Equalities are represented as dotted edges
  - E.g. \( f((a, b), b) = a \)

- We consider the transitive closure of dotted edges

Computing Congruence Closure

- We pick arbitrary representatives for all equivalence classes (nodes connected by dotted edges)

- For all nodes \( t = f(t_1, ..., t_n) \) and \( s = f(s_1, ..., s_n) \)
  - If \( t_i = s_i \) for all \( i \in \{1, n\} \) (find)
  - We add an edge between \( t^* \) and \( s^* \) and pick one of them as the representative for the entire class (union)

Computing Congruence Closure (Cont.)

- Congruence closure is an inference procedure for the theory of equality
  - Always terminates because it does not add nodes

- The hard part is to detect the congruent pairs or terms
  - There are tricks to do this in \( O(n \log n) \)

- We say that \( f(t_1, ..., t_n) \) is represented in the DAG if there is a node \( f(e_1, ..., e_p) \) such that \( e_i^* = t_i^* \)

Satisfiability Procedure for Equality

1. Given \( F \equiv \bigwedge_i t_i = t_i^* \land \bigwedge_j u_j = u_j^* \)
2. Represent all terms in the same DAG
3. Add dotted edges for \( t_i = t_i^* \)
4. Construct the congruence closure of these edges
5. Check that \( \forall j. u_j^* = u_j^* \)

Theorem: \( F \) is satisfiable iff \( \forall j. u_j^* = u_j^* \)

Example with Congruence Closure

- Consider: \( g(g(g(x))) = x \land g(g(g(g(x)))) = x \land g(x) = x \)

Conjugence Closure, Discussion

- The example from before has little to do with program verification
- But equality is still very useful
- The congruence closure algorithm is the basis for many unification-based satisfiability procedures
  - We add the additional axiom:
    \[
    \frac{f(E_1) = f(E_2)}{E_1 = E_2}
    \]
  - Or equivalently
    \[
    \frac{f(E_1) = f(E_2)}{f(E_1) = f(E_2)}
    \]
**Presburger Arithmetic**

- The theory of integers with \( \times, +, \leq, > \)
- The most useful in program verification after equality
- Also useful for program analysis also
- Example of a satisfiability problem:
  \[ y > 2x + 1 \land y > 1 \land y < 0 \]
- Satisfiability of a system of linear inequalities
  - Known to be in \( \mathbb{P} \) (with rational solutions)
  - Some of the algorithms are quite simple
- If we add the requirement that solutions are in \( \mathbb{Z} \) then the problem is \( \mathbb{NP} \)-complete

**Difference Constraints**

- A special case of linear arithmetic
- All constraints of the form:
  \[ x_i - x_j \leq c \quad \text{or} \quad x_i - x_j \geq c \]
- The most common form of constraint
  - Construct a directed graph with:
    - A node for each variable \( x_i \)
    - An edge from \( x_i \) to \( x_j \) of weight \( c \) for each \( x_i - x_j \leq c \)
  \[ x_i \rightarrow_{c} x_j \]

**Difference Constraints**

**Theorem:**

A set of difference constraints is satisfiable if and only if there is no negative weight cycle in the graph.

- Can be solved with Bellman-Ford in \( O(n^2) \)
  - In practice this is typically quite small
  - In practice we use incremental algorithms (to account for assumptions being pushed and popped)
- Algorithm is complete!
- Was used successfully in array-bounds checking elimination and induction variable discovery

**Extensions of Difference Constraints**

- Shostak extended the algorithm to \( ax + by \leq c \)
- Construct a graph as before
  - One node for each variable
  - One undirected edge for each constraint
  - An admissible loop in this graph is a loop in which any two adjacent edges \( "ax + by \leq c" \) and \( "cy + ez \leq f" \)
    - Have \( \text{sgn}(b) = \text{sgn}(c) \)
    - The residue of each adjacent edge is a constraint on \( x \) and \( z \)
      \[ a|x + b|z|c| + |f|b| \]
  - The residue for a loop is an inequality without variables
- **Theorem:** The inequalities are satisfiable if all residues for simple loops are satisfiable

**How Complete are These Procedures?**

- Consider: \( 3x \geq 2y \land 3y \geq 4 \land 3 \geq 2x \)
  \[
  \begin{array}{c|c|c}
  x & \frac{2y}{3} & 0 \\
  \frac{3y}{4} & \frac{2}{2} & \text{Residue is: } 13.5 > 8 \Rightarrow \text{satisfiable} \\
  \end{array}
  \]
  - But only in \( \mathbb{Q} \), not in \( \mathbb{Z} \).
- The unsat procedure is sound: unsat \( \mathbb{Q} \Rightarrow \text{unsat } \mathbb{Z} \)
- But it is incomplete!
- Not a problem in practice
- Or the problem goes away with tricks like this:
  - Transform \( "ax \geq b" \) into \( "x \geq \lfloor b/a \rfloor" \)

**Arithmetic Discussion**

- There are many satisfiability algorithms
  - Even for the general case (e.g. Simplex)
  - Except for difference constraints, all are incomplete in \( \mathbb{Z} \)
  - But \( \mathbb{Z} \) can be handled well with heuristics
- There are no practical satisfiability procedures for \( (\mathbb{Q}, \geq) \) and the satisfiability of \( (\mathbb{Z}, \geq) \) is only semi-decidable
Combining Satisfiability Procedures

- We have developed sat procedures for several theories
  - We considered each theory in part
  - Can we combine several sat procedures?

Consider equality and arithmetic

\[ f(x) - f(y) = f(z) \]
\[ x \leq y \quad y + z \leq x \quad 0 \leq z \]
\[ x = y \]
\[ f(x) = f(y) \]
\[ f(x) - f(y) = z \]
\[ f(x) - f(y) = 0 \]
\[ x = z \]
\[ y = x \]

Yet in any single verification problem we will have literals from several theories:
- Equality, arithmetic, lists, ...

When and how can we combine separate satisfiability procedures?

Nelson-Oppen Method (1)

1. Represent all conjuncts in the same DAG

\[ f(x) - f(y) = f(z) \land y \geq x \land x \geq y \land z \geq 0 \]

Nelson-Oppen Method (2)

2. Run each sat procedure

- Require it to report all contradictions (as usual)
- Also require it to report all equalities between nodes

Nelson-Oppen Method (3)

3. Broadcast all discovered equalities and re-run sat procedures

- Until no more equalities are discovered or a contradiction arises

Puzzle: Constructive vs. Classical Proofs

- Prove the following fact:

\[ \exists x \in R \setminus Q. \quad x^{\sqrt{2}} \in Q \]

- Hint: Try \( \sqrt{3}^{\sqrt{2}} \)