Nelson-Oppen review

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

\[ \vdash \vdash \vdash \]

\[ \models \models \models \]

\[ QED \]

Drawback of Nelson-Oppen

• Theory must be convex, otherwise must backtrack
• Some large overheads:
  – Each decision procedure must maintain its own equalities
  – There are a quadratic number of equalities that can be propagated

Shostak’s approach

• Alternate approach to combining theories that addresses some of the performance drawbacks of Nelson-Oppen
  – Published in 1984 in JACM, but the original formulation was later found to be flawed in several ways
  – Long line of work to correct these mistakes
  – Culminating in “Deconstructing Shostak” by Ruess and Shankar, which gives sound and complete version of Shostak
• Unpublished manuscript by Crocker from 1988 showing that Shostak is 10 times faster than Nelson-Oppen
• Recent paper by Barrett, Dill and Stump in 2002 shows that Shostak can be seen as a special case of Nelson-Oppen

Shostak’s approach

• Shostak is used in a variety of theorem provers, including PVS and SVC
• We will cover the intuition behind Shostak’s approach, but we won’t see the details
The key idea in Shostak

- Keep one congruence closure data-structure $S$ for all theories
- Each individual decision procedure finds new equalities based on the ones that are already in $S$
- As individual decision procedures find equalities, add them to $S$

Adding equalities to $S$

- Straightforward to encode equalities over uninterpreted function symbols in $S$
  - Since $S$ is a congruence-closure data structure, since congruence closure was originally intended for exactly these kinds of equalities
- Interpreted functions symbols require more care
  - For example, an equality $y + 1 = x + 2$ cannot be processed by simply putting $y + 1$ and $x + 2$ in the same equivalence class, since the original equality in fact entails a multitude of equalities, such as $y = x + 1$, $y - 1 = x$, $y - 2 = x - 1$, etc.

Impose two restrictions

1. Theories must be solvable: any set of equalities in the theory must have an equivalent solved form
   - Equalities are in solved form if the left hand side of the equalities are only variables and the right-hand sides are expressions that don’t reference any of the left-hand side variables
   - Will use this to substitute solved variables in all terms

Impose two restrictions

1. Theories must be solvable: any set of equalities in the theory must have an equivalent solved form
   - Equalities are in solved form if the left hand side of the equalities are only variables and the right-hand sides are expressions that don’t reference any of the left-hand side variables
   - Will use this to substitute solved variables in all terms
   - $x + y = z + 3$
   - $x - y = 3z + 1$
     - $y = 2x + x$
     - $y = 3 + x$

Impose two restrictions

2. Theory must be canonizable
   - There is a canonizer function $\sigma$ such that if $a = b$, then $\sigma(a)$ is syntactically equal to $\sigma(b)$
   - Canonizer for linear arithmetic: transform terms into ordered monomials
     - $\sigma(a + 3c + 4b + 3 + 2a + 4) = 3a + 4b + 3c + 7$
     - The intuition is that by canonizing all terms, we can then use syntactic equality to determine semantic equality
Putting it all together

\[ f(x - 1) - 1 = x + 1 \land f(y) + 1 = y - 1 \land y + 1 = x \]
\[ \begin{align*}
\{(x - 1) &= x + 2 \Rightarrow f(y) = y + 3 \\
\{y &= y - 2 \Rightarrow y - y = y + 3 \Rightarrow d = 5 \\
x &= y + 1
\end{align*} \]

ACL2 decision procedures

- ACL2 architecture
  - Given a goal, ACL2 has a set of strategies it can apply
  - For example: rewriting, simplification, induction
  - Applying a strategy produces sub-goals from the given goal
  - Each sub-goal needs to be proven recursively

Adding linear arithmetic

- First attempt was to just use the decision procedures directly as a strategy
- Not found to be useful, because it was rarely the case that the goal would reduce to TRUE using linear arithmetic
- Rather, they found they needed to add linear arithmetic in the rewrite system
- A rewrite rule: \( A \Rightarrow T_1 = T_2 \)
  - To trigger, need to establish \( A \)
  - They often needed linear arithmetic to establish \( A \)

Decision procedures summary

- Communication between decision procedures
  - Nelson-Oppen (simplify), Shostak (PVS, SVC)
- Communication from heuristic prover to decision procedures
  - assert formulas (most theorem provers)
- Communication from decision procedures to heuristic theorem prover
  - yes/no answers (all theorem provers)
  - terms to use for matching (Simplify, ACL2)
  - proofs to prune search space (Verifun)

Keep a linear arith DB

- A rewrite rule: \( A \Rightarrow T_1 = T_2 \)
- To establish \( A \), add \( \neg A \) to the current database of linear equalities and inequalities
- If an inconsistency is reached, we know \( A \) holds
  - We can perform the rewrite
  - Remove \( \neg A \) from the database, and add \( A \)
- As in Simplify, the arith DB is used for matching heuristic to instantiate quantifiers
So far

- Proof-system search ($\vdash \vdash \vdash \vdash$)
- Interpretation search ($\models \models \models \models$)
- Equality
- Induction
- Quantifiers

Cross-cutting aspects
- Decision procedures

Main search strategy
- Comms between decision procedures and between dec and G

• DPLL
• Backtracking
• Incremental SAT

Link between $\models$ and $\vdash$

- Soundness: $\Gamma \vdash \phi \implies \Gamma \models \phi$
- Completeness: $\Gamma \models \phi \implies \Gamma \vdash \phi$

- Virtually all inference systems are sound
- Therefore, to establish $\Gamma \models \phi$, all one needs to do is find a derivation of $\Gamma \vdash \phi$

Goal: find a proof

- Need two things:
  - A proof system
  - A search strategy
- These two are heavily intertwined
- Let’s start by looking at some proof systems

Hilbert-style systems

- Many axioms and usually just one inference rule, modus ponens

<table>
<thead>
<tr>
<th>Axiom (schemas)</th>
<th>Inference rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. $X \Rightarrow (Y \Rightarrow X)$</td>
<td>$A \Rightarrow B$</td>
</tr>
<tr>
<td>2. $(X \Rightarrow (Y \Rightarrow Z)) \Rightarrow ((X \Rightarrow Y) \Rightarrow (X \Rightarrow Z))$</td>
<td>$\Rightarrow$</td>
</tr>
<tr>
<td>3. $F \Rightarrow X$</td>
<td>$A \Rightarrow B$</td>
</tr>
<tr>
<td>4. $X \Rightarrow T$</td>
<td>$\Rightarrow$</td>
</tr>
<tr>
<td>5. $\sim X \Rightarrow X$</td>
<td></td>
</tr>
<tr>
<td>6. $X \Rightarrow (\sim X \Rightarrow Y)$</td>
<td>$B$</td>
</tr>
</tbody>
</table>

Coming up with a complete set of axiom schemas is not trivial

Example proof

1. $X \Rightarrow (Y \Rightarrow X)$
2. $(X \Rightarrow (Y \Rightarrow Z)) \Rightarrow ((X \Rightarrow Y) \Rightarrow (X \Rightarrow Z))$
3. $F \Rightarrow X$
4. $X \Rightarrow T$
5. $\sim X \Rightarrow X$
6. $X \Rightarrow (\sim X \Rightarrow Y)$

• Show: $P \Rightarrow P$
Example proof

- Show: $P \Rightarrow P$
  - Instantiate 2 with X being $P$, Y being $P$, and Z being $P$:
    - (P ⇒ ((P ⇒ P) ⇒ P)) ⇒ ((P ⇒ (P ⇒ P)) ⇒ (P ⇒ P))
  - Instantiate 1, taking X to be P and Y to be P:
    - P ⇒ (P ⇒ P)
  - Instantiate 1 with X and Y to be P:
    - P ⇒ P

Hilbert-style systems

- Does not mimic the way humans do proofs
- To prove $A \Rightarrow B$ in a Hilbert-style system, must find the right way instantiate axioms and then apply MP to get $A \Rightarrow B$
- How does a human prove $A \Rightarrow B$?

Natural deduction

- The system of natural deduction was developed by Gentzen in 1935 out of dissatisfaction with Hilbert-style axiomatic systems, which did not closely mirror the way humans usually perform proofs
- Gentzen wanted to create a system that mimics the “natural” way in which humans think

Example proof

- Show: $P \Rightarrow P$
  - Instantiate 2 with X being $P$, Y being $P$, and Z being $P$:
    - (P ⇒ ((P ⇒ P) ⇒ P)) ⇒ ((P ⇒ (P ⇒ P)) ⇒ (P ⇒ P))
  - Instantiate 1, taking X to be P and Y to be P:
    - P ⇒ (P ⇒ P)
  - Instantiate 1 with X and Y to be P:
    - P ⇒ P

Hilbert-style systems

- Does not mimic the way humans do proofs
- To prove $A \Rightarrow B$ in a Hilbert-style system, must find the right way instantiate axioms and then apply MP to get $A \Rightarrow B$
- How does a human prove $A \Rightarrow B$?
- Assume A, and show B
- In this context, showing $P \Rightarrow P$ is very easy

Natural deduction rule for $A \Rightarrow B$

$$\Gamma, A \vdash B$$

$$\Gamma \vdash A \Rightarrow B$$
Natural deduction rule for $A \Rightarrow B$

- This is called an introduction rule, since it introduces the $\Rightarrow$ connective

Natural deduction

- Each connective also has an elimination rule
Natural deduction

\[ \Gamma, A \vdash F \]
\[ \Gamma \vdash A \]
\[ \Gamma \vdash \neg A \quad \neg I \]
\[ \Gamma \vdash \neg \neg A \quad \neg \neg \neg I \]
\[ \Gamma \vdash A \quad \neg \neg E \]
\[ \Gamma \vdash F \quad F I \]
\[ \Gamma \vdash A \quad F E \]
\[ \Gamma \vdash T \quad T I \]

Note: one can get rid of the FE without losing expressiveness. Can someone see why?

Once we have a proof system

• Once we have a proof system, the goal is to devise a search algorithm to find a proof
• Search algorithm sound: proofs that it finds are correct
• Search algorithm complete: if there is a proof, the algorithm will find it
• These soundness and completeness properties relate the search algorithm to the proof system, and should not be confused with soundness and completeness of the proof system

Two main strategies

• Given a formula to prove:
  – One can start from axioms and apply inference rules forward, until a derivation of the given formula is found
  – One can start from the formula to prove (the goal) and apply inference rules backward to find sub-goals until all sub-goals are axioms

• The forward version is sometimes called forward chaining, the backward version backward chaining

Forward search

• Keep a knowledge base, which is the set of formulas that have been proved so far
• Given goal to prove:
  – Start with empty knowledge base
  – While goal not in knowledge base:
    • Instantiate an axiom or an inference rule to deduce a new formula
    • Add the new formula to the knowledge base
    • If the goal is in the knowledge base, return VALID
  • No need to backtrack

Forward search -- refutation

• Start with knowledge base being the negation of the goal
• While enlarging the knowledge base, if \( F \) becomes part of the knowledge, then return VALID

Backward search

• Given goal to prove:
  – If the goal is \( T \) then return VALID
  – Otherwise:
    • Let \( S \) be the set of inference rules that can be applied backward
    • Pick some subset \( S' \) of \( S \) that we want to consider
    • For each inference rule in \( S' \):
      – Apply the inference rule backward on the goal to produce \( n \) sub-goals (axioms produce sub-goals of \( T \))
      – Run the search recursively on each sub-goal
    • If all recursive calls return VALID, return VALID
    • Return INVALID

Note: This is a depth-first search. Can have other search orders, like breadth first, iterative deepening
Proofs

• One can easily adapt these algorithms to keep track of the proof tree, so that a proof can be produced if the goal is valid
• Contrast this with our backtracking search in the semantic domain, where generating a proof is not as simple
• On the other hand, what about when the proof fails?
  – much easier to get counter-example in interpretation search than in proof-system search

Non-determinism

• Whatever the direction of the search, one of the biggest problems is that there are a lot of choices to make. This is called non-determinism.
  – There may be many inference rules that are applicable
  – Even for one rule, there may be multiple instantiations
  – For example, applying \( \lor \) E backward requires one to determine A and B

\[
\Gamma, A \vdash C \\
\Gamma, B \vdash C \\
\Gamma \vdash A \lor B \\
\Gamma \vdash C \quad \text{\( \lor \)E}
\]

Two kinds of non-determinism

• Don’t care non-determinism (also called conjunctive non-determinism)
  – All choices will lead to a successful search, so we “don’t care” which one we take
  – Only consideration for making the choice is efficiency
• Don’t know non-determinism (also called disjunctive non-determinism)
  – Some of the choices will lead to a successful search, but we “don’t know” which one a priori
  – In order to deal with this kind of non-determinism, try all choices using some traversal order (depth-first, breadth-first, iterative deepening)

Next lecture

• We’ll see how to reduce non-determinism
• We’ll learn about tactics and tacticals, one of the important techniques used in proof system searches
• We’ll learn about some proof systems that are more suited for automated reasoning, like the sequent calculus and resolution