Clause Selection in Reolution-style Theorem Provers

Robert Veroff
Department of Computer Science
University of New Mexico
www.cs.unm.edu/~veroff

AITP 2016
April 3–7, Obergurgl, Austria

Objectives of Automated Theorem Proving (ATP)

- Mechanizing mathematics
  - repositories for accumulated knowledge
  - enforce correctness and rigor
- Automatic theorem proving
  - plug and chug
  - consistently and reliably prove "easy" problems easily
- Automated reasoning assistant
  - mathematically challenging problems
  - "interesting" (to someone)
  - new knowledge (for example, open questions)
- Real-world applications

One Approach

Problem representation

- first order clauses
- posed for proof by contradiction

Given an initial set C of clauses and a set of inference rules, find a derivation of the $empty\ clause$ (for example, by the resolution of two conflicting clauses `P` and `-P`).

Basic loop

```
while (no proof found)
{
   select clauses
   apply inference rules to selected clauses
   process inferred clauses
}
```

A common variation is to postpone some processing of inferred clauses until they are selected ("Otter Loop" vs. "Discount Loop").

Clause Selection Basics

Selection mechanisms

- symbol count (weighting)
- user-defined weighting patterns
- attribute-based restrictions (e.g., set of support)
- model-based selection (semantic guidance)
- subsumption-based selection (hints)

User-defined rules or scripts can be used to specify combinations of these mechanisms.

Weighting

- Analogous to a heuristic function (ordered search)
- Symbol count is a good starting point
- User-defined templates can be used to promote, avoid or focus attention on certain patterns. For example,

  weight(nand(x_1,x_2)) = weight(x_1) + weight(x_2) - 5

  weight(sqrt(sqrt(sqrt(x_1)))) = 9999

  weight(EL(x_1,x_2)) = weight(x_1) + 2 * weight(x_2) + 1

  where the x_i match any terms.

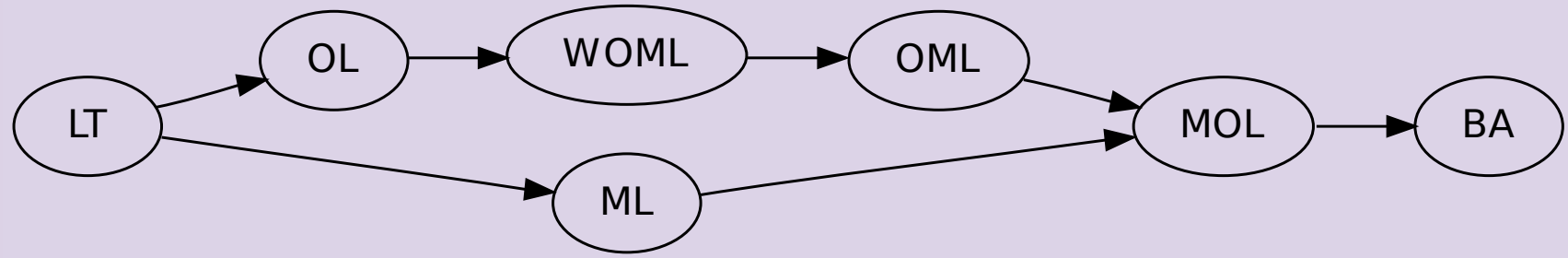
Set of Support

- Intuition: To prove a theorem t in theory A, focus the search on t rather than on deriving the general theory A.
- Idea: Restrict the application of inference rules to require that at least one "parent" have in its derivation history a clause from the representation of t.
- The initial clauses representing t and their descendants are the *set of support*.
- In general, the user can specify the initial set of support.

There are other attribute-based restriction strategies (for example, based on literal orderings).

Theory Hierarchies

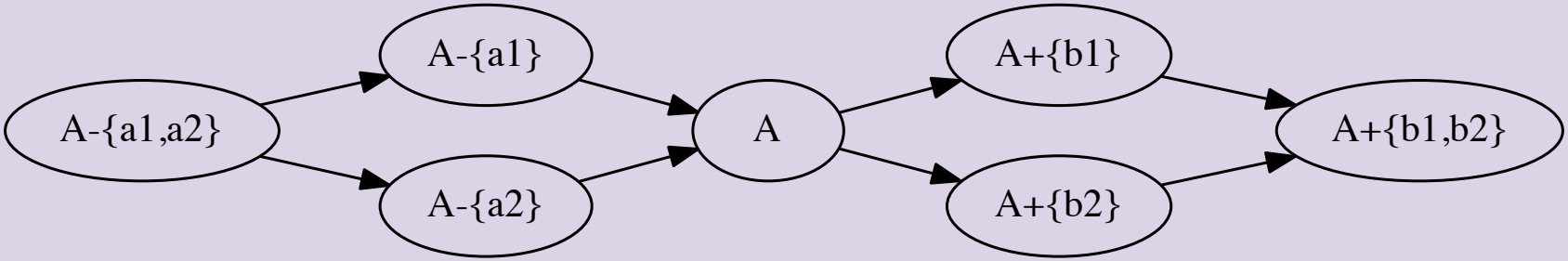
Example: Lattice Theory



$\mathcal{LT}$

+ Invertibility and Compatibility ($\mathcal{OL}$)
+ Weak Orthomodularity ($\mathcal{WOML}$)
+ Orthomodularity ($\mathcal{OML}$)
+ Modularity ($\mathcal{MOL}$)
+ Distributivity ($\mathcal{BA}$)

Theory Hierarchies



What extensions to a theory suffice to yield a proof of theorem t?

How far up the hierarchy can we push theorem t?

Can knowledge about t in weakenings or extensions of A help us find a proof of t in A?

Semantic Guidance

- Say $A \Rightarrow c$ is a theorem but $A - \{a\} \Rightarrow c$ is *not* a theorem.
- Let I be an interpretation (model) that satisfies $A - \{a\}$ and falsifies c.
- Key observation: In order to infer a clause that evaluates to **False** under I, at least one parent p of the inference must evaluate to **False** under I.

  Similarly for the parents of p, and so on ...

- It follows that a proof of c from A will necessarily include steps that evaluate to **False** under I.
- Idea: Have some selection bias for clauses that evaluate to **False** under I.
- The challenge is to find weakenings of A yielding good candidate interpretations I.

Proof Sketches

Consider a derivation as a sequence of clauses,

$$c_1, c_2, \ldots, c_i, \ldots, c_j, \ldots, c_n$$

where

- c_i is an extra assumption for the target theory A
- derived clause c_j has c_i in its derivation history

c_j either is derivable from A or it is not.

- if yes, it suffices to find a new derivation of c_j
- if no, it suffices to "bridge the gaps" to the consequences of c_j

In either case, we have a partial proof that *might* be easier to complete than finding a proof from scratch.

The Proof Sketches Method

- Idea: Collect proofs of the target theorem in extended theories (i.e., with extra assumptions) and have a selection bias for clauses that match clauses in these proofs.
- The emphasis is on the *sufficiency* of the collected "proof sketches". This does not preclude finding a different proof.
- Move up the hierarchy by systematically generating new proof sketches with fewer extra assumptions, including all previous proof sketches for guidance.
- The challenge is to find effective extensions of the target theory (extra assumptions).

Prover9 supports proof sketches via *hints*.

Hints

As in approaches bases on analogy, proof planning, and abstraction, the point is to have high-level control of the proof search based on some notion of mapping and/or matching.

- Idea: subsumption as part of a search strategy
  - input list of *hint* clauses
  - hint subsumers are notable milestones
  - bias search (clause selection) accordingly
- Uses
  - proof sketches
  - analogy (related problems in a large study)
  - proof checking and completion (vs. divide an conquer)
  - proof transformation

Example (ring theory): $(\forall x(x^2 = x)) \Rightarrow (\forall x, y(x * y = y * x))$

Ring axioms:

```
0 + x = x.                          % L_Ident_+
-x + x = 0.                         % L_Inv_+
x + y = y + x.                      % Comm_+
(x + y) + z = x + (y + z).          % Assoc_+
(x * y) * z = x * (y * z).          % Assoc_*
x * (y + z) = (x * y) + (x * z).    % L_Dist_(*/+)
(y + z) * x = (y * x) + (z * x).    % R_Dist_(*/+)
```

Theorem hypothesis: $\forall x(x^2 = x)$

```
x * x = x.
```

Negation of the conclusion: $\exists x, y(x * y \neq y * x)$

```
a * b != b * a.
```

Example (continued): How Hints Can Affect a Search

- Run with the original formulation of the problem.
  - proof found after 89 given clauses
- Extract the proof, including the intermediate rewrite steps.
- Rerun, including the already found proof as hints.
  - proof found after 54 given clauses

The prover "recognized" matched (subsumed) hints as possibly significant milestones and adjusted its search accordingly.

The hints are *not assumed* but provide guidance without constraining the search.

Of course, this is *not* where hints come from when searching for new proofs!

See `www.cs.unm.edu/~veroff/AITP16/`.

Subtleties

What to do when an already matched hint is matched again?

> Intuitively, the first matcher of a hint h_1 is more important than the second matcher of a hint h_2.

What to do when something back subsumes a hint matcher?

> Inutitively, it depends on whether the subsumed hint matcher has already been selected and used.

These issues become relevant when working with very large sets of hints.

Taking Full Advantage

Clause processing is a significant part of search strategy,

For example, which direction should

$$x * (y + z) = x * y + x * z$$

be used as a rewrite rule, if at all?

Say we have a candidate proof sketch in the form of a sequence of lemmas, but these lemmas are most easily proved under different search strategies.

We could treat each of the lemmas as an independent subproblem, but it might be better to use the sketch for guidance rather than limiting ourselves to this particular proof.

Idea: Make multiple runs under different strategies, assuming previous hint matchers as input lemmas in later runs.

Current Status

The Good,

the Bad

and the Ugly ...

The Good

- Numerous results, including solutions to several open questions in mathematics and logic
- Proofs substantially longer than anything we were able to find previously
- For example, in one project,
  - results of mathematical interest with $>$ 20K steps
  - intermediate results with $>$ 70K steps

The Bad

- Very large sets of accumulated hints
- For example, in that same project,
  - accumulated $>$ 200K hints from results of mathematical interest
  - many thousands more from intermediate proof sketches
- Prohibitive affect on performance

We are starting to look at machine learning and other methods from AI to manage and prune the hints for a given problem.

The Ugly

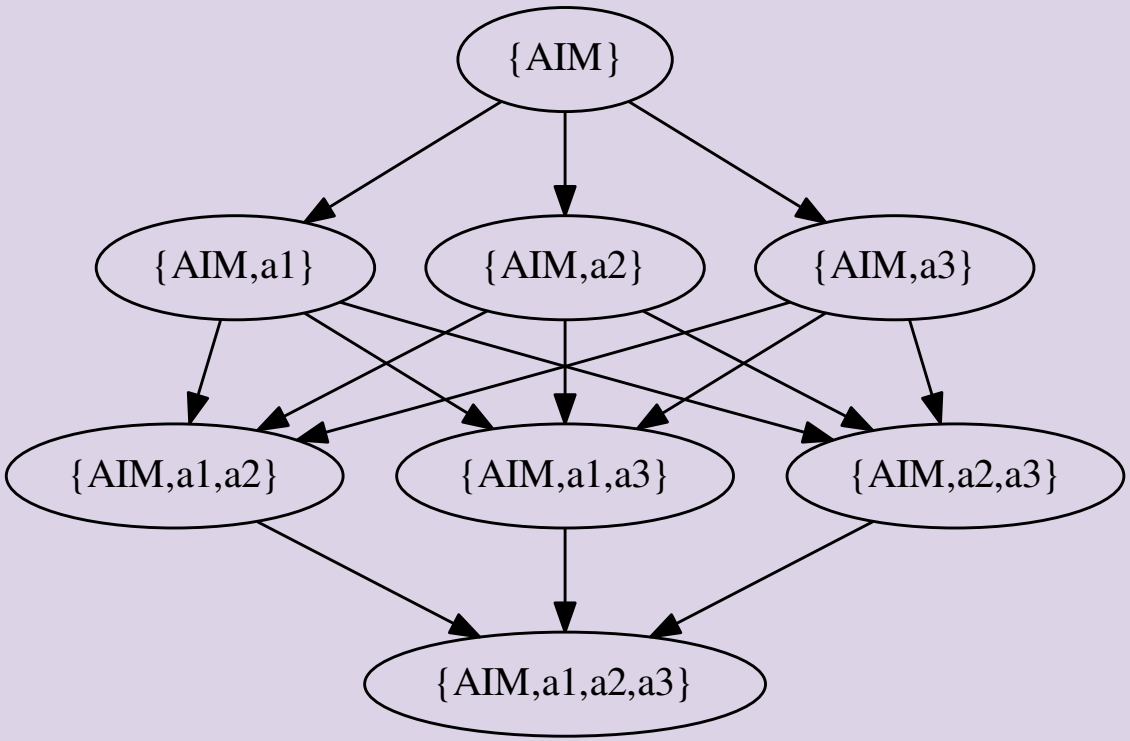
- Inconsistent signatures
  - different symbols (e.g., $\vee$ vs. $+$ in Boolean algebra)
  - different axiom systems (e.g., $\{\wedge, \vee, \neg\}$ vs. $|$ in Boolean algebra)
- Can we identify (and transform) automatically?

We may be able to borrow ideas from information theory to help deal with this.

The AIM Problem

- Concerns Abelian inner mappings in loop theory
- Proposed as an automated deduction challenge problem by Michael Kinyon at ADAM 2009
- Collaboration with Petr Vojtěchovský, J. D. Phillips and Aleš Drápal
- Several candidate extensions; some combinations of special interest
- Working our way up the hierarchy

The AIM Hierarchy



Several extensions of mathematical (not just strategic) interest.

References

- J. McCharen, R. Overbeek and L. Wos. Complexity and Related Enhancements for Automated Theorem-proving Programs. *Computers and Mathematics with Applications* **2**:1–16, 1976.
- L. Wos, G. Robinson and D. Carson. Efficiency and Completeness of the Set of Support Strategy in Theorem Proving. *J. of the ACM* **12**:536–41, 1965.
- W. McCune. Prover9. `www.cs.unm.edu/~mccune/prover9/`.
- R. Veroff. Using Hints to Increase the Effectiveness of an Automated Reasoning Program: Case Studies. *J. Automated Reasoning* **16**(3):223–239, 1996.
- R. Veroff. Solving Open Questions and Other Challenge Problems Using Proof Sketches. *J. Automated Reasoning* **27**(2):157–174, 2001.