PATAP

Premise selection for Automated Theorem proving using Alternating Paths

Jannis Gehring, Stephan Schulz

DHBW Stuttgart

2025-08-01

Outline

- Premise selection in Automated Theorem Proving
- The Alternating Path method
- Implementation
- Evaluation
- Conclusion
- Future Work

Premise selection

Why premise selection?

- Large axiomatisations
- Few axioms are relevant for any given proof problem
- Problem: Lots of irrelevant premises hurt prover performance
- Solution: Restricting proof search to “relevant” axioms

Premise selection - related work

MoPe

- Conjecture is relevant
- Similarity of symbol frequency vectors determines relevance
- Iterative expansion

SInE

- Symbols in conjectures are relevant
- Formulas “defining” relevant symbols become relevant, and so do their other symbols
- Iterative expansion
- Good performance in past CASC’s

Premise selection - related work

Various ML-based methods

- Good performance in specialized situations
- Requires training for each application domain

Alternating paths for premise selection

Idea of PATAP - Abstract Idea

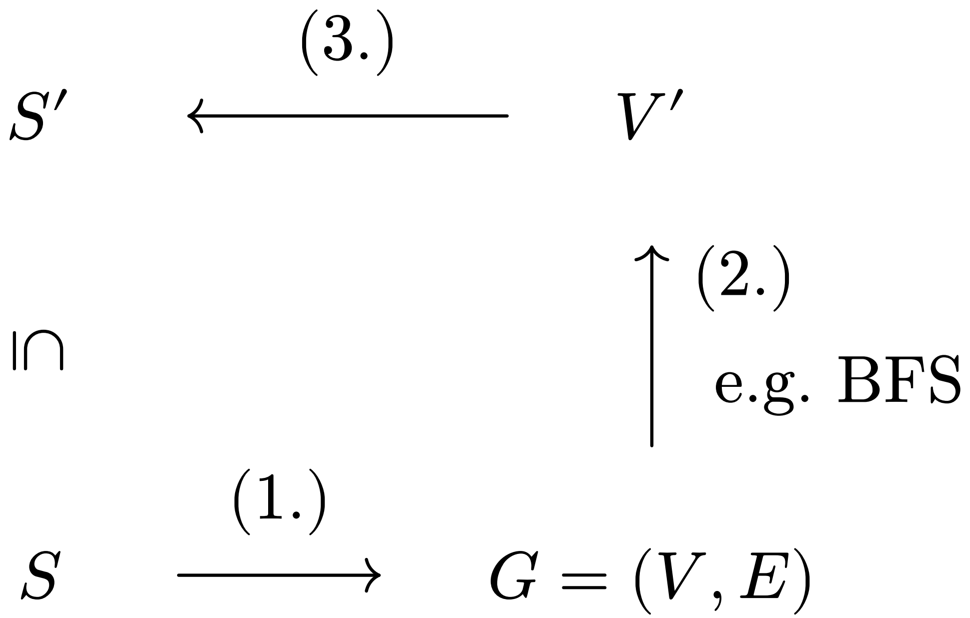
- Per set-of-support: if there is a proof, there is a set-of-support proof
- Identify and select clauses that can participate in such a proof
- Additionally: Limit number of clauses, prefer *closer* ones

Idea of PATAP - Technical Description

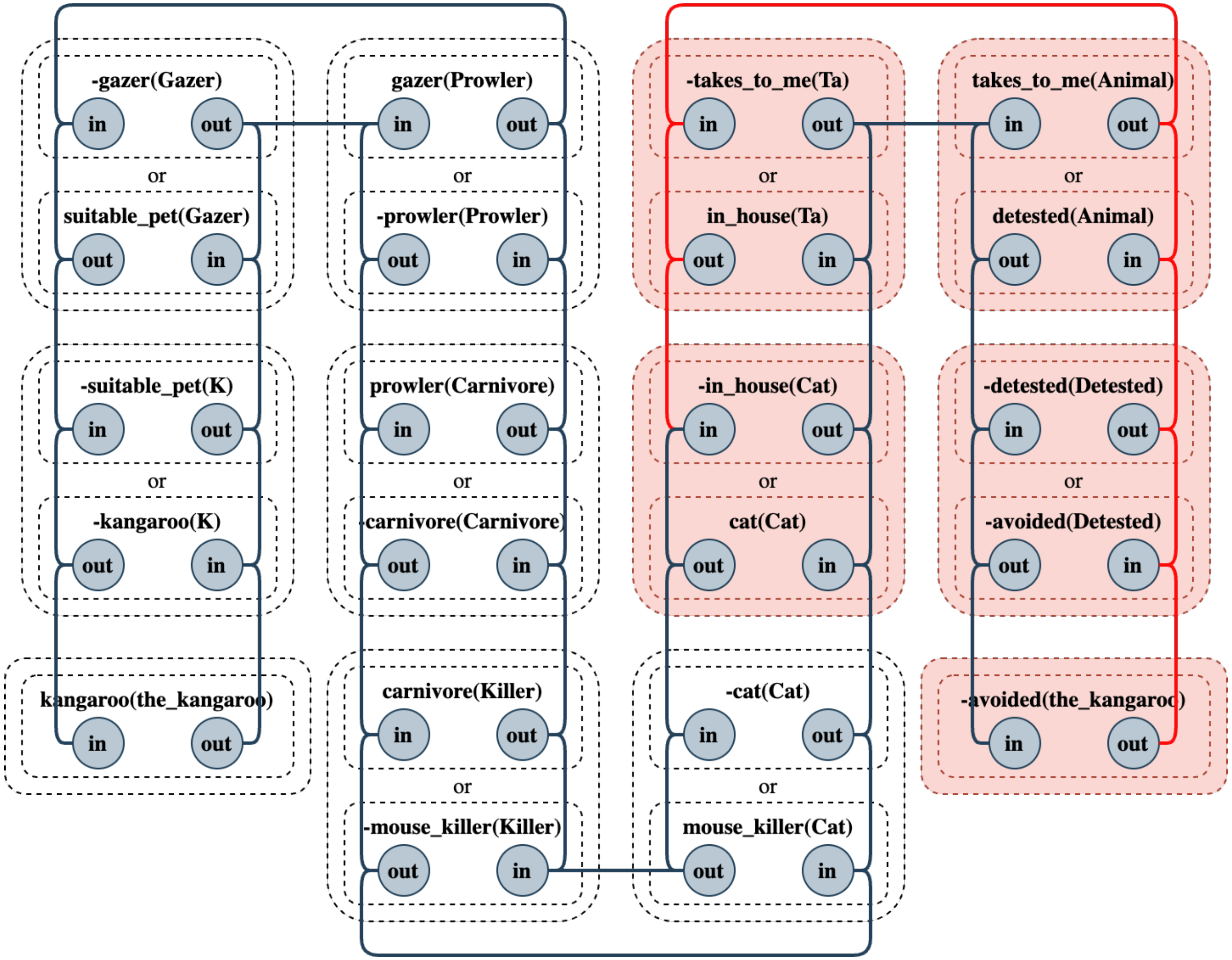
1. Construct Graph G from set of clauses S
   - Node for every clause c, literal $l \in c$ and direction $d \in \{\text{in}, \text{out}\}$
   - Edges: switching l in c or unifiability of two clauses
2. Compute neighbourhood V' of conjectures' nodes
3. Convert V' back to clauses S'
4. Test satisfiability of S' (e.g. resolution)

- S' unsatisfiable $\rightarrow$ S unsatisfiable
- (S' satisfiable $\rightarrow$ S unknown)

Idea of PATAP - Technical Description



Example demonstration



Implementation

Implementation

- Integrated into PyRes
  - FOL prover implementing binary resolution
  - Readability first
- Requirement: class that implements:
  - `def construct_graph(clause_set)`
  - `def get_rel_neighbourhood(from_clauses, distance: int)`
- Different approaches to data structure for edges
  1. Universal set
  2. Adjacency set
  3. Adjacency matrix

Evaluation

Evaluation

SZS statuses	ResourceOut	Unsat.	Theorem	Satisfiable	CtrSat.	Unknown	GaveUp	Inappropriate	Σ
ResourceOut	969	9	8	0	0	0	263	0	1249
Unsatisfiable	10	510	0	0	0	0	102	0	622
Theorem	11	0	385	0	0	0	2	0	398
Satisfiable	0	0	0	50	0	0	28	0	78
CounterSatisfiable	1	0	0	0	59	0	3	0	63
Unknown	44	0	0	0	0	0	12	0	56
GaveUp	0	0	0	0	0	0	0	0	0
Inappropriate	0	0	0	0	0	0	0	0	0
Σ	1035	519	393	50	59	0	410	0	2466

Evaluation

Usefulness

😐

- Some new problems could be solved, some were lost

Efficiency

😐

- Most efficient: Implementing edges through adjacency sets
- Current bottleneck: Constructing unification edges

Evaluation

Usability

- Only one parameter with a clear domain
- Combinable with other methods

Maintainability

- Concepts rather simple (graph construction, BFS)
- Little to no dependencies to third-party libraries

Conclusion

Conclusion

- Experimental results so far are inconclusive
- Current implementation is too slow for large problems (where we would expect most benefits)
- Bottleneck: Find complementary unifiable atoms

Future Work

1. Evaluate selection quality and cost independently
   - Select clauses without time limit
   - Use high-performance prover with uniform time limit for proof search
2. Optimized graph construction
   - Indexing for unification edges
3. Search for optimal relevance distance, automatic suggestion

Thanks!