Learning Theorem Proving by Example — Implementing JavaRes

Stephan Schulz
DHBW Stuttgart
schulz@eprover.org

Talk for ThEdu
July 11, 2021

Adam Pease
Articulate Software
apease@articulatesoftware.com

Overview

- Motivation
- Algorithms
- Architecture and Data Structures
- Conclusion

What is this?

- We know some things about the world
- Encode that knowledge in logic
- Run a theorem prover to answer questions

Why First Order Logic

- Anything less than that leaves too much knowledge implicit and not accessible to computation
  - Explanation, validation, GDPR etc
- In one study of ~7500 sentences from Brown Corpus and COCA, ~45% required at least FOL
- Sound – can’t derive something false from true premises
- Complete – will find an answer at least in infinite time
  - But FOL isn’t decidable – not guaranteed to terminate in finite time – oh well….
  - but a typical case with modern provers is answers in under a second, even with large theories

Why Automated Theorem Proving?

- Definitions fix shared meaning of terms
- Large numbers of definitions are impractical to check by hand for consistency – “dictionary for computers to read”
- Question answering with explanation of deductions is potentially very powerful
- I have a big theory – Suggested Upper Merged Ontology (SUMO)– and I want to do practical computation with it

Why Another Prover?

- Understanding ATP is hard (at least it was for me)
- Barrier to entry is high – papers and textbooks on ATP have lots of math, and not a lot on data structures and architecture
- Maybe if more people understood the power of FOL ATP, more would use it – create a simple example FOL ATP for education

Process

- Stephan Schulz is the developer of Eprover
  - One of the top performing FOL provers in the yearly CASC competitions for decades
- Stephan wrote PyRes – FOL ATP in Python while I “shadowed” him, asking for explanations and writing the same algorithms in Java – JavaRes
- Goal – explain ATP from a programmer’s perspective, not a logician’s

What was hard?

- One of the CNF algorithms
  - Need examples for each step
  - Distinction between literals and clauses makes things more complicated
- Integration testing
  - Needed to develop a small and fast set of problems
- Heuristics and optimizations
  - Needed more examples
- Result – lots more unit and integration tests

The Core Algorithms

Resolution Theorem Proving

- a | ~b | ~c
- ~a | d

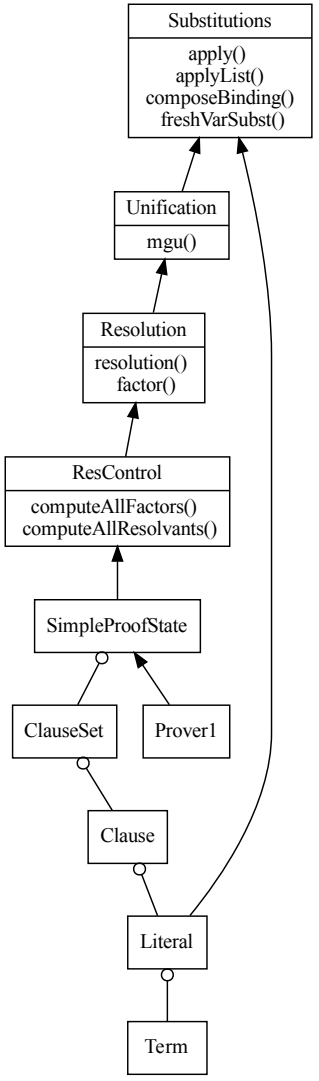
- ~b | ~c | d

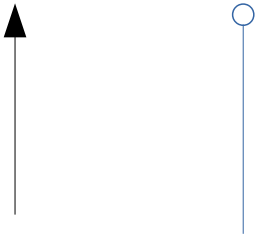
“a or not b or not c” is true

Clausification

- FOL has and, or, not, exists, forall, implies, iff, equals
- Conjunctive Normal Form (CNF) has and, or, not, equals
  - Relatively flat structure – a set of clauses, where a clause is a set of literals
- Simpler to write a prover with fewer operators
- “classical” algorithm described in Russel&Norvig AIMA
  - And also SmallCNF from (Nonnengart&Weidenbach, 2001)

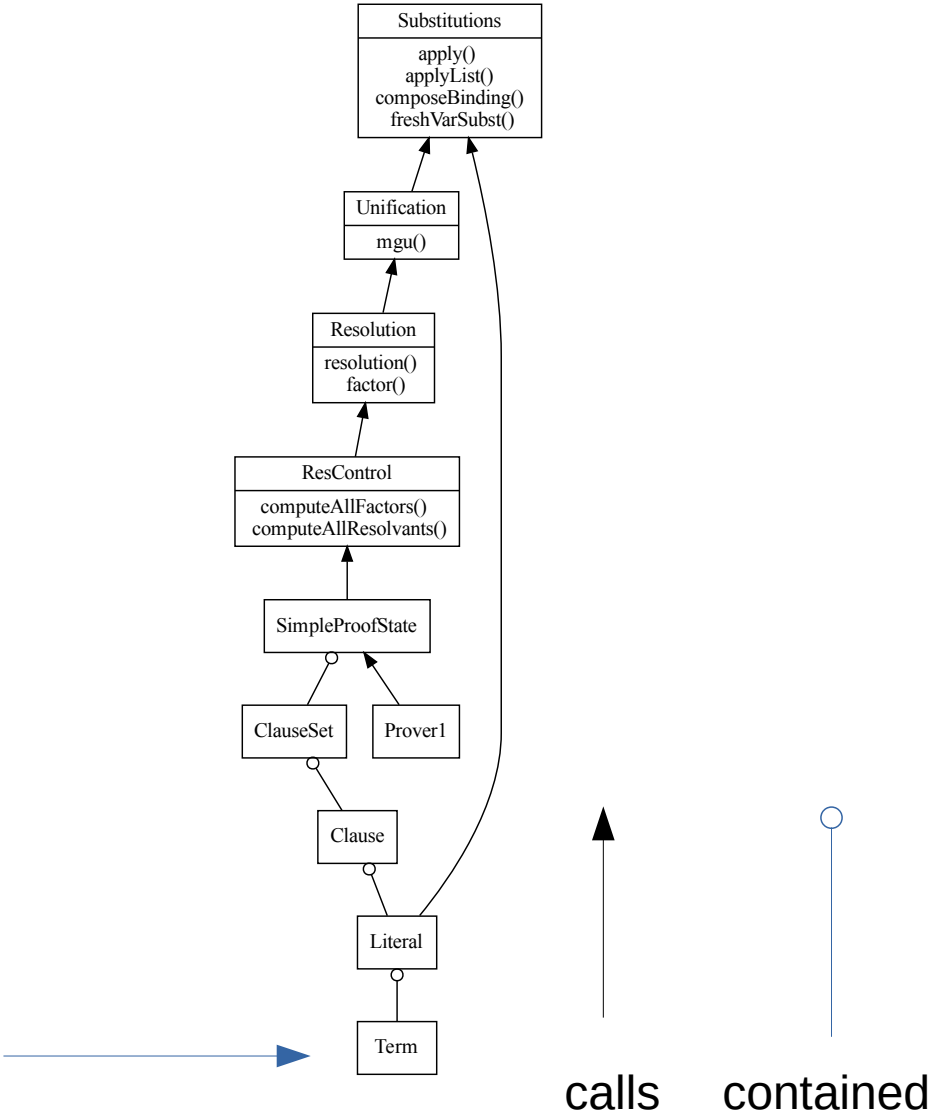
Simple Architecture





Term

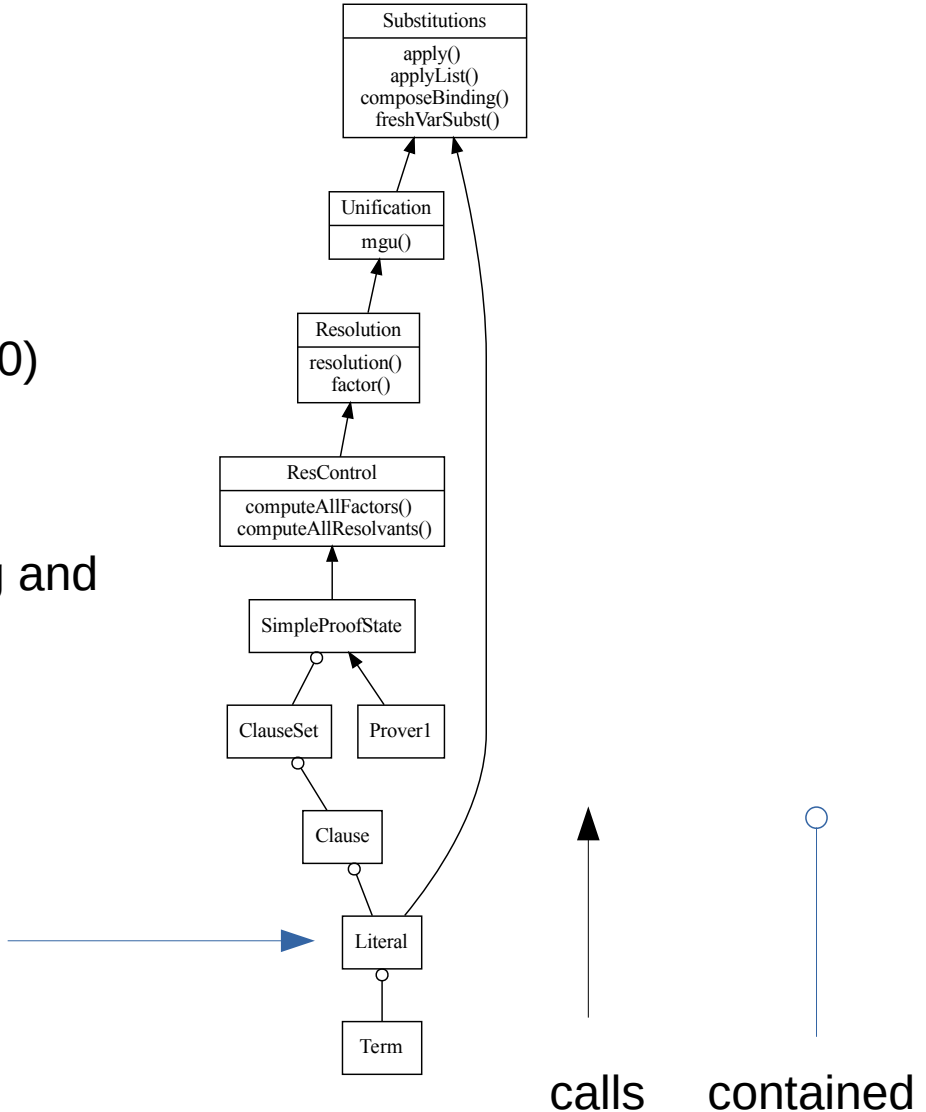
a – a constant
?A – a variable
governmentFn(unitedStates) – a functional term



Literal

instance(koko, gorilla)
memberSize(governmentFn(unitedStates),1000000)
~likes(mary,bill)

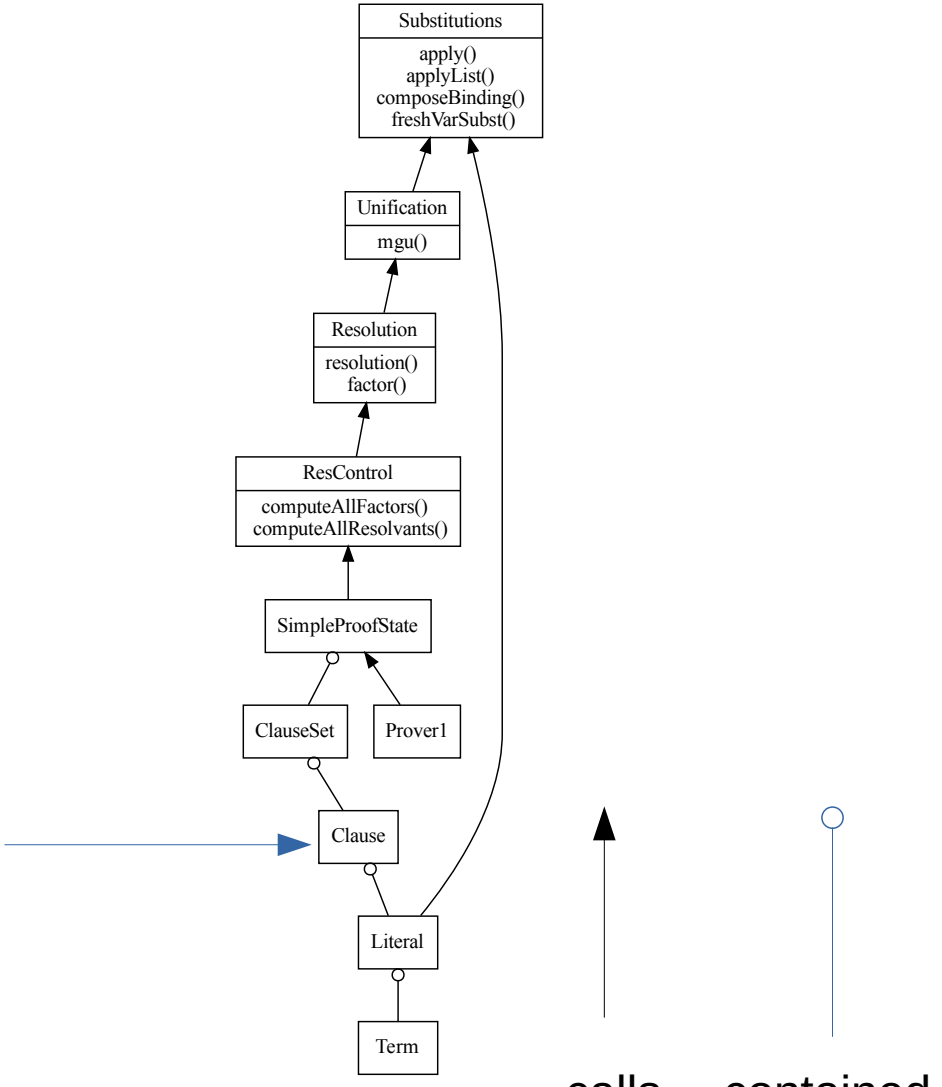
Literals have truth values, terms denote something and don’t have a truth value



Clause

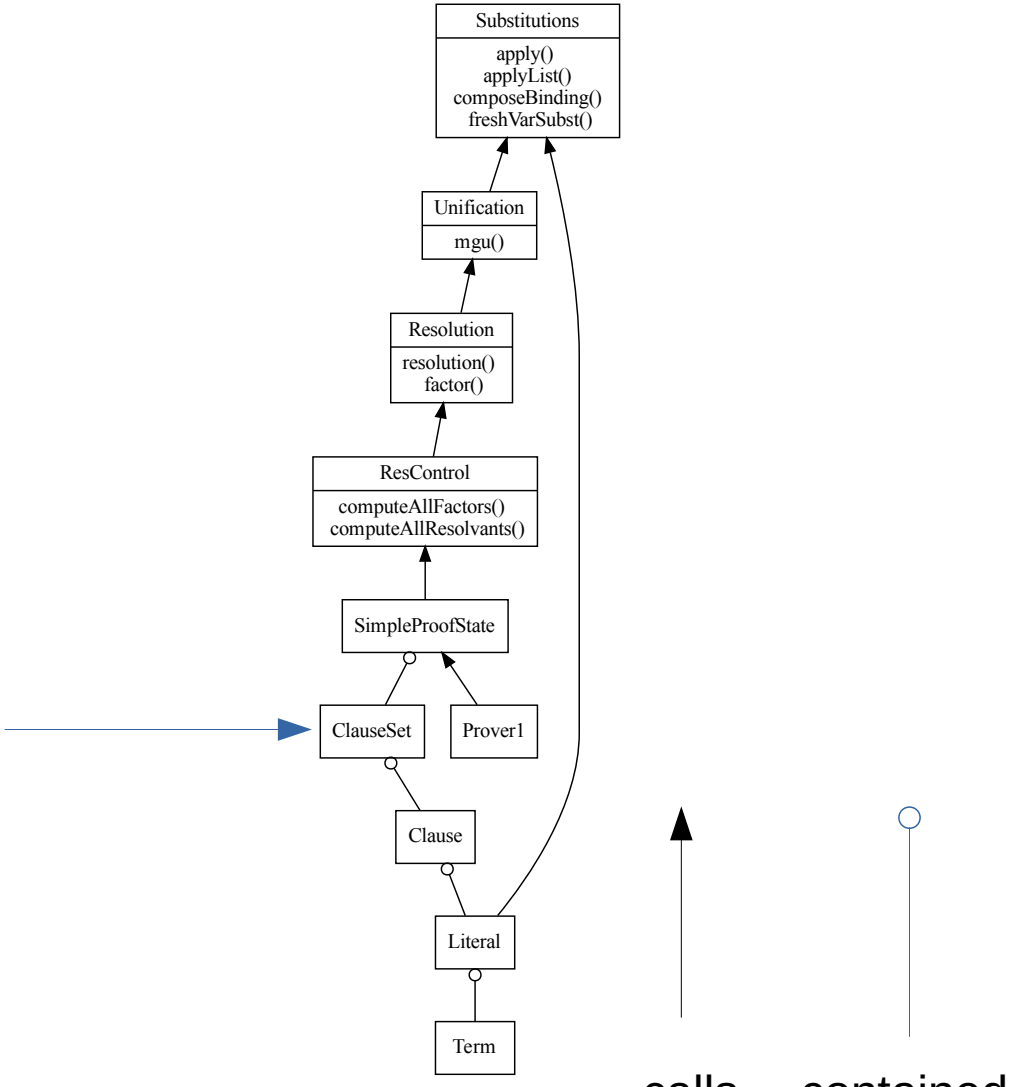
~likes(A,B) | likes(B,A)

A disjunction of (possibly negated) literals



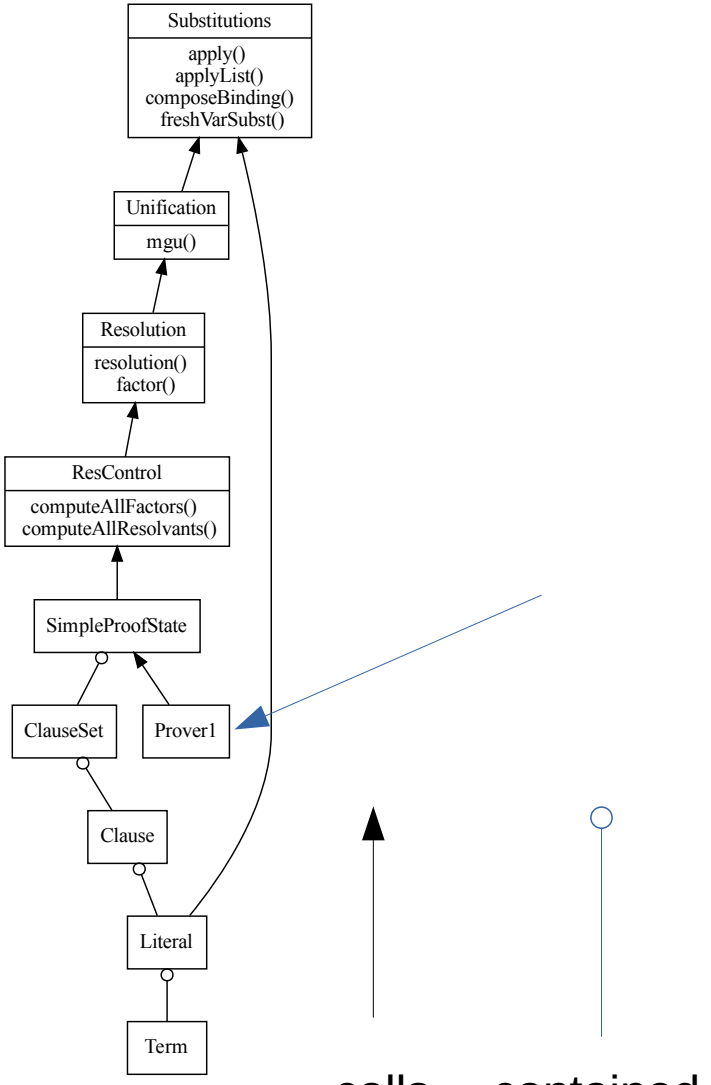
ClauseSet

A set of clauses
Implicitly a conjunction



Prover1

Just initialize a proof state and start it



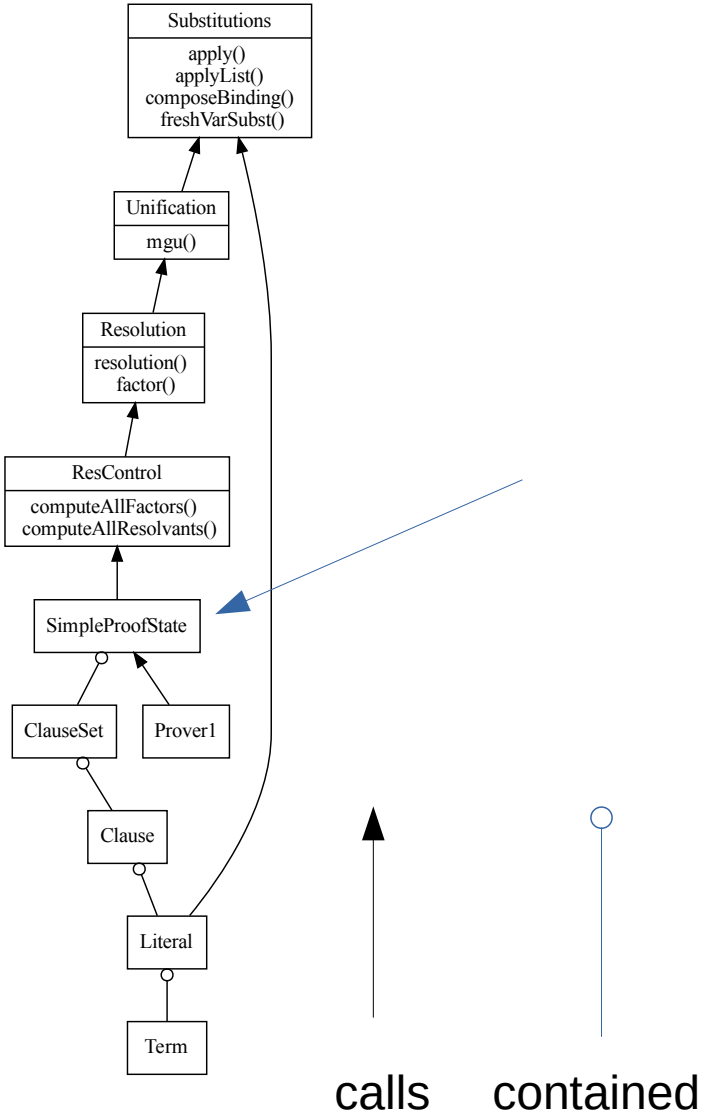
Another Algorithm: Factoring

```
   c|a|b
----------       where sigma = mgu(a,b)
sigma(c|a)
```

- g(X) | f(X) | g(a)
- Unifying the first and third literal yields the substitution
- {X → a}
- Return f(a) | g(a)

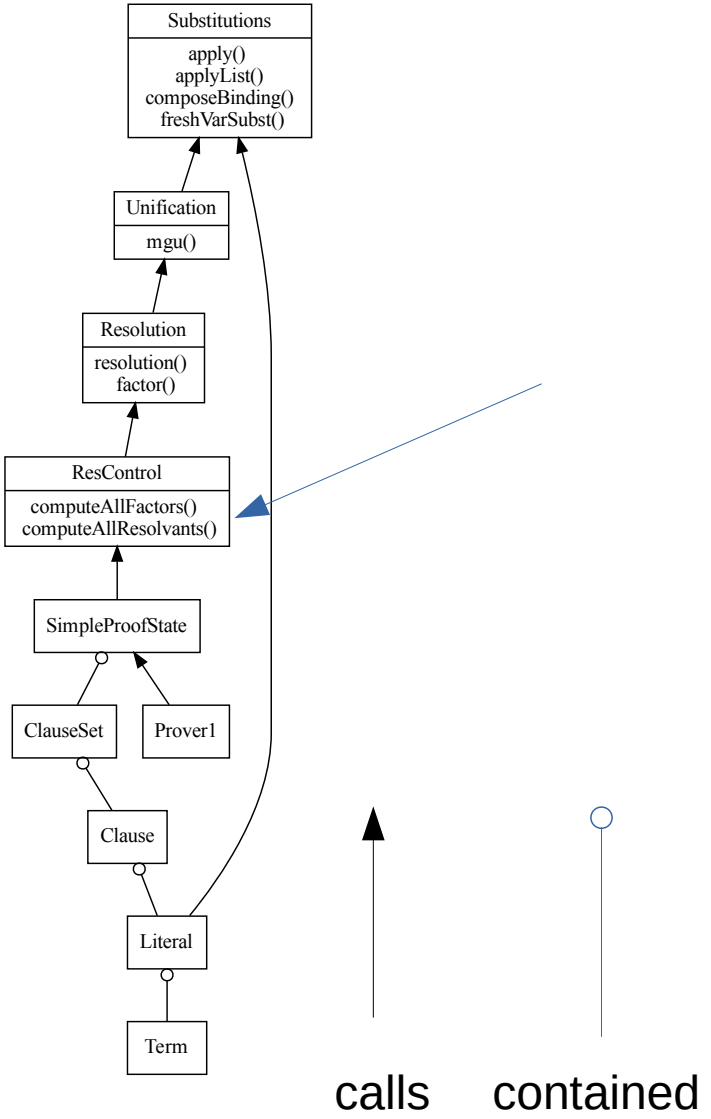
SimpleProofState

Iteration
  Picks a clause from the unprocessed list
    Call Compute factors
    Call Compute resolvants



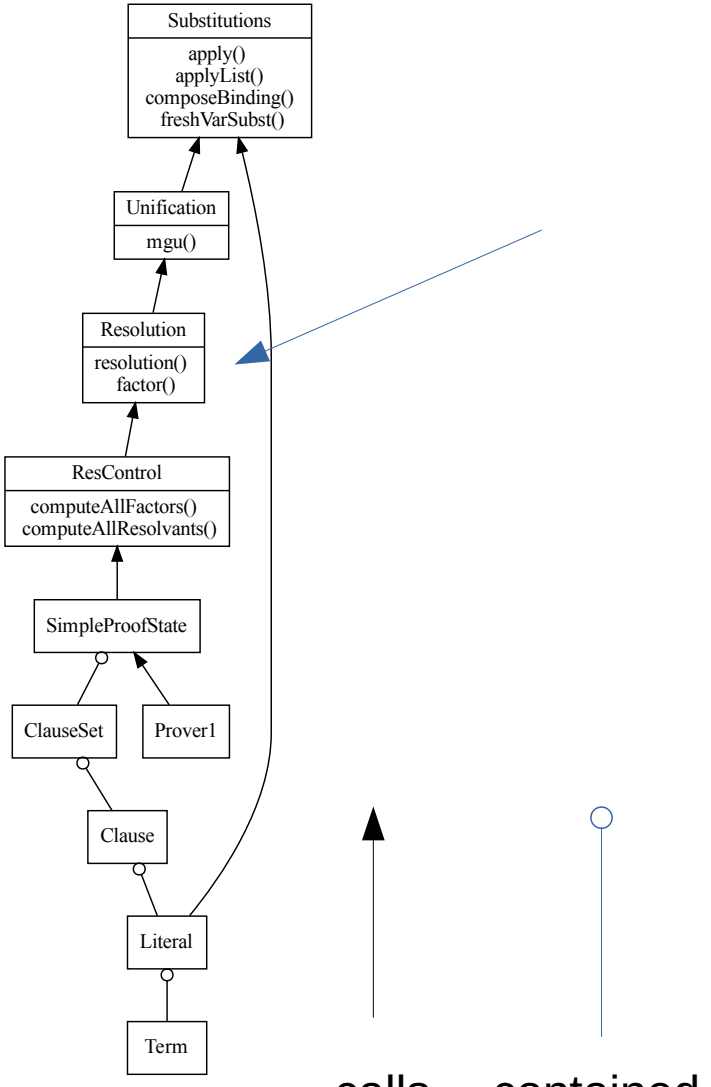
ResControl

Compute all factors for a given clause
    check all pairs of literals
Compute all resolvants for a given clause and clause set
    check all literals in the given clause against all
      clauses in the clause set



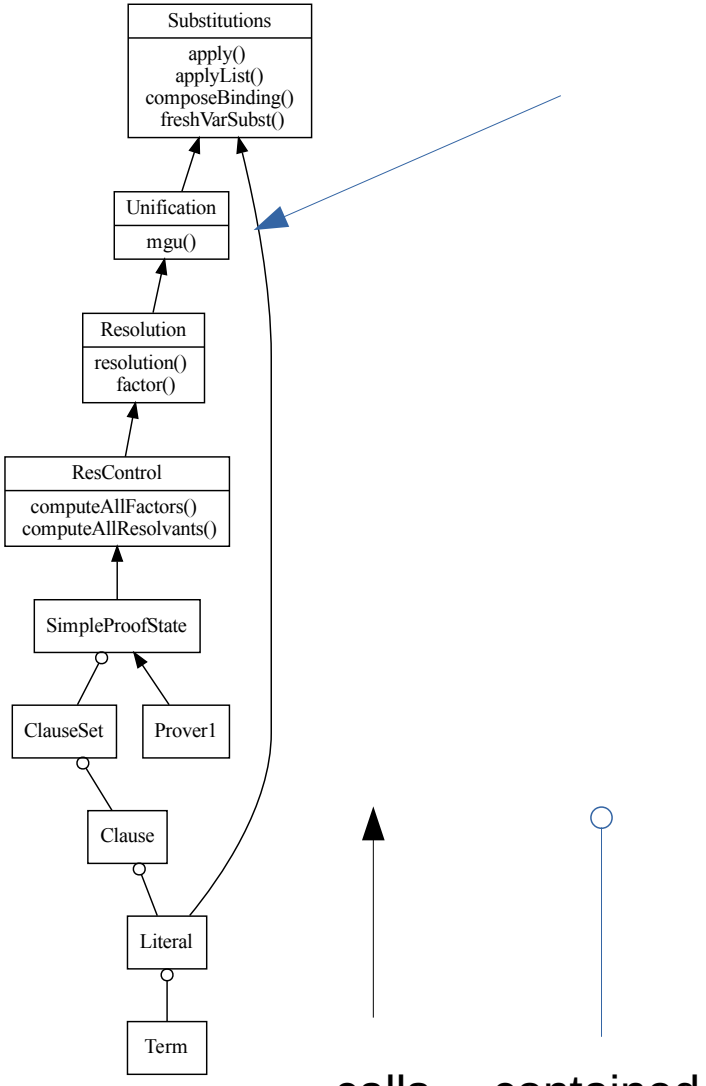
Resolution

Compute factors for a given clause of a given pair of literals
Compute resolvant for a given pair of clauses and literals



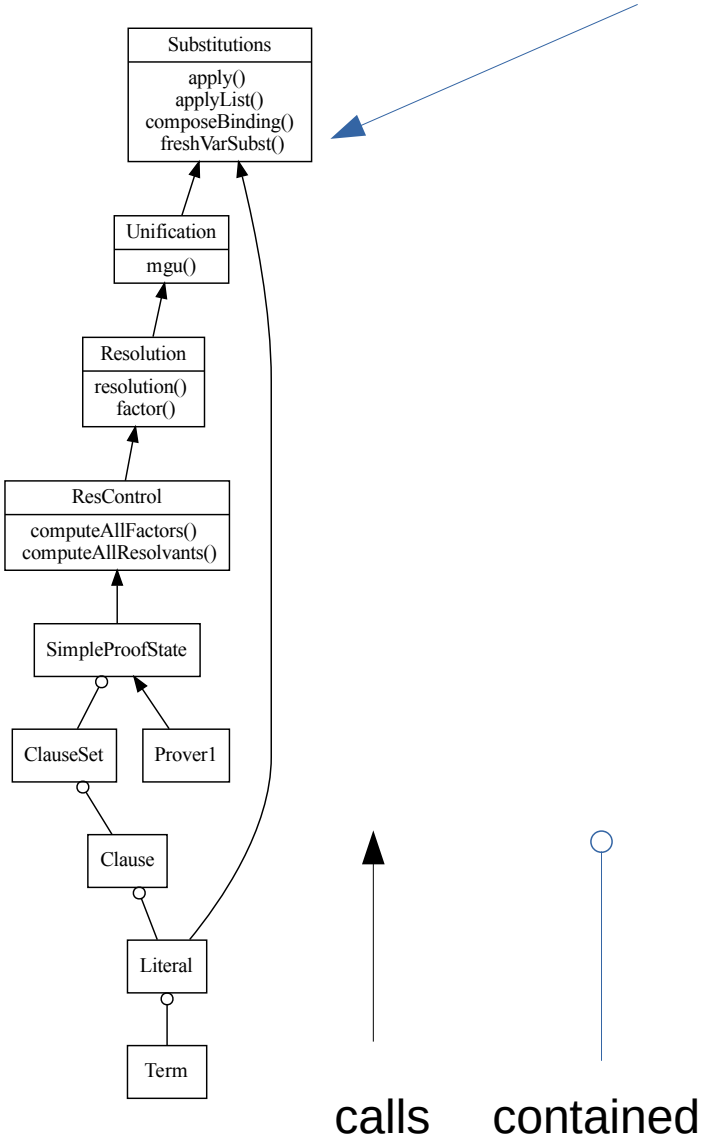
Unification

Compare two terms (iterating over their sub-terms)
- equal constants unify
- a variable unifies with a constant or a sub-term or variable
- apply substitution to the entire term upon unifying
  - a sub-term



Substitutions

Apply a substitution set to a term or list of terms
Compose two sets of substitutions
Rename variables
(variables have scope of a single clause)

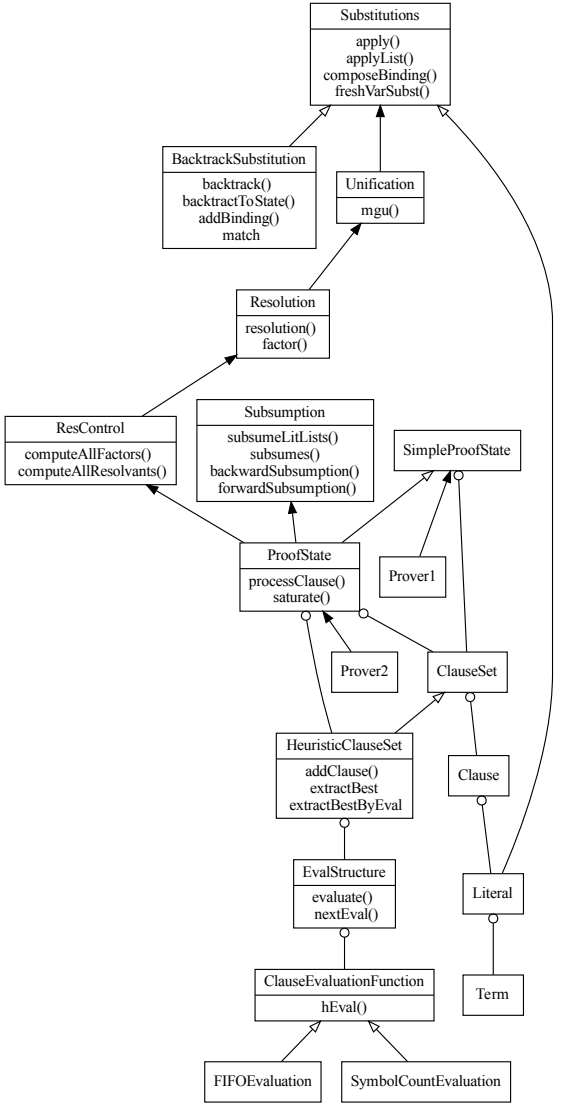


Extensions

- We have choices about which **clause**, **literal** or **term** to attempt to resolve in various steps - *selection*
  - This choice can be optimized
- Removing clauses that are redundant can reduce the search space – *subsumption*
- We can optimize how we find the next thing to try - *indexing*

New Architecture

- Backtrack substitution
- HeuristicClauseSet
  - EvalStructure
  - ClauseEvaluationFunction
    - FIFOEvaluation, SymbolCountEvaluation
- ProofState – displays proofs
- IndexedClauseSet – index by predicate name and whether negated



HeuristicClauseSet

- Backtrack substitution
- HeuristicClauseSet
  - EvalStructure
  - ClauseEvaluationFunction
    - FIFOEvaluation, **SymbolCountEvaluation**

SymbolCountEvaluation

- termWeight(f(a,b), 1, 1) = 3 - three symbols, weight of 1 = 1*3=3
- termWeight(f(a,b), 2, 1) = 6 - three symbols, weight of 2 = 2*3=6
- termWeight(X, 2, 1) = 1 - one variable with a weight of one = 1
- termWeight(g(a), 3, 1) = 6 - two symbols with a weight of 3 = 6

- We use function weight of 2, variable weight of 1

Literal Selection

- FIRST just leaves the literals in the order they appear in the clause
- SMALLEST sorts literals by their weight (see 10 above)
- LARGEST sorts literals by their weight (see 10 above) but in the opposite order from SMALLEST
- LEASTVARS sorts by the number of variables in the literal
- EQLEASTVARS sorts by whether a literal is a pure equality statement between variables (such as X=Y) and then by the literal with the smallest number of variables

Indexing

- Given
  - p(a, X) | p(X, a))
  - ~p(a, b) | p(f(Y), a))
- Positive
  - p -> { {p(a, X) | p(X, a)), {0,1}},
  - { {~p(a, b) | p(f(Y), a))}, {1}}
- Negative
  - p -> { {~p(a, b) | p(f(Y), a))}, {0}}

Equality Axioms

- Reflexivity: ![X]:X=X
- Symmetry: ![X,Y]:(X=Y -> Y=X)
- Transitivity: ![X,Y,Z]:((X=Y & Y=Z) -> X=Z)

Equality Axioms

- Given functions f and a and predicate p
- cnf(funcompat0,plain,~X1=Y1|~X2=Y2|~X3=Y3|
  f(X1,X2,X3)=f(Y1,Y2,Y3)).
- cnf(predcompat1,plain,~X1=Y1|~X2=Y2|~X3=Y3|~X4=Y4|
  ~X5=Y5|~p(X1,X2,X3,X4,X5)|p(Y1,Y2,Y3,Y4,Y5)).
- [cnf(funcompat2,plain,~X1=Y1|~X2=Y2|f(X1,X2)=f(Y1,Y2))., cnf(predcompat3,plain,~X1=Y1|~X2=Y2|~X3=Y3|
  ~p(X1,X2,X3)|p(Y1,Y2,Y3)).]

Metrics and Validation

- TPTP problem set – the yearly ATP competition

Number of problems answered correctly, by category (16094 total problems)

Category	UEQ	CNE	CEQ	FNE	FEQ	All
Class size	(1193)	(2383)	(4442)	(1771)	(6305)	(16094)
PyRes	113	945	499	632	725	2914
JavaRes	173	1081	615	737	1589	4195
E 2.4	813	1939	2648	1484	4054	10938
Prover9-1109a	728	1316	1678	709	2001	6432
LeanCoP 2.2	6	0	0	969	1826	2801

Size and Effort

- About three months of full-time effort in done in two chunks, for JavaRes
- JavaRes adds another CNF module, SinE axiom selection algorithm, proof graph generation etc
- JavaRes is 19,334 total lines of code, versus 8553 lines for PyRes (including comments, docstrings, and unit tests).
- actual production code, there are 7508 lines of effective code for JavaRes and only 3681 lines of effective code in PyRes

Conclusion

- It’s feasible for a novice to ATP (but experienced in FOL) to write an ATP system using PyRes as a model
- It performs pretty well, although not compared to the best FOL ATP systems
- It lacks superposition calculus for equality handling, which should make a big difference – maybe that’s our next effort
- Faster languages help – C++ would likely be better than Java or Python

Thanks Stephan!

Links

- https://www.ontologyportal.org
- https://github.com/ontologyportal
  - https://github.com/ontologyportal/JavaRes
  - https://github.com/eprover/PyRes
- https://www.youtube.com/user/peaseadam
- Schulz, S., Pease, A., (2020). Teaching Automated Theorem Proving by Example: PyRes 1.2 (system description). Proc. of IJCAR-2020: Vol. 12167, Lecture Notes in Computer Science, Springer. https://adampease.org
- Pease, A., Schulz, S., (2021). Learning Automated Theorem Proving from an Example: JavaRes, ThEdu workshop at CADE-21.