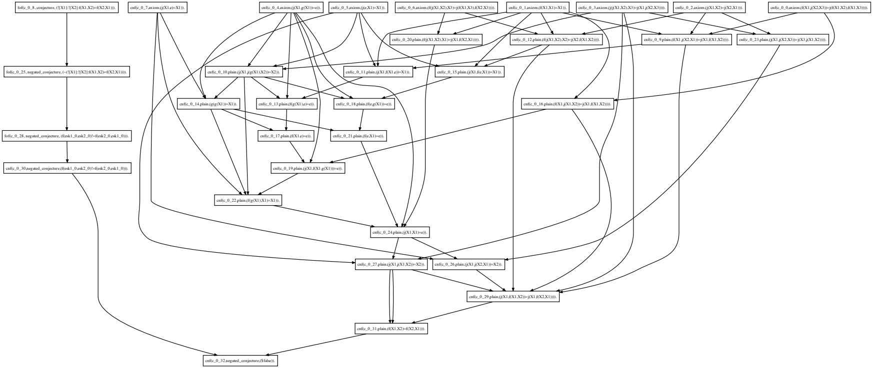


Credo quia absurdum (?)

Proof Generation for Saturating First-Order Theorem Provers

Stephan Schulz

DHBW
Duale Hochschule
Baden-Württemberg
Stuttgart

Geoff Sutcliffe

UNIVERSITY
OF MIAMI

Agenda

Structure and Representation of Proofs

Proof Generation

Proof Applications

Conclusion

Structure and Representation of Proofs

Refutational Theorem Proving

$\{A_1, A_2, \ldots, A_n\} \models C$

Refutational Theorem Proving

$$\{A_1, A_2, \ldots, A_n\} \models C$$

iff

$\{A_1, A_2, \ldots, A_n, \neg C\}$ is unsatisfiable

Refutational Theorem Proving

$$\{A_1, A_2, \ldots, A_n\} \models C$$

iff

$\{A_1, A_2, \ldots, A_n, \neg C\}$ is unsatisfiable

iff

$cnf(\{A_1, A_2, A_n, \neg C\})$ is unsatisfiable

Refutational Theorem Proving

$$\{A_1, A_2, \ldots, A_n\} \models C$$

iff

$$\{A_1, A_2, \ldots, A_n, \neg C\} \text{ is unsatisfiable}$$

iff

$$\text{cnf}(\{A_1, A_2, A_n, \neg C\}) \text{ is unsatisfiable}$$

iff

$$\text{cnf}(\{A_1, A_2, A_n, \neg C\}) \vdash \Box$$

Refutational Theorem Proving

$\{A_1, A_2, \ldots, A_n\} \models C$

iff

$\{A_1, A_2, \ldots, A_n, \neg C\}$ is unsatisfiable

iff

$\text{cnf}(\{A_1, A_2, A_n, \neg C\})$ is unsatisfiable

Clausification

iff

$\text{cnf}(\{A_1, A_2, A_n, \neg C\}) \vdash \Box$

Refutation

Ideal: Proofs as Sequences of Proof Steps

A derivation is a list of steps

Each step carries a clause/formula

Each step is either...

- Assumed (e.g. axioms, conjecture)
- Logically derived from earlier steps

A proof is a derivation that either...

- derives the conjecture
- derives a contradiction from the negated conjecture

Good mental model!

Reality: Proofs as Sequences of Proof Steps

Initial clauses/formulas

- Axioms/Conjectures/Hypotheses
- Justified by assumption

Derived clauses/formulas

- Justified by reference to (topologically) preceding steps
- Defined logical relationship to predecessors
  - Most frequent case: theorem of predecessors
  - Exceptions: Skolemization, negation of conjecture, . . .

(Introduced definitions)

- Don't affect satisfiability/provability
- Justified by definition

Logical Languages for FOF

Historical

Problems

- Otter "lists"
- LOP (CNF only)
- DFG
- TPTP (v1, v2)
- ...

Proofs/Derivations

- Otter "proof object"
- PCL (UEQ only)
- DFG (but nobody uses DFG)
- ...

Logical Languages for FOF

Historical

Problems

- Otter “lists”
- LOP (CNF only)
- DFG
- TPTP (v1, v2)
- ...

Proofs/Derivations

- Otter “proof object”
- PCL (UEQ only)
- DFG (but nobody uses DFG)
- ...

Modern Convergence: TPTP v3

TPTP v3 language

Consistent syntax for different classes

- CNF is sub-case of FOF
- FOF is sub-case of TFF

Applicable for a wide range of applications

- Problem specifications
- Proofs/derivations
- Models

Easily parsable

- Prolog-parsable
- Lex/Yacc grammar
- Recursive-descent with 1-token look-ahead

Widely used and supported

- CASC
- Major provers (E, SPASS, Vampire, iProver, ...)
- Used by integrators

Example

```
fof(c_0_0, conjecture, (?[X3]:(human(X3)&X3!=john)), file('humen.p', someone_not_john)).
fof(c_0_1, axiom, (?[X3]:(human(X3)&grade(X3)=a)), file('humen.p', someone_got_an_a)).
fof(c_0_2, axiom, (grade(john)=f), file('humen.p', john_failed)).
fof(c_0_3, axiom, (a!=f), file('humen.p', distinct_grades)).
fof(c_0_4, negated_conjecture, (~(?[X3]:(human(X3)&X3!=john))),
    inference(assume_negation,[status(cth)],[c_0_0])).
fof(c_0_5, negated_conjecture, (![X4]:(~human(X4)|X4=john)),
    inference(variable_rename,[status(thm)],[inference(fof_nnf,[status(thm)],[c_0_4])])).
fof(c_0_6, plain, ((human(esk1_0)&grade(esk1_0)=a)),
    inference(skolemize,[status(esa)],[inference(variable_rename,[status(thm)],[c_0_1])])).
cnf(c_0_7,negated_conjecture,(X1=john|~human(X1)),
    inference(split_conjunct,[status(thm)],[c_0_5])).
cnf(c_0_8,plain,(human(esk1_0)),
    inference(split_conjunct,[status(thm)],[c_0_6])).
cnf(c_0_9,plain,(grade(esk1_0)=a),
    inference(split_conjunct,[status(thm)],[c_0_6])).
cnf(c_0_10,negated_conjecture,(esk1_0=john),
    inference(spm,[status(thm)],[c_0_7, c_0_8])).
cnf(c_0_11,plain,(grade(john)=f),
    inference(split_conjunct,[status(thm)],[c_0_2])).
cnf(c_0_12,plain,(a!=f),
    inference(split_conjunct,[status(thm)],[c_0_3])).
cnf(c_0_13,plain,($false),
    inference(sr,[status(thm)],[inference(rw,[status(thm)],
    [inference(rw,[status(thm)],[c_0_9, c_0_10]), c_0_11]), c_0_12]), ['proof']).
```

```
cnf(c_0_13,
    plain,
    ($false),
    inference(sr,[status(thm)],
              [inference(rw,[status(thm)],
                         [inference(rw,[status(thm)],
                                    [c_0_9, c_0_10]),
                          c_0_11]),
               c_0_12]),
    ['proof']).
```

```
cnf(c_0_13,
   plain,
    ($false),
    inference(sr,[status(thm)],
              [inference(rw,[status(thm)],
                         [inference(rw,[status(thm)],
                                    [c_0_9, c_0_10]),
                          c_0_11]),
               c_0_12]),
    ['proof']).
```

Language
(cnf, fof, tff, ...)

```
cnf(c_0_13,
     plain,
     ($false),
     inference(sr,[status(thm)],
               [inference(rw,[status(thm)],
                          [inference(rw,[status(thm)],
                                     [c_0_9, c_0_10]),
                           c_0_11]),
                c_0_12]),
     ['proof']).
```

Language
(cnf, fof, tff, ...)

Name
(arbitrary, but
unique)

```
cnf(c_0_13,
   plain,
    ($false),
    inference(sr,[status(thm)],
              [inference(rw,[status(thm)],
                         [inference(rw,[status(thm)],
                                    [c_0_9, c_0_10]),
                          c_0_11]),
               c_0_12]),
    ['proof']).
```

Language (cnf, fof, tff, ...)

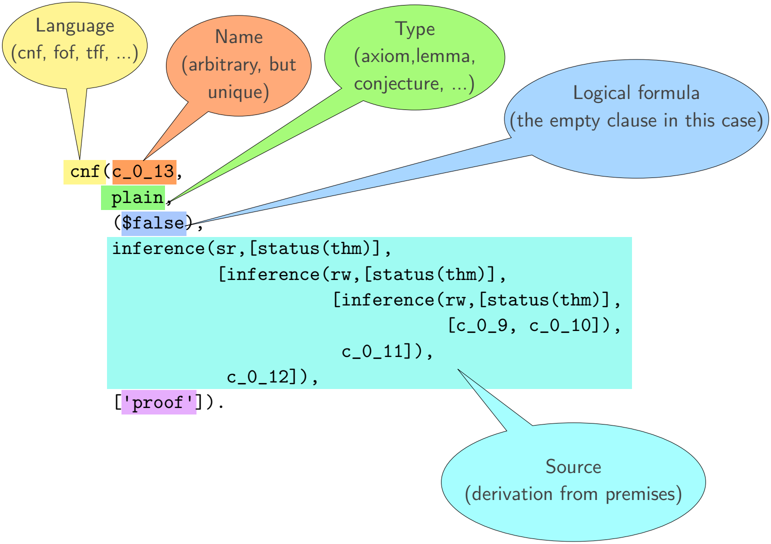
Name (arbitrary, but unique)

Type (axiom,lemma, conjecture, ...)

```
cnf(c_0_13,
   plain,
   ($false),
   inference(sr,[status(thm)],
             [inference(rw,[status(thm)],
                        [inference(rw,[status(thm)],
                                   [c_0_9, c_0_10]),
                         c_0_11]),
              c_0_12]),
   ['proof']).
```

- Language (cnf, fof, tff, ...)
- Name (arbitrary, but unique)
- Type (axiom,lemma, conjecture, ...)
- Logical formula (the empty clause in this case)

```
cnf(c_0_13,
   plain,
    ($false),
    inference(sr,[status(thm)],
              [inference(rw,[status(thm)],
                          [inference(rw,[status(thm)],
                                      [c_0_9, c_0_10]),
                           c_0_11]),
               c_0_12]),
    ['proof']).
```


Language
(cnf, fof, tff, ...)
Name
(arbitrary, but
unique)
Type
(axiom,lemma,
conjecture, ...)
Logical formula
(the empty clause in this case)
cnf(c_0_13,
plain,
($false),
inference(sr,[status(thm)],
[inference(rw,[status(thm)],
[inference(rw,[status(thm)],
[c_0_9, c_0_10]),
c_0_11]),
c_0_12]),
['proof']).
Source
(derivation from premises)

```
cnf(c_0_13,
   plain,
   ($false),
   inference(sr,[status(thm)],
             [inference(rw,[status(thm)],
                        [inference(rw,[status(thm)],
                                   [c_0_9, c_0_10]),
                         c_0_11]),
              c_0_12]),
   ['proof']).
```

Language (cnf, fof, tff, ...)

Name (arbitrary, but unique)

Type (axiom,lemma, conjecture, ...)

Logical formula (the empty clause in this case)

Source (derivation from premises)

Optional "useful information" (this step concludes the proof)

```
cnf(c_0_13,
    plain,
    ($false),
    inference(sr,[status(thm)],
              [inference(rw,[status(thm)],
                         [inference(rw,[status(thm)],
                                    [c_0_9, c_0_10]),
                          c_0_11]),
               c_0_12]),
    ['proof']).
```

```
cnf(c_0_13,
    plain,
    ($false),
    inference(sr,[status(thm)],
              [inference(rw,[status(thm)],
                         [inference(rw,[status(thm)],
                                    [c_0_9, c_0_10]),
                          c_0_11]),
               c_0_12]),
    ['proof']).
```

```
cnf(c_0_13,
    plain,
    ($false),
    inference(sr,[status(thm)],
              [inference(rw,[status(thm)],
                         [inference(rw,[status(thm)],
                                    [c_0_9, c_0_10]),
                          c_0_11]),
               c_0_12]),
    ['proof']).
```

Inference rule (sr: Simplify-reflect, rw: Rewriting, pm: Paramodulation, ...)

```
cnf(c_0_13,
    plain,
    ($false),
    inference(sr,[status(thm)],
              [inference(rw,[status(thm)],
                         [inference(rw,[status(thm)],
                                    [c_0_9, c_0_10]),
                          c_0_11]),
               c_0_12]),
    ['proof']).
```

Inference rule (sr: Simplify-reflect, rw: Rewriting, pm: Paramodulation, ...)

"Useful information": logical status (formula is theorem of premises)

```
cnf(c_0_13,
    plain,
    ($false),
    inference(sr,[status(thm)],
              [inference(rw,[status(thm)],
                         [inference(rw,[status(thm)],
                                    [c_0_9, c_0_10]),
                          c_0_11]),
               c_0_12]),
    ['proof']).
```

Inference rule (sr: Simplify-reflect, rw: Rewriting, pm: Paramodulation, ...)

"Useful information": logical status (formula is theorem of premises)

```
cnf(c_0_13,
    plain,
    ($false),
    inference(sr,[status(thm)],
              [inference(rw,[status(thm)],
                         [inference(rw,[status(thm)],
                                    [c_0_9, c_0_10]),
                          c_0_11]),
               c_0_12]),
    ['proof']).
```

Names of the premises

```
cnf(c_0_13,
    plain,
    ($false),
    inference(sr,[status(thm)],
              [inference(rw,[status(thm)],
                         [inference(rw,[status(thm)],
                                    [c_0_9, c_0_10]),
                          c_0_11]),
               c_0_12]),
    ['proof']).
```

```
cnf(c_0_13,
    plain,
    ($false),
    inference(sr,[status(thm)],
              [inference(rw,[status(thm)],
                         [inference(rw,[status(thm)],
                                    [c_0_9, c_0_10]),
                          c_0_11]),
               c_0_12]),
    ['proof']).
```

```
c_0_9:  grade(esk1_0)=a
c_0_10: esk1_0=john
c_0_11:grade(john)=f
c_0_12: a!=f
```

```
cnf(c_0_13,
    plain,
    ($false),
    inference(sr,[status(thm)],
              [inference(rw,[status(thm)],
                           [inference(rw,[status(thm)],
                                      [c_0_9, c_0_10]),
                            c_0_11]),
               c_0_12]),
    ['proof']).
```

```
c_0_9:  grade(esk1_0)=a
c_0_10: esk1_0=john
c_0_11:grade(john)=f
c_0_12: a!=f
```

Innermost inference:
Rewrite c_0_9 with
c_0_10

```
cnf(c_0_13,
    plain,
    ($false),
    inference(sr,[status(thm)],
              [inference(rw,[status(thm)],
                         [inference(rw,[status(thm)],
                                    [c_0_9, c_0_10]),
                          c_0_11]),
               c_0_12]),
    ['proof']).
```

Intermediate inference: Rewrite the result of the innermost inference with c_0_11

Innermost inference: Rewrite c_0_9 with c_0_10

```
c_0_9:  grade(esk1_0)=a
c_0_10: esk1_0=john
c_0_11:grade(john)=f
c_0_12: a!=f
```

Outermost (final) inference: Cut off a literal from the result of the intermediate inference with c_0_12

Intermediate inference: Rewrite the result of the innermost inference with c_0_11

```
cnf(c_0_13,
    plain,
    ($false),
    inference(sr,[status(thm)],
              [inference(rw,[status(thm)],
                         [inference(rw,[status(thm)],
                                    [c_0_9, c_0_10]),
                          c_0_11]),
               c_0_12]),
    ['proof']).
```

Innermost inference: Rewrite c_0_9 with c_0_10

```
c_0_9:  grade(esk1_0)=a
c_0_10: esk1_0=john
c_0_11:grade(john)=f
c_0_12: a!=f
```

Compl[ie]mentary Example

```
fof(c_0_1,
    axiom,
    (?[X3]:(human(X3)&grade(X3)=a)),
    file('humen.p', someone_got_an_a)).
```

TPTP v3 idiosyncrasies

No inference semantics

- Rules are just names
- Rules are system-dependent

Incomplete inference description

- “Rules are just names”
- Syntactic support not widely supported

TPTP v3 idiosyncrasies

No inference semantics

- Rules are just names
- Rules are system-dependent

Incomplete inference description

- "Rules are just names"
- Syntactic support not widely supported

Workarounds:

- Inference status
- Proof reconstruction

Proof Generation

Refutational Theorem Proving

$$\{A_1, A_2, \ldots, A_n\} \models C$$

iff

$\{A_1, A_2, \ldots, A_n, \neg C\}$ is unsatisfiable

iff

$cnf(\{A_1, A_2, A_n, \neg C\})$ is unsatisfiable

Clausification

iff

$$cnf(\{A_1, A_2, A_n, \neg C\}) \vdash \Box$$

Refutation

Clausification and Saturation

Clausification

- Terminating
- (Usually) deterministic
- (Usually) non-destructive
- Sometimes done by external tool

Saturation

- Many degrees of freedom
- Arbitrary search time
- Generating inferences
  - Create new clauses
  - Necessary for completeness
- Simplifying inferences
  - Modify/remove existing clauses
  - Necessary for performance

Clausification and Saturation

Clausification

- Terminating
- (Usually) deterministic
- (Usually) non-destructive
- Sometimes done by external tool

Saturation

- Many degrees of freedom
- Arbitrary search time
- Generating inferences
  - Create new clauses
  - Necessary for completeness
- Simplifying inferences
  - Modify/remove existing clauses
  - Necessary for performance

Recording clausification is straightforward

- ...but not always done

Efficiently recording saturation is difficult

- ...some settle for inefficient

Deduction vs. Simplification

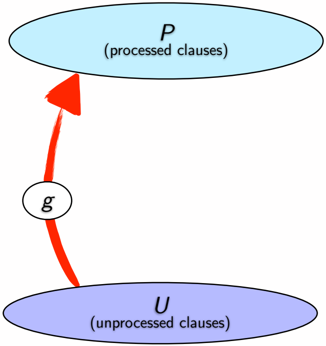
Superposition $$\frac{s \simeq t \vee S \quad u \not\simeq v \vee R}{\sigma(u[p \leftarrow t] \not\simeq v \vee S \vee R)}$$

if $\sigma = mgu(u|_p, s)$, [...]

Rewriting $$\frac{s \simeq t \quad u \not\simeq v \vee R}{s \simeq t \quad u[p \leftarrow \sigma(t)] \not\simeq v \vee R}$$

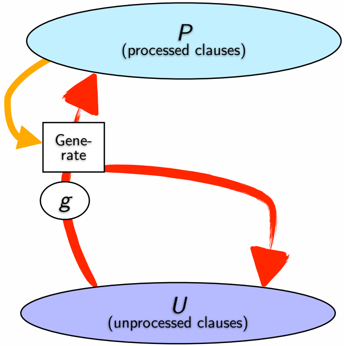
if $u|_p = \sigma(s)$ and $\sigma(s) > \sigma(t)$

The Given-Clause Algorithm



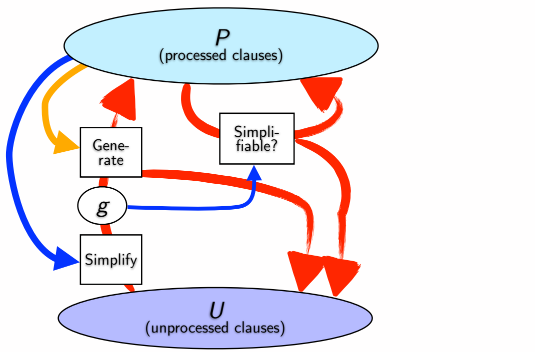
- Aim: Move everything from *U* to *P*

The Given-Clause Algorithm



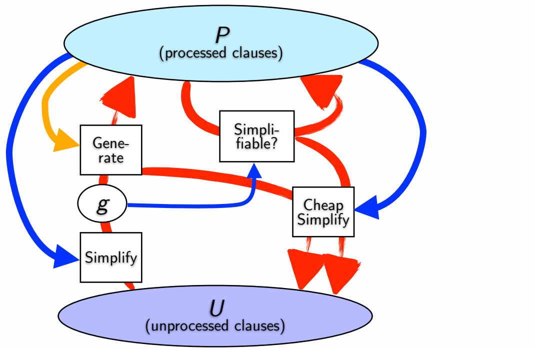
- Aim: Move everything from U to P
- Invariant: All generating inferences with premises from P have been performed

The Given-Clause Algorithm



- Aim: Move everything from *U* to *P*
- Invariant: All generating inferences with premises from *P* have been performed
- Invariant: *P* is interreduced

The Given-Clause Algorithm



- Aim: Move everything from U to P
- Invariant: All generating inferences with premises from P have been performed
- Invariant: P is interreduced
- Clauses added to U are simplified with respect to P

Naive Proof Generation

Basic approach:

- Store (or dump) all intermediate proof steps
- Extract proof steps in post-processing

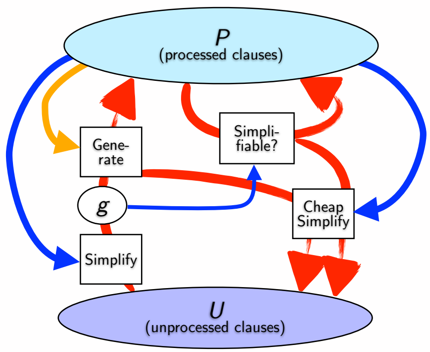
Problem: Necessary steps only known after the proof concludes

- Intermediate results are expensive to store
- Example: A ring with $X^4 = X$ is Abelian
  - Proof search (E): 5.4s
  - Proof search with inference dump: 11.4s
  - Post-processing: 17.6s
  - Temporary file size: 480 000 steps, 117MB
  - Proof size: 154 steps, 31 kB

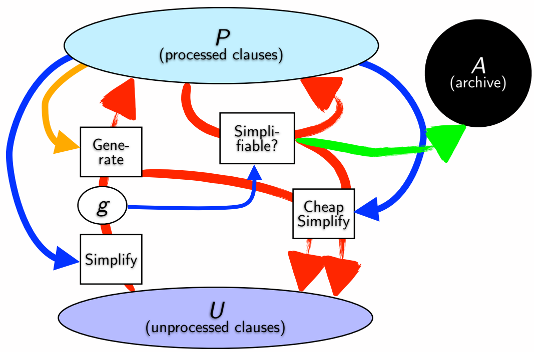
Only suitable for small problems/short run-times

Optimized Proof Object Construction

Observation: Only clauses in *P* are premises!



Optimized Proof Object Construction

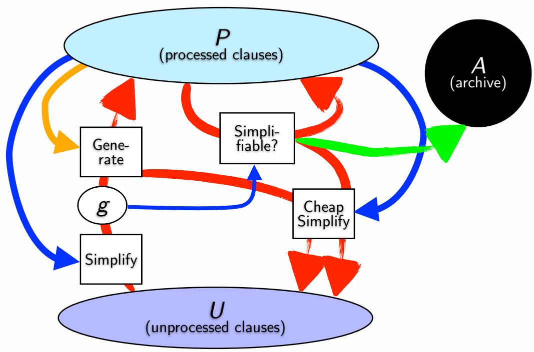


Observation: Only clauses in *P* are premises!

Proof recording:

- Simplified *P*-clauses are archived
- Clauses record their history
  - Inference rules
  - *P*-clauses involved

Optimized Proof Object Construction



Observation: Only clauses in *P* are premises!

Proof recording:

- Simplified *P*-clauses are archived
- Clauses record their history
  - Inference rules
  - *P*-clauses involved

Proof extraction

- Track parent relation
- Topological sort
- Print proof

Optimized Proof Generation

Example: A ring with $X^4 = X$ is Abelian

- Naive approach
  - Proof search (E): 5.4s
  - Proof search with inference dump: 11.4s
  - Post-processing: 17.6s
  - Temporary file size: 480 000 steps, 117MB
  - Proof size: 154 steps, 31 kB
- Optimized approach
  - Proof search (E): 5.5s
  - Proof search with inference dump: -
  - Post-processing: -
  - Temporary file size: -
  - Proof size: 154 steps, 31 kB
- Example is typical
  - Optimized overhead: 0.24% over TPTP 5.4.0

Proof Applications

Why Proofs?

Trust

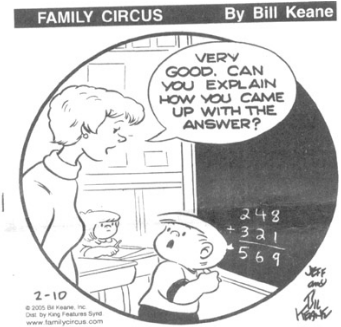
- in the proof
- in the ATP system
- in the specification

Understanding

- of the proof
- of the domain
- of the search process

Learning

- of important domain statements
- of search control information
- of the domain structure



"No, I think it was just Divine intervention."

Proof Checking

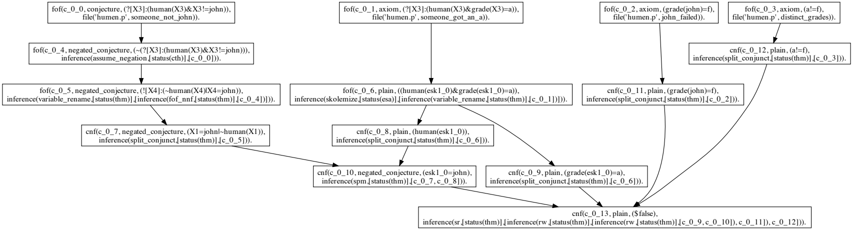
Semantic proof checking

- Step-by-step check
- Verify semantic status (conclusion can be derived “somehow” from premises)
- Use alternative theorem prover (or configuration)

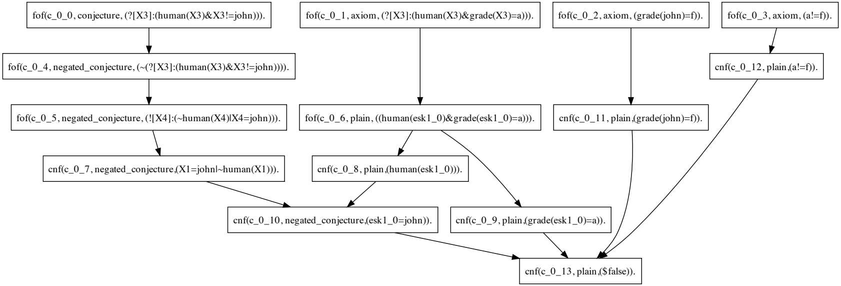
Syntactic proof checking

- Show correctness of individual inference rule applications
- With TPTP syntax: Requires proof reconstruction
- E.g. Metis in Isabelle/Sledgehammer

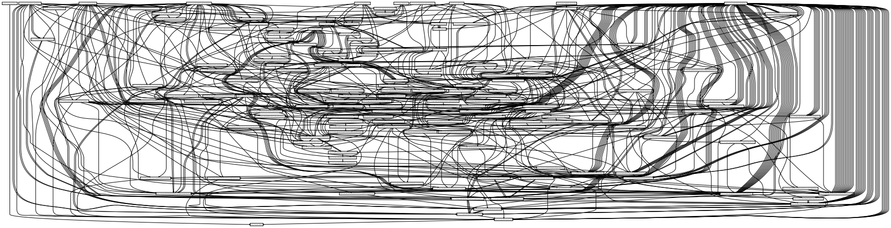
Proof Visualization



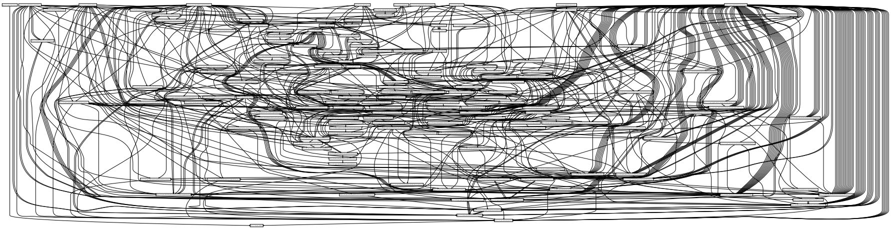
Proof Visualization



Another Example

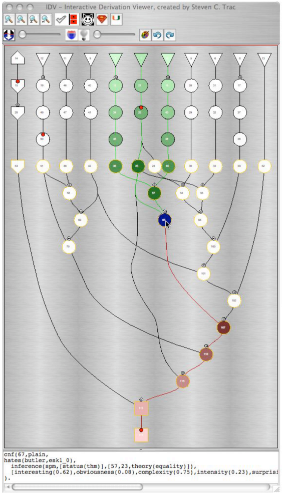


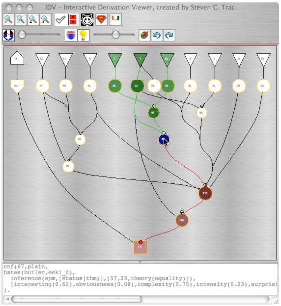
Another Example



(A ring with $X^4 = X$ is Abelian)

Interactive Visualization





Learning

Heuristics learning

- Find formulas that frequently appear in proofs
- Generalize and reuse

Axiom selection

- Learn relationship between conjecture and useful axioms

. . .

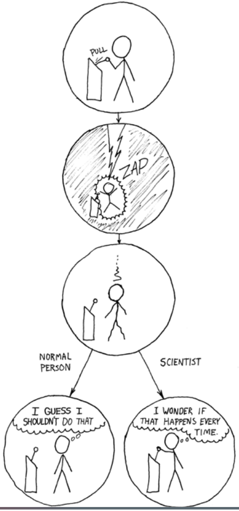


Image credit: `http://xkcd.com/242/`

Conclusion

Efficient proof generation is non-trivial, but possible

TPTP v3 is a useful and used standard for proof representation

Proof objects are useful for trust building and learning

Use of proof objects is still in its infancy - we need more tools

Conclusion

Efficient proof generation is non-trivial, but possible

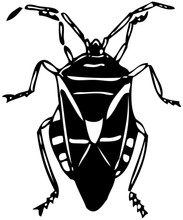
TPTP v3 is a useful and used standard for proof representation

Proof objects are useful for trust building and learning

Use of proof objects is still in its infancy - we need more tools

Proof presentation is a big open area

Ceterum Censeo...



Bug reports for E should include:

- The exact command line leading to the bug
- All input files needed to reproduce the bug
- A description of what seems wrong
- **The output of `eprover --version`**