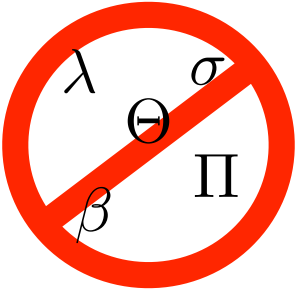
Pragmatics of Efficient Proof Generation

Stephan Schulz




```c
void ClausePushDerivation(Clause_p clause, DerivationCode op,
                          void* arg1, void* arg2)
{
   assert(clause);
   assert(op);

   CLAUSE_ENSURE_DERIVATION(clause);
   assert(DCOpHasCnfArg1(op)||DCOpHasFofArg1(op)||!arg1);
   assert(DCOpHasCnfArg2(op)||DCOpHasFofArg2(op)||!arg2);
   assert(DCOpHasCnfArg1(op)||!DCOpHasCnfArg2(op));

   PStackPushInt(clause->derivation, op);
   if(arg1)
   {
      PStackPushP(clause->derivation, arg1);
      if(arg2)
      {
         PStackPushP(clause->derivation, arg2);
      }
   }
}
```

DHBW Duale Hochschule Baden-Württemberg Stuttgart

Introduction

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

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

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

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

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

iff

$$\mathrm{cnf}(\{A_1, A_2, A_n, \neg C\}) \vdash^{*} \square$$

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

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

Clausification

iff

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

Refutation/
Saturation

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

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)

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']).
```

Optional "useful
information" (this step
concludes the 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']).
```

```
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']).
```

```
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

grade(john)=a

```
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

f=a

```
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

grade(john)=a

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

f=a

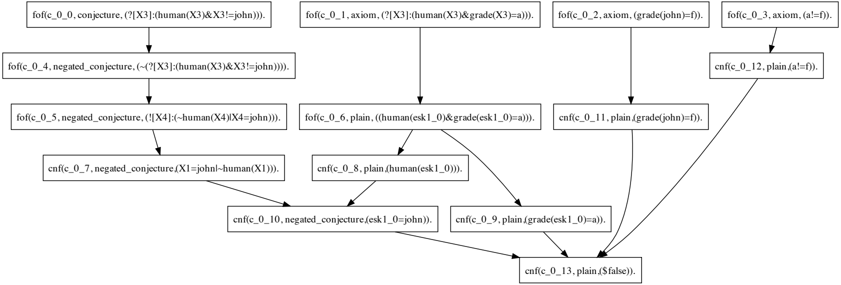
```
cnf(c_0_13,
    plain,
    ($false),
    inference(sr,[status(thm)],
              [inference(rw,[status(thm)],
                           [inference(rw,[status(thm)],
                                      [c_0_9, c_0_10]),
                            c_0_11]),
               c_0_12]),
    ['proof']).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

```
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

grade(john)=a

Proofs as Graphs



Proof Generation and Representation

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

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

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

- 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 $\sigma(s) > \sigma(t)$

Deduction vs. Simplification

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

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

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

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

- Generating inferences *create* new clauses

Deduction vs. Simplification

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

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

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

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

- Generating inferences *create* new clauses
- Simplifiying inferences *replace* or remove clauses

Clauses vs. Clause Objects

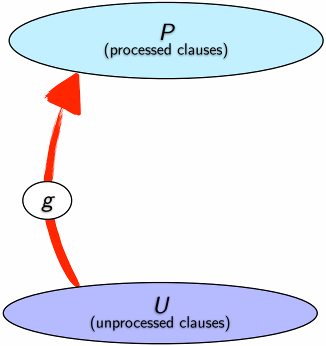
Clauses	Clause Objects
Logical abstraction	Real data structures
Purely logical object	Can carry meta-information and history
Immutable (changing a clause creates a completely new clause)	Mutable, changes are frequent
Used in theoretical inference systems	Implemented in actual provers

Clauses vs. Clause Objects

Clauses	Clause Objects
Logical abstraction	Real data structures
Purely logical object	Can carry meta-information and history
Immutable (changing a clause creates a completely new clause)	Mutable, changes are frequent
Used in theoretical inference systems	Implemented in actual provers

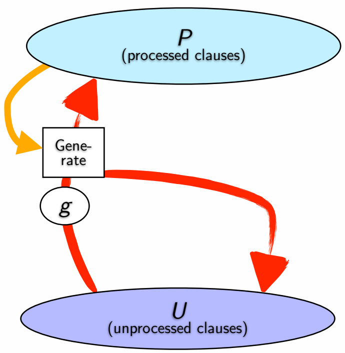
Clause objects represent a sequence of clauses over time (and possibly their history)

The Given-Clause Algorithm



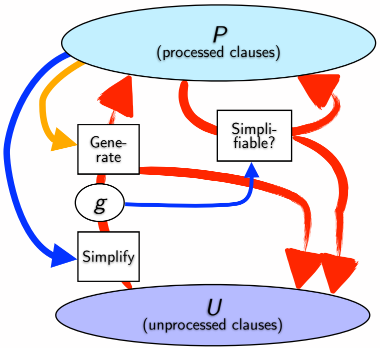
- Aim: Move everything from U to P

The Given-Clause Algorithm



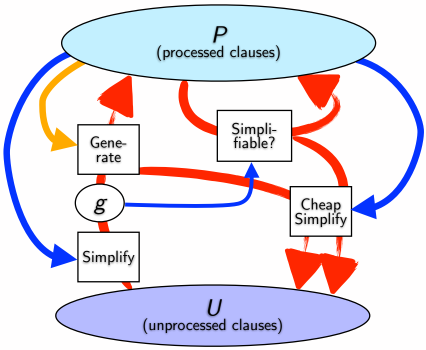
- Aim: Move everything from U to P
- Invariant: All generating inferences with premises from P have been performed

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

Typical Clause Object Lifecycle

- Generating inference *creates* a new clause object
  - Usually paramodulation (but may be equality factoring, equality resolution, ...)
- Simplifying inferences *modifies* the clause object
  - Multiple rewrite steps
  - Possibly literal cutting, trivial literal removal, ...
  - This modifies the existing clause object...
  - $\approx$10 modifications per clause on average (varies wildly)
- Deleting inference *removes* clause, but not (always) clause object
  - Subsumption
  - Tautology deletion
  - Typically $\approx$90% of all clauses

Typical Clause Object Lifecycle

- Generating inference *creates* a new clause object
  - Usually paramodulation (but may be equality factoring, equality resolution, . . .)
- Simplifying inferences *modifies* the clause object
  - Multiple rewrite steps
  - Possibly literal cutting, trivial literal removal, . . .
  - This modifies the existing clause object. . .
  - $\approx$10 modifications per clause on average (varies wildly)
- Deleting inference *removes* clause, but not (always) clause object
  - Subsumption
  - Tautology deletion
  - Typically $\approx$90% of all clauses

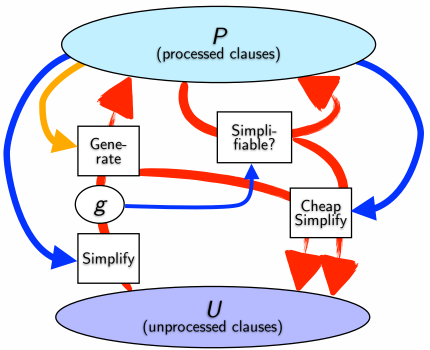
90% of clauses eventually deleted, 9 modified versions
$\Longrightarrow$ 99% of (logical) clauses are not persistent

Typical Clause Object Lifecycle

- Generating inference *creates* a new clause object
  - Usually paramodulation (but may be equality factoring, equality resolution, ...)
- Simplifying inferences *modifies* the clause object
  - Multiple rewrite steps
  - Possibly literal cutting, trivial literal removal, ...
  - This modifies the existing clause object...
  - $\approx$10 modifications per clause on average (varies wildly)
- Deleting inference *removes* clause, but not (always) clause object
  - Subsumption
  - Tautology deletion
  - Typically $\approx$90% of all clauses

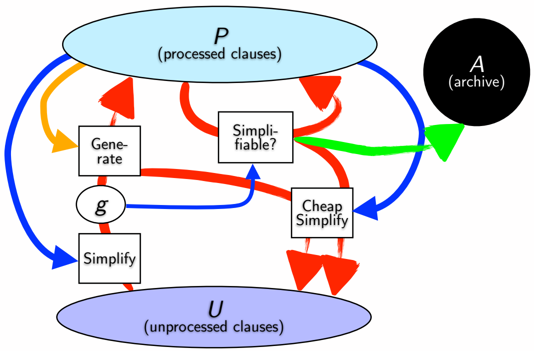
Storing all clauses is too expensive, but
we don't know a-priori which clauses are needed!

Optimized Proof Object Construction



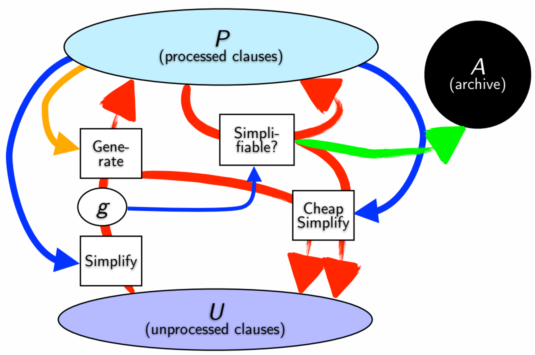
- Observation: Only clauses in *P* are premises!

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

Real Clauses (excerpt)

```
typedef struct clause_cell
{
   long               ident;       /* Hopefully unique ident for
                                      all clauses created during
                                      proof run */
   ...
   Eqn_p              literals;    /* List of literals */
   ...
   FormulaProperties  properties;/* Anything we want to note at
                                      the clause? */
   ...
   ClauseInfo_p       info;        /* Currently about source in
                                      input, NULL for derived
                                      clauses */
   PStack_p           derivation;/* Derivation of the clause for
                                      proof reconstruction. */
   ...
}ClauseCell, *Clause_p;
```

Internal Representation

```
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']).
```

Internal Representation

```
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']).
```

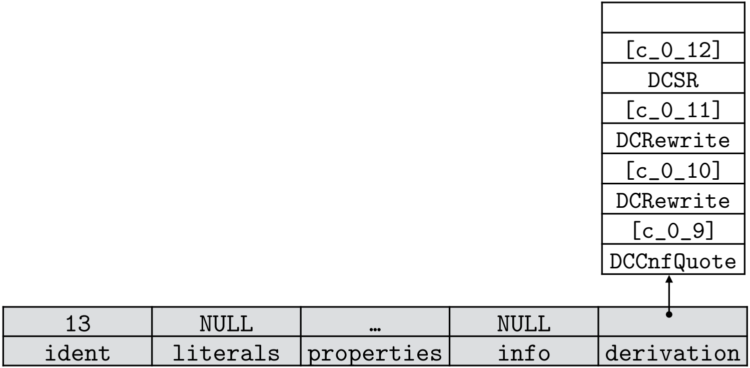
Internal Representation

```
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']).
```

13	NULL	…	NULL	
ident	literals	properties	info	derivation

Internal Representation

```
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']).
```



Internal Representation

```
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']).
```

...clause 12	[c_0_12]
...and looses a literal by cutting with...	DCSR
...clause 11...	[c_0_11]
...and further rewritten with...	DCRewrite
...with clause number 10	[c_0_10]
It was modified by rewriting...	DCRewrite
...of clause number 9	[c_0_9]
Original clause is copy...	DCCnfQuote

13	NULL	...	NULL	
ident	literals	properties	info	derivation

Encoding Operations (except)

```
typedef enum
{
    DONop,
    DOQuote,
    DORewrite,
    DOApplyDef,
    DOSR,
    ...
    DONegateConjecture,
    DOFofSimplify,
    DOFNNF,
    DOShiftQuantors,
    DOSkolemize,
    ...
    DOParamod,
    DOSimParamod,
    DOOrderedFactor,
    ...
}OpCode;
```

Encoding Operations (except)

```
typedef enum
{
    DONop,
    DOQuote,
    DORewrite,
    DOApplyDef,
    DOSR,
    ...
    DONegateConjecture,
    DOFofSimplify,
    DOFNNF,
    DOShiftQuantors,
    DOSkolemize,
    ...
    DOParamod,
    DOSimParamod,
    DOOrderedFactor,
    ...
}OpCode;
```

```
typedef enum
{
    Arg1Fof = 1<<8,
    Arg1Cnf = 1<<9,
    Arg1Num = 1<<10,
    Arg2Fof = 1<<11,
    ...
}ArgDesc;
```

Encoding Operations (except)

```
typedef enum
{
    DONop,
    DOQuote,
    DORewrite,
    DOApplyDef,
    DOSR,
    ...
    DONegateConjecture,
    DOFofSimplify,
    DOFNNF,
    DOShiftQuantors,
    DOSkolemize,
    ...
    DOParamod,
    DOSimParamod,
    DOOrderedFactor,
    ...
}OpCode;
```

```
typedef enum
{
    Arg1Fof  = 1<<8,
    Arg1Cnf  = 1<<9,
    Arg1Num  = 1<<10,
    Arg2Fof  = 1<<11,
    ...
}ArgDesc;
```

```
typedef enum
{
    DCNop        = DONop,
    DCCnfQuote=  DOQuote|Arg1Cnf,
    DCFofQuote=  DOQuote|Arg1Fof,
    ...
    DCRewrite    = DORewrite|Arg1Cnf,
    ...
    DCParamod    = DOParamod|Arg1Cnf|Arg2Cnf,
    ...
}
```

Consistent Encodings

- DerivationCodes inspired by machine instructions
  - Operation (OpCode)
  - Arguments
- Supports flexible processing
  - Proof *structure* is independent of exact inferences

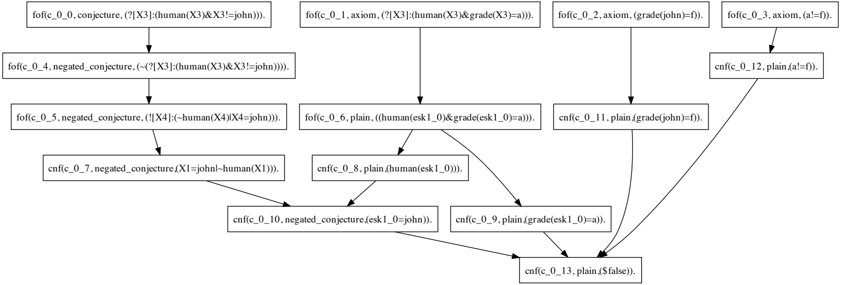
Consistent Encodings

- DerivationCodes inspired by machine instructions
  - Operation (OpCode)
  - Arguments
- Supports flexible processing
  - Proof *structure* is independent of exact inferences
  - OpCodes only needed for output

Consistent Encodings

- DerivationCodes inspired by machine instructions
  - Operation (OpCode)
  - Arguments
- Supports flexible processing
  - Proof *structure* is independent of exact inferences
  - OpCodes only needed for output
  - . . . and exact proof reconstruction

Proofs as Graphs



Consistent Encodings

- DerivationCodes inspired by machine instructions
  - Operation (OpCode)
  - Arguments
- Supports flexible processing
  - Proof *structure* is independent of exact inferences
  - OpCodes only needed for output

Consistent Encodings

- DerivationCodes inspired by machine instructions
  - Operation (OpCode)
  - Arguments
- Supports flexible processing
  - Proof *structure* is independent of exact inferences
  - OpCodes only needed for output
  - ...and exact proof reconstruction

Consistent Encodings

- DerivationCodes inspired by machine instructions
  - Operation (OpCode)
  - Arguments
- Supports flexible processing
  - Proof *structure* is independent of exact inferences
  - OpCodes only needed for output
  - . . . and exact proof reconstruction

Derivation stacks allow a compact and efficient (in data and code) representation of clause history.

Challenges

Unambiguous Inferences

- Complete inference records
  - Add inference positions (more arguments to OpCodes)
  - Add unifiers (if neccessary, e.g. HO)
  - ...
- Complete clausification records
  - Clause simplification as rewriting (?)
  - Mini-scoping as rewriting (?)
  - Step-by-step skolemization

Theoretically managable, but practically difficult – especially retroactively

Proof Expansion

- Calculus level expansion
  - Explicit results of each inference
  - Good for semantic proof checking
  - Good for understanding the structure of the proof
  - Potentially good for machine learning
- Primitive inferences
  - Convert inferences into primitive operations
  - For superposition:
    - Instantiation
    - Lazy conditional term replacement
    - Deleting trivial and duplicated literals
  - Uniform proof format for different provers/calculi
  - Uniform post-processing (proof checking, proof presentation, . . .)

Conclusion

Conclusion

- Efficient proof generation is non-trivial, but possible
- Clause objects and derivation stacks can represent internal inference sequences compactly
- TPTP v3 is a useful and used standard for external 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
- Clause objects and derivation stacks can represent internal inference sequences compactly
- TPTP v3 is a useful and used standard for external proof representation
- Proof objects are useful for trust building and learning
- Use of proof objects is still in its infancy - we need more tools

Efficient and flexible proof representation is a key to useful proof output.

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`**