Where, What, and How?
Lessons from the Evolution of E

Stephan Schulz
schulz@eprover.org
Automated Theorem Proving

$A \models C$

...where

$A = \{A_1, \ldots, A_n\}$ is a set of axioms

$C$ is the conjecture

...in First-Order Logic with Equality
E

is a
fully automatic
Theorem Prover
for
First-Order Logic
with Equality
Stephan Schulz

From the E NEWS file:

*Sat Jul 5 02:28:25 MET DST 1997: First line of code written (in BASICS/clb_defines.h). StS*
First place in MIX at CASC 17

First place in CNF at CASC 23
What inference system to use?

How to do inferences efficiently?

Where to search for proofs?

Calculus

Implementation

Search Control
A Virtual Tour in Time

E has been under development since 1997

- ≈15 years of ATP history
- Mostly one developer

(Mostly) conservative extensions

- New features have been added to the core
- New features can be activated/deactivated

Non-conservative changes

- Scaleability
- Robustness
- Improvements to basic data types

We can simulate many aspects of old versions of E
Strengths and Limitations

Simulated:
- Calculus
- Search heuristics
- Many alternative algorithms
- Scope/language
- Usability

Not simulated:
- Robustness issues
- Most scaleability features
- Bugs!
Benefits

Historical situation
- Interleaved evolution of features
- Only major steps published

Simulation supports isolation of variables
- Implementation features
- Calculus modifications
- Search control
Agenda

Introduction
Scope and Usability
Implementation
Calculus evolution
Search control
Conclusion
Scope and Usability
Stephan Schulz

1.8

Parser

Relevancy Pruning

Clausification

Clausal Preprocessing

Saturation

Proof Extraction

User

Raw Analysis

CNF Analysis
Calculus

Implementation

Search

Control
Scope

Usability

Robustness & Scalability
Clausification and FOF support

Automatic Auto-Mode

Automatic Strategy Scheduling
Full FOF and Clausification

Historical
- First-order = CNF
- Proving = Showing unsatisfiability

Clausification in E
- E 0.82 (2004): Original “naive” clausifier
- E 0.91 (2006): Clausifier with definitions

Implementation
- Based on Nonnengart/Weidenbach: *Computing Small Clause Normal Forms*, 2001
- Shared formulas
- Shared definition
Automatic Modes

Common properties:

- Analyze problem
- Determine problem class
- Pick strategy or strategies
- Automatically generated from test data

E 0.5 (1999): Auto-Mode
- Pick single best strategy for class

E 1.8 (2013): Auto-Schedule
- Simple portfolio approach
- Try 5 strategies with fixed time allocation
- Greedy schedule generation
Implementation
“Who controls the present controls the past”

Early E: Undeserved reputation for speed
   ▶ ... written in C (?)
   ▶ ... convenient explanation for performance (?)
“Who controls the present controls the past”

Early E: Undeserved reputation for speed
► ... written in C (?)
► ... convenient explanation for performance (?)

Countermeasures

Top-down: Tarnish that reputation
► E: A Brainiac Theorem Prover

Bottom-up: Justify that reputation
► Löchner’s Linear KBO/Polynomial LPO
► Feature Vector Indexing (subsumption)
► Fingerprint Indexing (rewriting and superposition)
Calculus

Superposition calculus (evolved from [BG94])

- Refutational calculus
- Proof state: Set of clauses
- Goal: Derive empty clause
- Method: Saturation up to redundancy
What is a clause?

Multi-set of equational literals
- \{f(X) \neq a, P(a) \neq \text{true}, g(Y) = f(a)\}

Disjunction of literals
- \(f(X) \neq a \lor \neg P(a) \lor g(Y) = f(a)\)

Conditional rewrite-rule
- \(f(X) = a \land P(a) \implies g(Y) = f(a)\)

Special clauses
- The empty clause \(\square = \{\}\) is unsatisfiable
- Unit clauses \(s = t\) are potential rewrite rules
Inferences

Generating inferences

- 1-2 premises generate new clause
- Superposition, equality resolution, equality factoring

Necessary evil

Contracting/simplifying inferences

- Replace or remove main premise
- Rewriting, subsumption, ...

Expensive, but well worth it
The Given-Clause Algorithm

Aim: Move everything from $U$ to $P$

$P$ (processed clauses)

$U$ (unprocessed clauses)

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
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 $P$**
Gene
rate
Simplifiable?
Simplify
P
(processed clauses)

U
(unprocessed clauses)
Given-Clause Loop

while $U \neq \{\}$
    $g = \text{delete\_best}(U)$
    $g = \text{simplify}(g, P)$
    if $g == \Box$
        SUCCESS, Proof found
    if $g$ is not subsumed by any clause in $P$ (or otherwise redundant w.r.t. $P$)
        $P = P \setminus \{c \in P \mid c \text{ subsumed by (or otherwise redundant w.r.t.) } g\}$
        $T = \{c \in P \mid c \text{ can be simplified with } g\}$
        $P = (P \setminus T) \cup \{g\}$
        $T = T \cup \text{generate}(g, P)$
    foreach $c \in T$
        $c = \text{cheap\_simplify}(c, P)$
        if $c$ is not trivial
            $U = U \cup \{c\}$
    SUCCESS, original $U$ is satisfiable
while $U \neq \{}$

$g = \text{delete\_best}(U)$

$g = \text{simplify}(g, P)$

if $g == \square$

SUCCESS, Proof found

if $g$ is not redundant w.r.t. $P$

$P = P \setminus \{c \in P \mid c \text{ redundant w.r.t. } g\}$

$T = \{c \in P \mid c \text{ simplifiable with } g\}$

$P = (P \setminus T) \cup \{g\}$

$T = T \cup \text{generate}(g, P)$

foreach $c \in T$

$c = \text{cheap\_simplify}(c, P)$

if $c$ is not trivial

$U = U \cup \{c\}$

SUCCESS, original $U$ is satisfiable
while $U \neq \{\}$
    $g = \text{delete\_best}(U)$
    $g = \text{simplify}(g, P)$
    if $g == \Box$
        SUCCESS, Proof found
    if $g$ is not redundant w.r.t. $P$
        $P = P \setminus \{c \in P \mid c \text{ redundant w.r.t. } g\}$
        $T = \{c \in P \mid c \text{ simplifiable with } g\}$
        $P = (P \setminus T) \cup \{g\}$
        $T = T \cup \text{generate}(g, P)$
    foreach $c \in T$
        $c = \text{cheap\_simplify}(c, P)$
        if $c$ is not trivial
            $U = U \cup \{c\}$
        SUCCESS, original $U$ is satisfiable
while $U \neq \{\}$

$g = \text{delete\_best}(U)$

$g = \text{simplify}(g, P)$

if $g == \square$

SUCCESS, Proof found

if $g$ is not redundant w.r.t. $P$

$P = P \setminus \{c \in P | c$ redundant w.r.t. $g\}$

$T = \{c \in P | c$ simplifiable with $g\}$

$P = (P \setminus T) \cup \{g\}$

$T = T \cup \text{generate}(g, P)$

foreach $c \in T$

$c = \text{cheap\_simplify}(c, P)$

if $c$ is not trivial

$U = U \cup \{c\}$

SUCCESS, original $U$ is satisfiable
while $U \neq \{\}$
  $g = \text{delete-best}(U)$
  $g = \text{simplify}(g, P)$
  if $g == \square$
    SUCCESS, Proof found
  if $g$ is not redundant w.r.t. $P$
    $P = P \setminus \{c \in P \mid c \text{ redundant w.r.t. } g\}$
    $T = \{c \in P \mid c \text{ simplifiable with } g\}$
    $P = (P \setminus T) \cup \{g\}$
    $T = T \cup \text{generate}(g, P)$
  foreach $c \in T$
    $c = \text{cheap-simplify}(c, P)$
    if $c$ is not trivial
      $U = U \cup \{c\}$
  SUCCESS, original $U$ is satisfiable
while $U \neq \{\}$

$g = \text{delete\_best}(U)$

$g = \text{simplify}(g, P)$

if $g == \square$

SUCCESS, Proof found

if $g$ is not redundant w.r.t. $P$

$P = P \\{c \in P \mid c \text{ redundant w.r.t. } g\}$

$T = \{c \in P \mid c \text{ simplifiable with } g\}$

$P = (P \\ T) \cup \{g\}$

$T = T \cup \text{generate}(g, P)$

foreach $c \in T$

$c = \text{cheap\_simplify}(c, P)$

if $c$ is not trivial

$U = U \cup \{c\}$

SUCCESS, original $U$ is satisfiable
while $U \neq \{\}$
  $g = \text{delete\_best}(U)$
  $g = \text{simplify}(g, P)$
  if $g == \Box$
    SUCCESS, Proof found
  if $g$ is not redundant w.r.t. $P$
    $P = P \setminus \{c \in P \mid c \text{ redundant w.r.t. } g\}$
    $T = \{c \in P \mid c \text{ simplifiable with } g\}$
    $P = (P \setminus T) \cup \{g\}$
    $T = T \cup \text{generate}(g, P)$
    foreach $c \in T$
      $c = \text{cheap\_simplify}(c, P)$
      if $c$ is not trivial
      $U = U \cup \{c\}$
    SUCCESS, original $U$ is satisfiable
while $U \neq \{\}$

$g = \text{delete\_best}(U)$

$g = \text{simplify}(g, P)$

if $g == \square$

SUCCESS, Proof found

if $g$ is not redundant w.r.t. $P$

$P = P \setminus \{c \in P \mid c \text{ redundant w.r.t. } g\}$

$T = \{c \in P \mid c \text{ simplifiable with } g\}$

$P = (P \setminus T) \cup \{g\}$

$T = T \cup \text{generate}(g, P)$

foreach $c \in T$

$c = \text{cheap\_simplify}(c, P)$

if $c$ is not trivial

$U = U \cup \{c\}$

SUCCESS, original $U$ is satisfiable
Speed Demon

Run times "E 0.2 FAST"

Run times "E 0.2"
Speed Demon tamed (?)

- "E 0.2 FOF"
- E 0.2 FOF Fast
Speed Demon tamed (?)

![Graph showing the performance of different models: E 1.8 Best, "E 0.2 FOF", and E 0.2 FOF Fast. The graph compares the models' performance across a range of values.](image-url)
On the other Hand

Run times "E 1.8 Best"

Run times "E 1.8 Best Slow"
Some Vindication

![Graph showing two lines. The top line is labeled 'E 1.8 Best' and the bottom line is labeled 'E 1.8 Slow'. The x-axis is labeled with values 0 to 300, and the y-axis is labeled with values from 0 to 10000. The graph shows the performance comparison between the two lines.]
Calculus evolution
Simultaneous superposition
Contextual Simplify-Reflect = Subsumption Resolution = Clause simplification = ...

Destructive equality resolution
AC redundancy elimination

E 1.8
E 1.5
E 1.4
E 1.2
E 1.0
E 0.9
E 0.8
E 0.82
E 0.7
E 0.61
E 0.6
E 0.5
E 0.4
E 0.3, 0.31, 0.5
E 0.2
E 0.1
E 0.0
Calculus evolution alone
Search control
Clause selection

while $U \neq \{\}$

$g = \text{delete_best}(U)$

$g = \text{simplify}(g, P)$

if $g == \square$

SUCCESS, Proof found

if $g$ is not redundant w.r.t. $P$

$P = P \setminus \{c \in P \mid c \text{ redundant w.r.t. } g\}$

$T = \{c \in P \mid c \text{ simplifiable with } g\}$

$P = (P \setminus T) \cup \{g\}$

$T = T \cup \text{generate}(g, P)$

foreach $c \in T$

$c = \text{cheap_simplify}(c, P)$

if $c$ is not trivial

$U = U \cup \{c\}$

SUCCESS, original $U$ is satisfiable
Basic Approaches

Symbol counting
- Pick smallest clause in \( P \)
- \(|\{f(X) \neq a, P(a) \neq \text{true}, g(Y) = f(a)\}| = 10\)

FIFO
- Always pick oldest clause in \( P \)

Flexible weighting
- Symbol counting, but give different weight to different symbols
- E.g. lower weight to symbols from goal!

Combinations
- Interleave different schemes
Influences on E

DISCOUNT
- Different *experts* (heuristic evaluation functions)
- Only one *expert* per saturation phase

Otter
- Interleaves size/age selection
- Larry Wos: "*The optimal pick-given ration is 5*"

Waldmeister
- Larry is right in general, wrong in detail
The Second System Effect

The general tendency is to over-design the second system, using all the ideas and frills that were cautiously sidetracked on the first one. The result, as Ovid says, is a “big pile.”

— Frederick P. Brooks, Jr.
Given-Clause Selection in E

Domain Specific Language (DSL) for clause selection scheme
Arbitrary number of queues
Each queue ordered by:
  ▶  Unparameterized priority function
  ▶  Parameterized heuristic evaluation function
Clauses picked using weighted round-robin scheme
  ▶  Example:
    ▶  4 clauses from queue 1
    ▶  2 clauses from queue 2
    ▶  2 clauses from queue 3
    ▶  Start over at queue 1

Second-system effect gone wild
Clause selection DSL

First goal-directed clause selection

Stephan Schulz
The Influence of Clause Selection

![Graph showing the influence of clause selection with different lines representing various goals and strategies.](image-url)
The Influence of Clause Selection

![Graph showing the influence of clause selection]

- E 1.8 Best
- E 0.2 Goals
- E 0.2 Larry
- E 0.2 FOF
- E 0.2 SC
Literal Selection

Literal selection in superposition:
- In clauses with negative literals, pick any single negative literal
- Only this selected literal is used for inferences
- Otherwise, all maximal literals are used

Intuition:
- \( f(X) = a \land P(a) \implies g(Y) = f(a) \)
- We need to solve all conditions before the implication becomes relevant
- So start with any one condition...
Anonymous Reviewers
Literal Selection in E

Ca. umpteen hard-coded strategies

Example 1: SelectSmallestNegLit
  ▶ Always select the smallest literal
  ▶ Idea: Fewer inferences possible

SelectMaxLComplexAvoidPosPred
  ▶ Select, in the following order:
    ▶ Maximal, pure variable (X \neq Y)
    ▶ Maximal, ground, largest size difference
    ▶ Maximal, non-ground, largest difference
    ▶ Pure variable
    ▶ Ground, largest size difference
    ▶ Non-ground, largest difference
    ▶ \ldots all things being equal, avoid predicates from positive literals
The Influence of Literal Selection

- E 0.2 SmallestNegLit
- E 0.2 MaxLComplexAvoidPosPred
- E 0.2 FOF
The Influence of Literal Selection

E 1.8 Best
E 0.2 SmallestNegLit
E 0.2 MaxLComplexAvoidPosPred
E 0.2 FOF

54
Conclusion
Conclusion

E’s core progress has been due to

- Primarily **search control**
- Secondarily calculus and implementation

Significant interplay between

- Calculus and implementation
- Literal selection and term orderings

Users profit from usability and scope

- Full automation (including parameterization)
- Support for rich(er) logics
Some Open Points

Understand literal selection
- What makes a good strategy?
- Interaction of literal selection and ordering

Proof search
- Improve goal-directed search
- Better meta-control ("Auto-Mode")

Can big-data approaches help?
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`