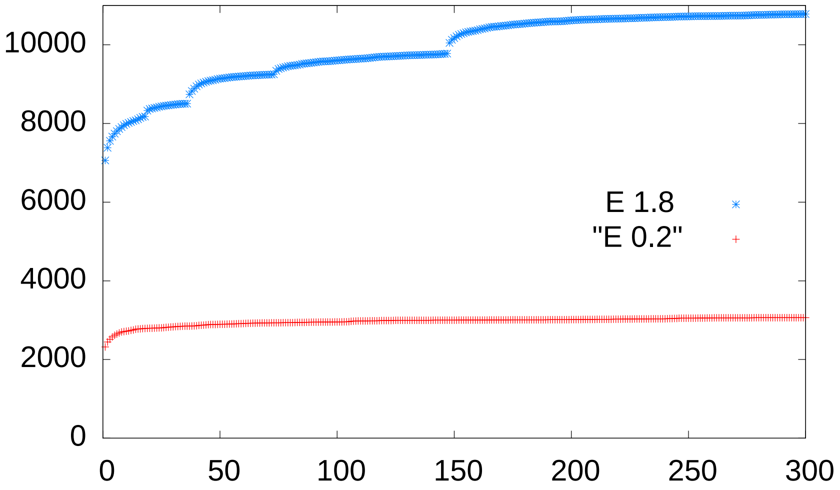
Where, What, and How?

Lessons from the Evolution of E

Stephan Schulz

`schulz@eprover.org`


10000
8000
6000
4000
2000
0
0
50
100
150
200
250
300
E 1.8
"E 0.2"

Automated Theorem Proving

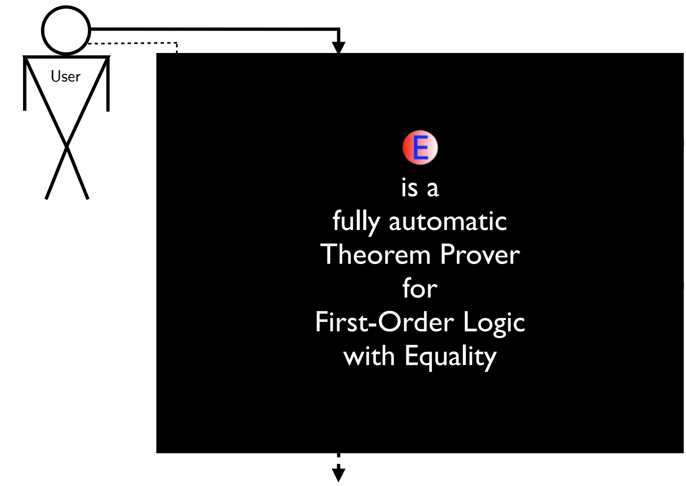
$$A \overset{?}{\models} C$$

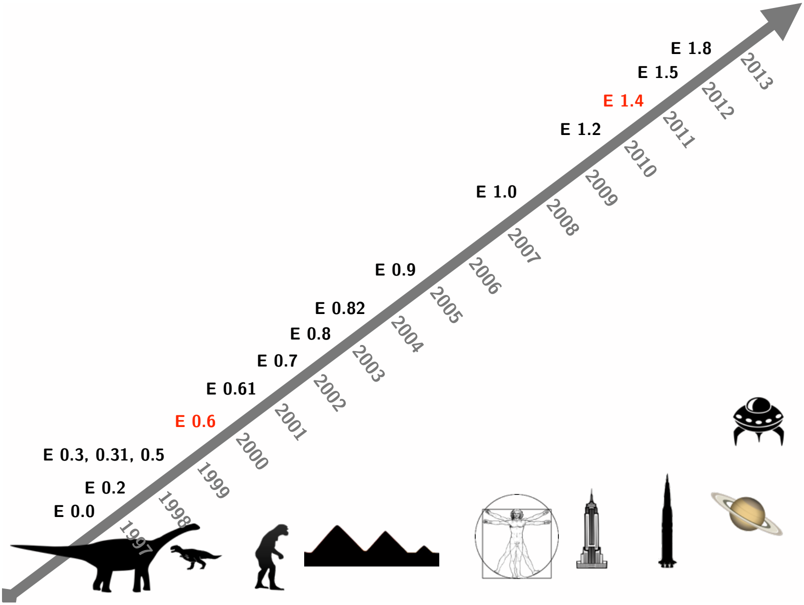
…where

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

…in First-Order Logic with Equality

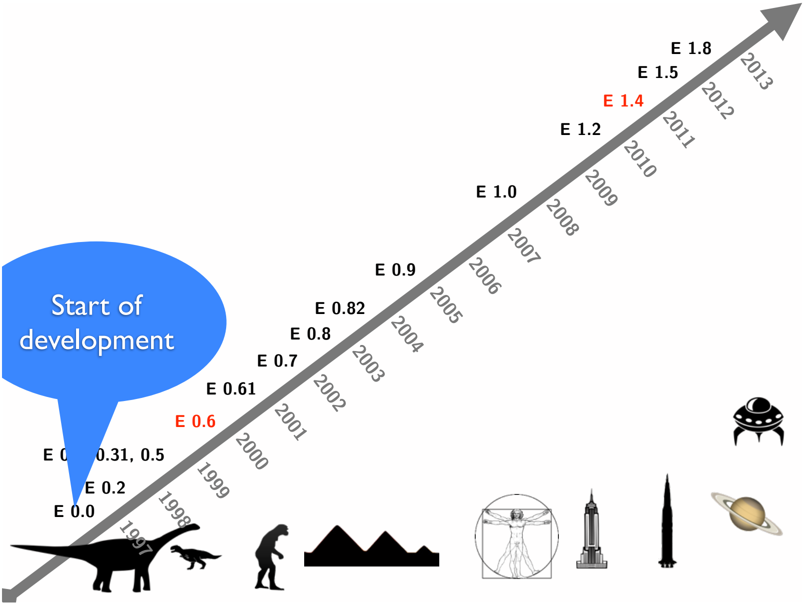
E

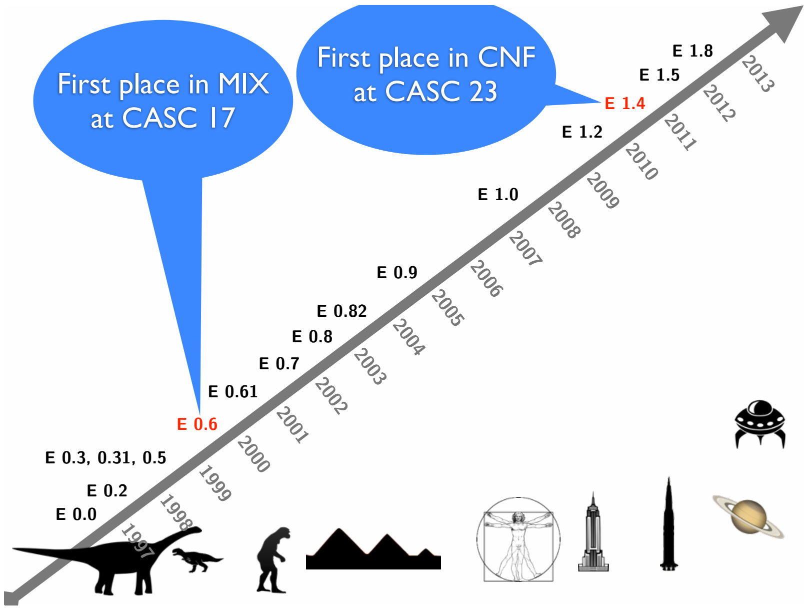

User
E
is a
fully automatic
Theorem Prover
for
First-Order Logic
with Equality

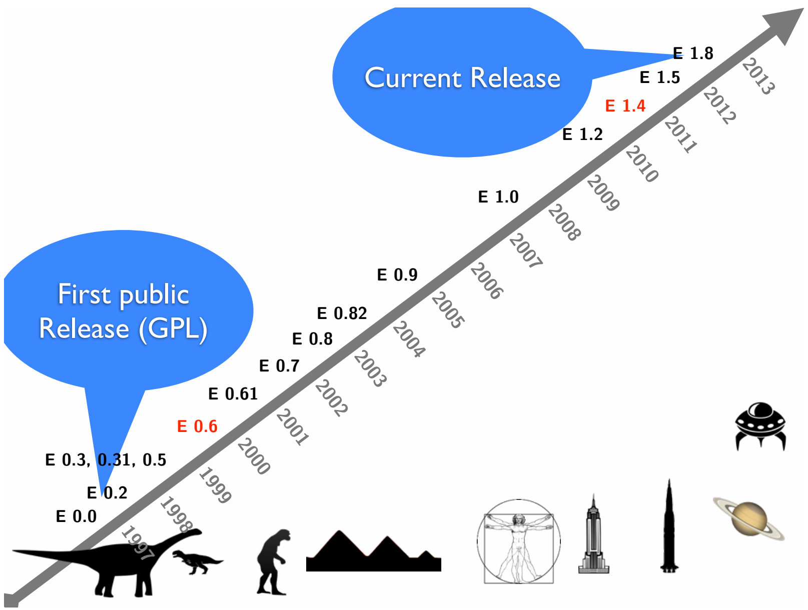

E 0.0
E 0.2
E 0.3, 0.31, 0.5
E 0.6
E 0.61
E 0.7
E 0.8
E 0.82
E 0.9
E 1.0
E 1.2
E 1.4
E 1.5
E 1.8
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013

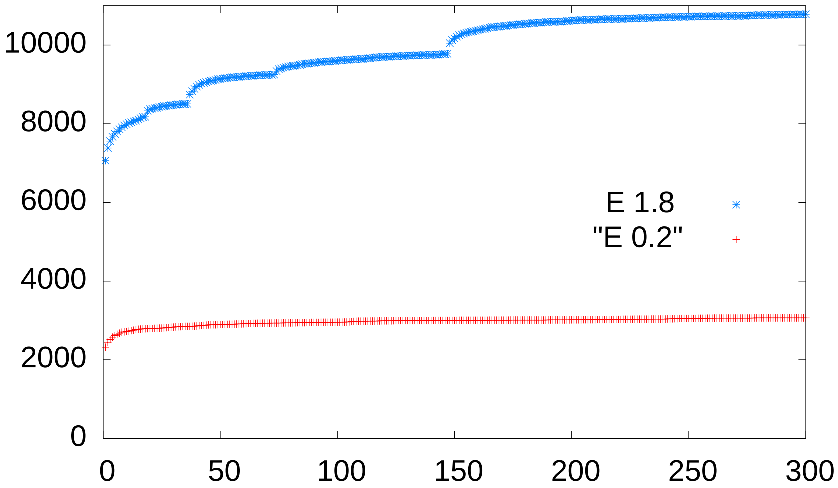
From the E NEWS file:

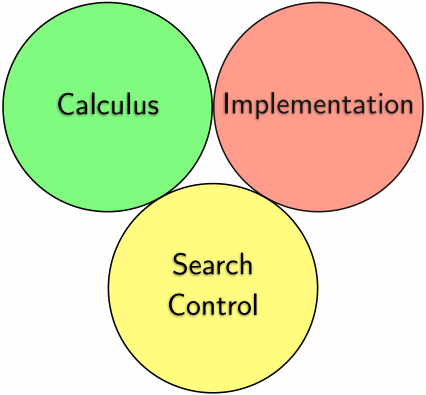
Sat Jul 5 02:28:25 MET DST 1997: First line of code written (in BASICS/clb_defines.h). StS


Start of development
E 0.0
E 0.2
E 0
0.31, 0.5
E 0.6
E 0.61
E 0.7
E 0.8
E 0.82
E 0.9
E 1.0
E 1.2
E 1.4
E 1.5
E 1.8
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013


First place in MIX at CASC 17
First place in CNF at CASC 23
E 0.0
E 0.2
E 0.3, 0.31, 0.5
E 0.6
E 0.61
E 0.7
E 0.8
E 0.82
E 0.9
E 1.0
E 1.2
E 1.4
E 1.5
E 1.8
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013

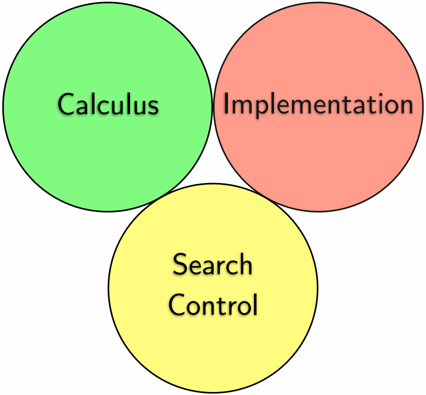

Current Release
First public
Release (GPL)
E 0.0
E 0.2
E 0.3, 0.31, 0.5
E 0.6
E 0.61
E 0.7
E 0.8
E 0.82
E 0.9
E 1.0
E 1.2
E 1.4
E 1.5
E 1.8
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013


10000
8000
6000
4000
2000
0
0
50
100
150
200
250
300
E 1.8
"E 0.2"


Calculus
Implementation
Search
Control

What inference system to use?

How to do inferences efficiently?



Where to search for proofs?

A Virtual Tour in Time

E has been under development since 1997

- $\approx$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

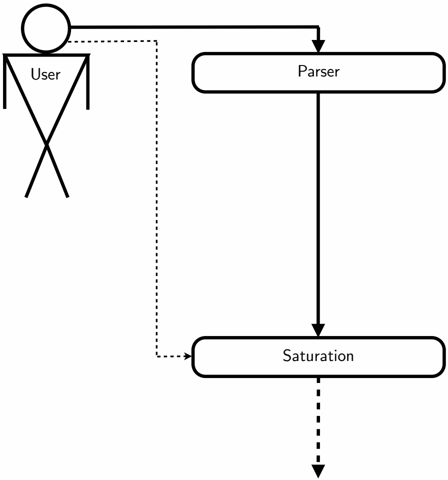
Calculus evolution

Search control

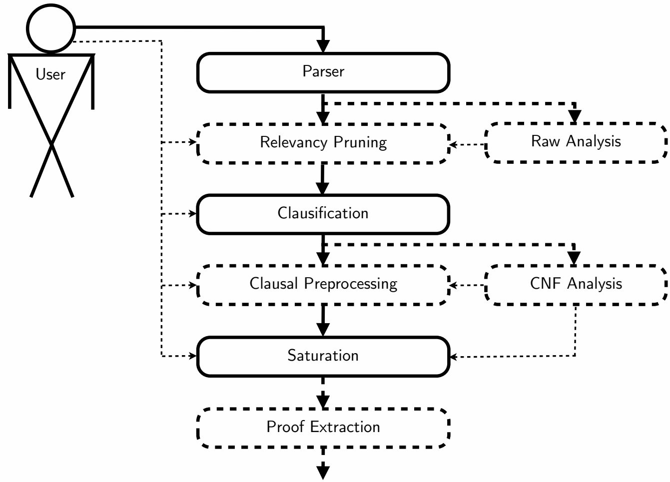
Conclusion

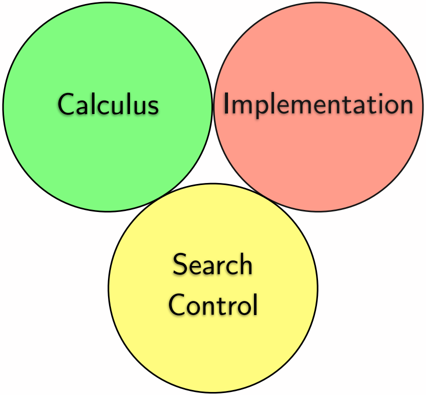
Scope and Usability

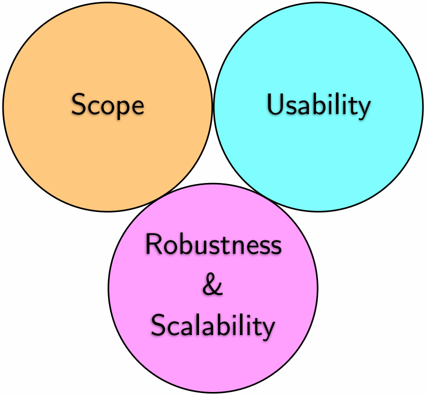
E 0.2

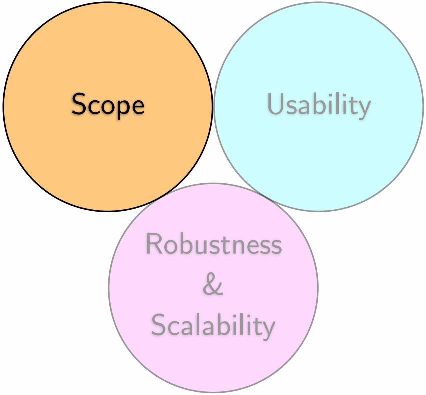


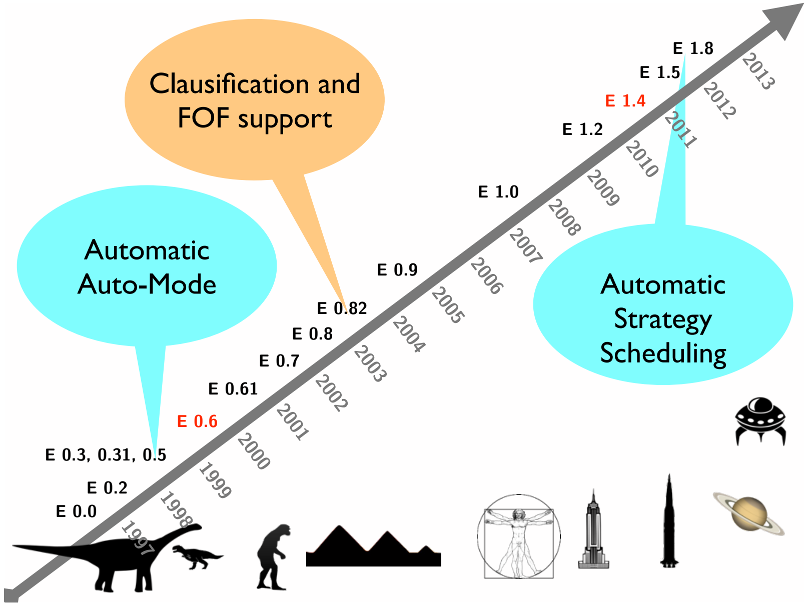
E 1.8


User
Parser
Relevancy Pruning
Raw Analysis
Clausification
Clausal Preprocessing
CNF Analysis
Saturation
Proof Extraction


Calculus
Implementation
Search
Control


Scope
Usability
Robustness
&
Scalability


Scope
Usability
Robustness
&
Scalability



Clausification and FOF support
Automatic Auto-Mode
Automatic Strategy Scheduling
E 0.0
E 0.2
E 0.3, 0.31, 0.5
E 0.6
E 0.61
E 0.7
E 0.8
E 0.82
E 0.9
E 1.0
E 1.2
E 1.4
E 1.5
E 1.8
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013

Full FOF and Clausification

Historical

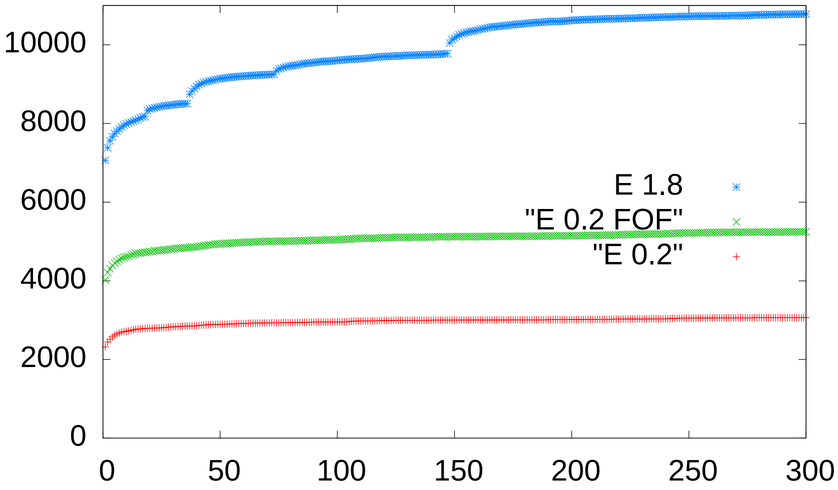
- First-order = CNF
- Proving = Showing unsatisfiability

Clausification in E

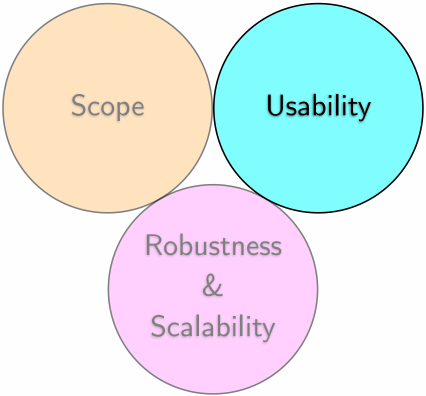
- E 0.82 (2004): Original “naive” clausifier
- E 0.91 (2006): Clausifier with definitions

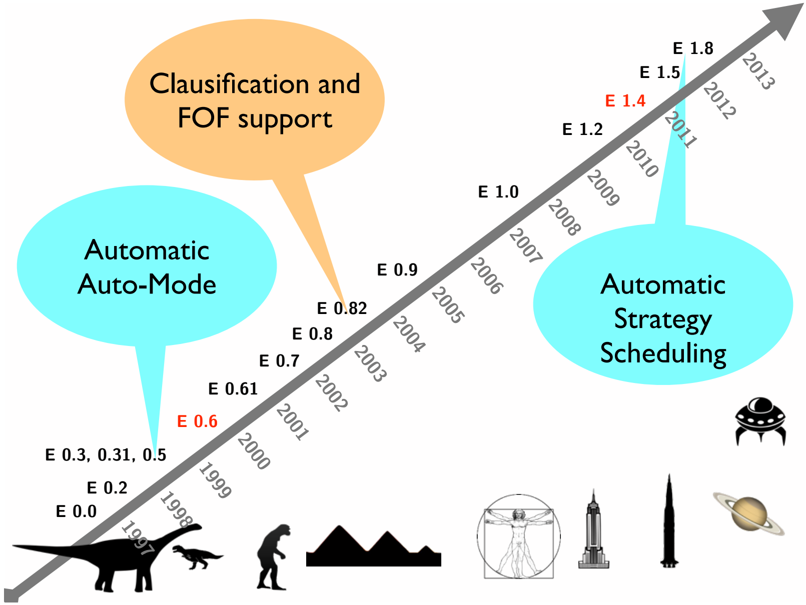
Implementation

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



10000
8000
6000
4000
2000
0
0
50
100
150
200
250
300
E 1.8
"E 0.2 FOF"
"E 0.2"


Scope
Usability
Robustness
&
Scalability


Clausification and
FOF support
Automatic
Auto-Mode
Automatic
Strategy
Scheduling
E 0.0
E 0.2
E 0.3, 0.31, 0.5
E 0.6
E 0.61
E 0.7
E 0.8
E 0.82
E 0.9
E 1.0
E 1.2
E 1.4
E 1.5
E 1.8
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013

Automatic Modes

Common properties:

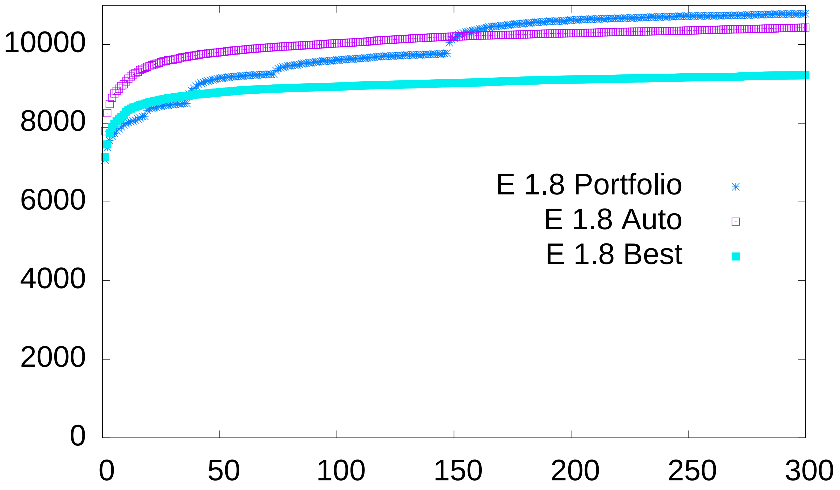
- Analyze problem
- Determine problem class
- Pick strategy or strategies
- Automatically generated from test data

E 0.5 (1999): Auto-Mode

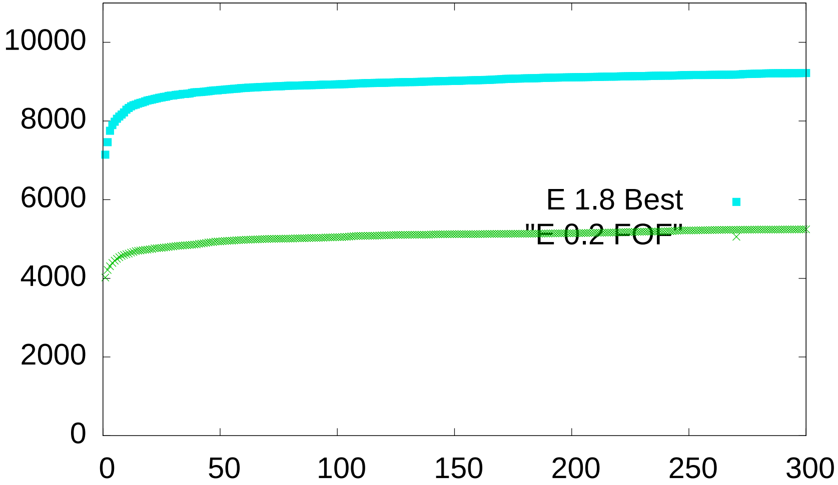
- Pick single best strategy for class

E 1.8 (2013): Auto-Schedule

- Simple portfolio approach
- Try 5 strategies with fixed time allocation
- Greedy schedule generation



10000
8000
6000
4000
2000
0
0
50
100
150
200
250
300
E 1.8 Portfolio
E 1.8 Auto
E 1.8 Best


10000
8000
6000
4000
2000
0
0
50
100
150
200
250
300
E 1.8 Best
"E 0.2 FOF"

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

- $\{f(X) \neq a, P(a) \neq \$true, g(Y) = f(a)\}$

Disjunction of literals

- $f(X) \neq a \vee \neg P(a) \vee g(Y) = f(a)$

Conditional rewrite-rule

- $f(X) = a \wedge P(a) \Longrightarrow g(Y) = f(a)$

Special clauses

- The empty clause $\Box = \{\}$ is unsatisfiable
- Unit clauses $s = t$ are potential rewrite rules

Inferences

Generating inferences

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

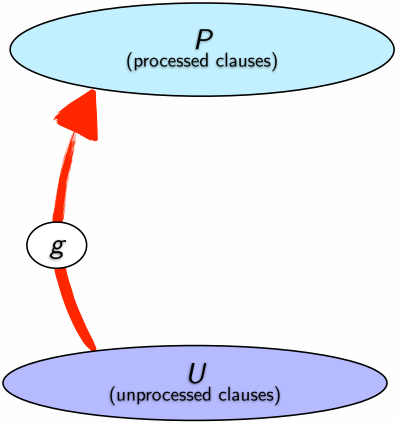
Necessary evil

Contracting/simplifying inferences

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

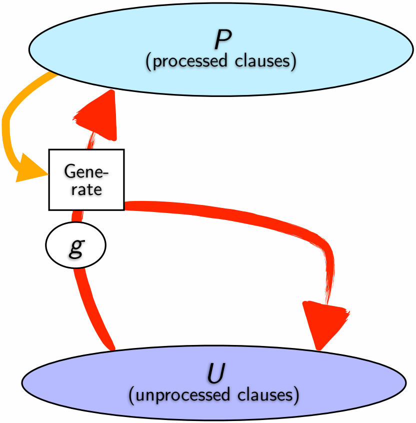
Expensive, but well worth it

The Given-Clause Algorithm



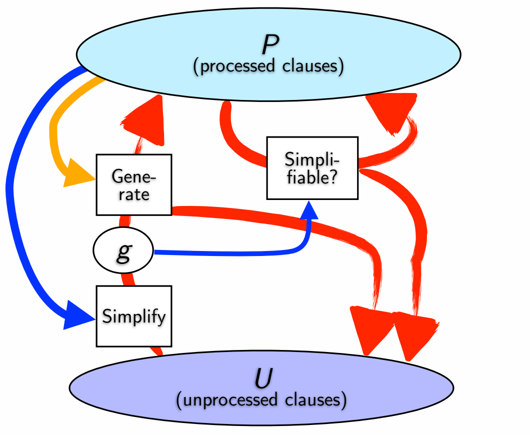
- Aim: Move everything from U to P

The Given-Clause Algorithm



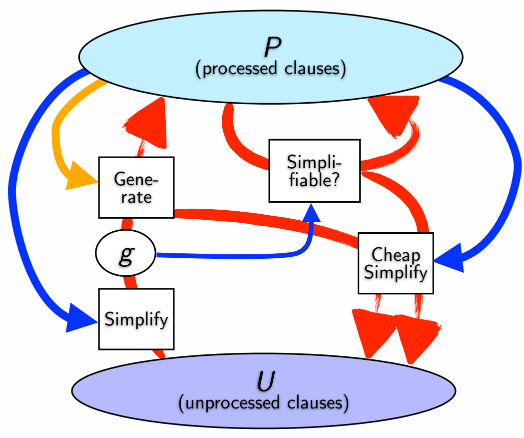
- Aim: Move everything from U to P
- Invariant: All generating inferences with premises from P have been performed

The Given-Clause Algorithm

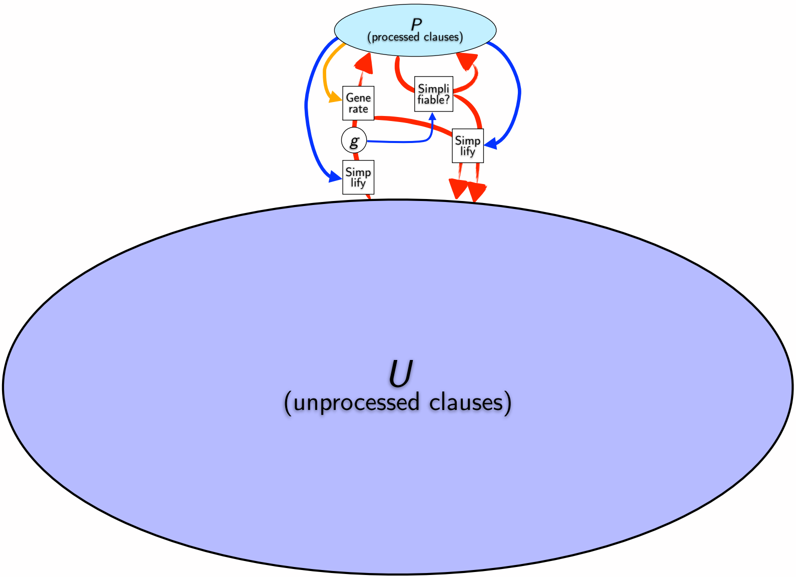


- Aim: Move everything from U to P
- Invariant: All generating inferences with premises from P have been performed
- Invariant: P is interreduced

The Given-Clause Algorithm



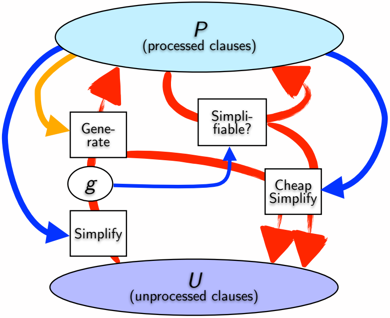
- Aim: Move everything from U to P
- Invariant: All generating inferences with premises from P have been performed
- Invariant: P is interreduced
- Clauses added to U are simplified with respect to P


P
(processed clauses)
Gene
rate
g
Simp
lify
Simpli
fiable?
Simp
lify
U
(unprocessed clauses)

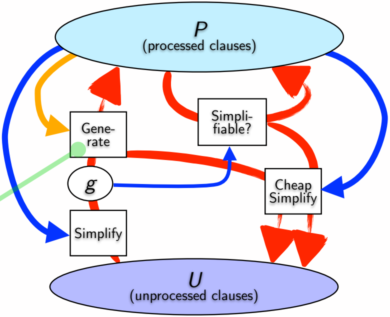
Given-Clause Loop

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)
    $P = P \backslash \{c \in P \mid c$ subsumed by (or otherwise redundant w.r.t.) $g\}$
    $T = \{c \in P \mid c$ can be simplified with $g\}$
    $P = (P \backslash T) \cup \{g\}$
    $T = T \cup \text{generate}(g, P)$
    foreach $c \in T$
      $c = \text{cheap\_simplify}(c, P)$
      if c is not trivial
        $U = U \cup \{c\}$
SUCCESS, original U is satisfiable

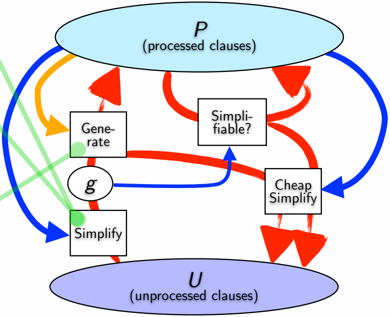
while $U \neq \{\}$
  $g = \text{delete\_best}(U)$
  $g = \text{simplify}(g, P)$
  if $g == \Box$
    SUCCESS, Proof found
  if g is not redundant w.r.t. P
    $P = P \backslash \{c \in P \mid c \text{ redundant w.r.t. } g\}$
    $T = \{c \in P \mid c \text{ simplifiable with } g\}$
    $P = (P \backslash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



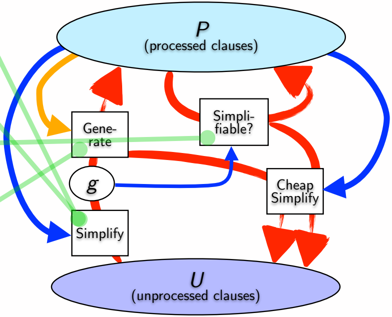
while $U \neq \{\}$
    $g = \text{delete\_best}(U)$
    $g = \text{simplify}(g, P)$
    if $g == \Box$
        SUCCESS, Proof found
    if g is not redundant w.r.t. P
        $P = P \backslash \{c \in P \mid c \text{ redundant w.r.t. } g\}$
        $T = \{c \in P \mid c \text{ simplifiable with } g\}$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



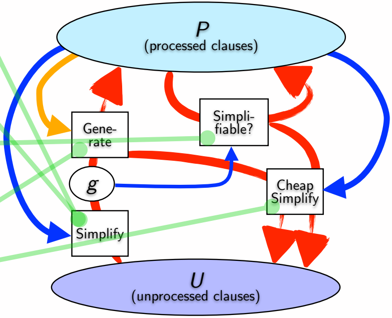
while $U \neq \{\}$
  $g = $ delete_best(U)
  $g = $ simplify(g, P)
  if $g == \Box$
    SUCCESS, Proof found
  if g is not redundant w.r.t. P
    $P = P \backslash \{c \in P \mid c$ redundant w.r.t. $g\}$
    $T = \{c \in P \mid c$ simplifiable with $g\}$
    $P = (P \backslash T) \cup \{g\}$
    $T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



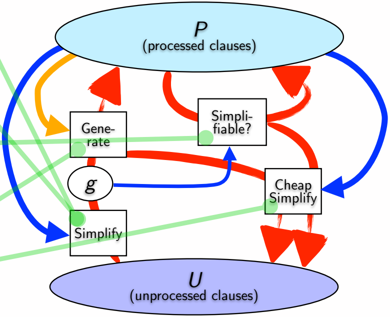
while $U \neq \{\}$
  g = delete_best(U)
  g = simplify(g, P)
  if $g == \Box$
    SUCCESS, Proof found
  if g is not redundant w.r.t. P
    $P = P \backslash \{c \in P \mid c \text{ redundant w.r.t. } g\}$
    $T = \{c \in P \mid c \text{ simplifiable with } g\}$
    $P = (P \backslash T) \cup \{g\}$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



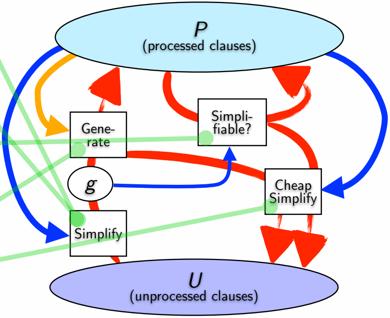
while $U \neq \{\}$
  $g = \text{delete\_best}(U)$
  $g = \text{simplify}(g, P)$
  if $g == \square$
    SUCCESS, Proof found
  if g is not redundant w.r.t. P
    $P = P \backslash \{c \in P \mid c \text{ redundant w.r.t. } g\}$
    $T = \{c \in P \mid c \text{ simplifiable with } g\}$
    $P = (P \backslash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

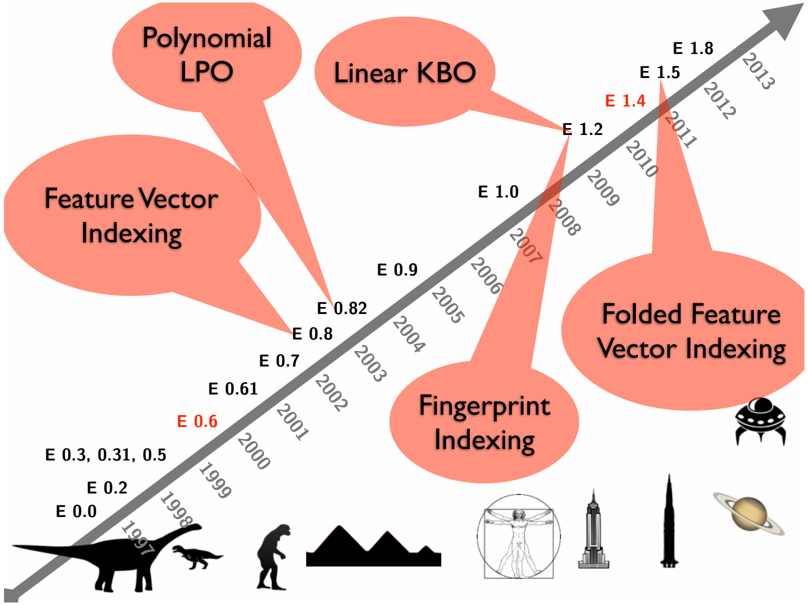


while $U \neq \{\}$
  g = delete_best(U)
  g = simplify(g, P)
  if $g == \Box$
    SUCCESS, Proof found
  if g is not redundant w.r.t. P
    $P = P \backslash \{c \in P \mid c$ redundant w.r.t. $g\}$
    $T = \{c \in P \mid c$ simplifiable with $g\}$
    $P = (P \backslash T) \cup \{g\}$
    $T = T \cup$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

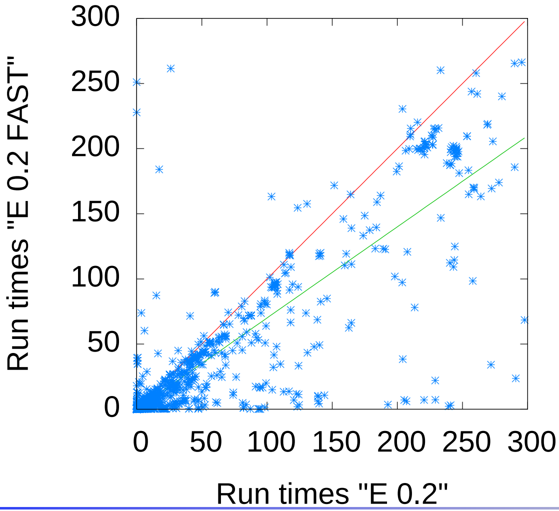


while $U \neq \{\}$
  $g = \text{delete\_best}(U)$
  $g = \text{simplify}(g, P)$
  if $g == \Box$
    SUCCESS, Proof found
  if g is not redundant w.r.t. P
    $P = P \backslash \{c \in P \mid c \text{ redundant w.r.t. } g\}$
    $T = \{c \in P \mid c \text{ simplifiable with } g\}$
    $P = (P \backslash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

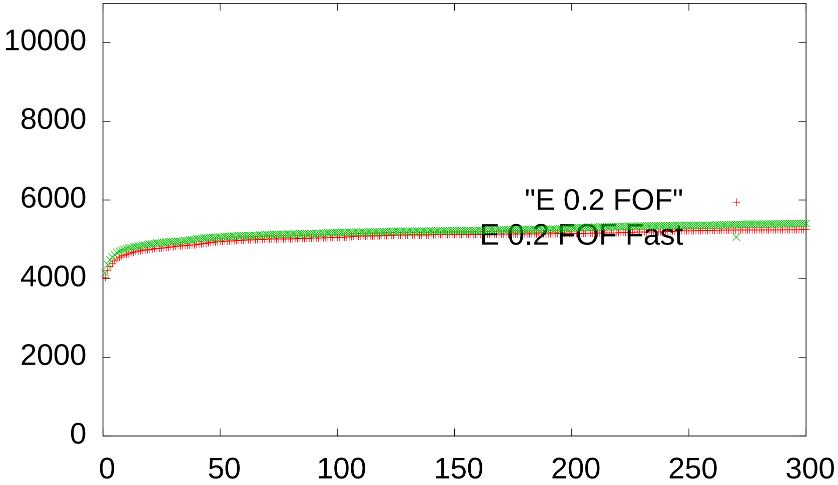



Polynomial LPO
Linear KBO
Feature Vector Indexing
Fingerprint Indexing
Folded Feature Vector Indexing
E 0.0
E 0.2
E 0.3, 0.31, 0.5
E 0.6
E 0.61
E 0.7
E 0.8
E 0.82
E 0.9
E 1.0
E 1.2
E 1.4
E 1.5
E 1.8
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013

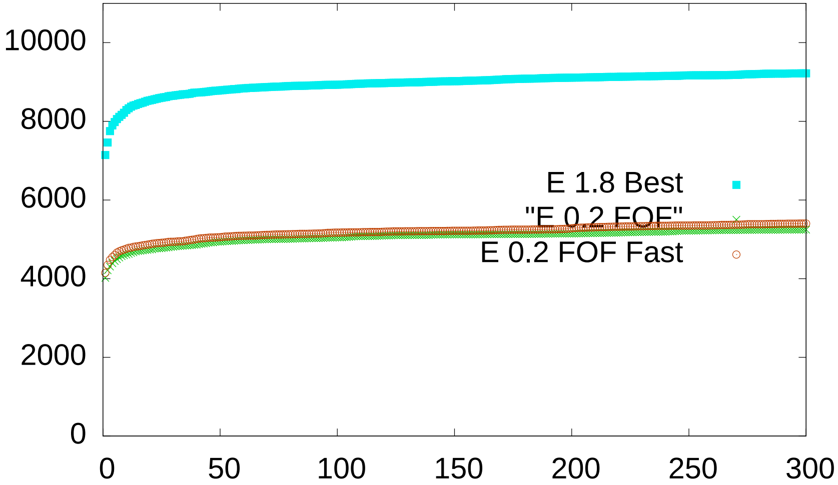
Speed Demon


300
250
200
150
100
50
0
Run times "E 0.2 FAST"
0
50
100
150
200
250
300
Run times "E 0.2"

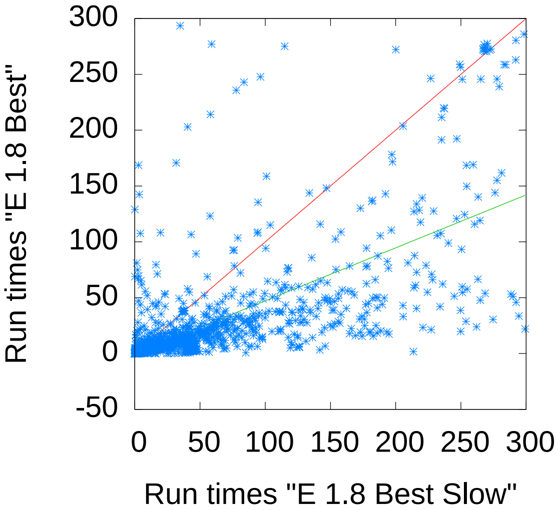
Speed Demon tamed (?)



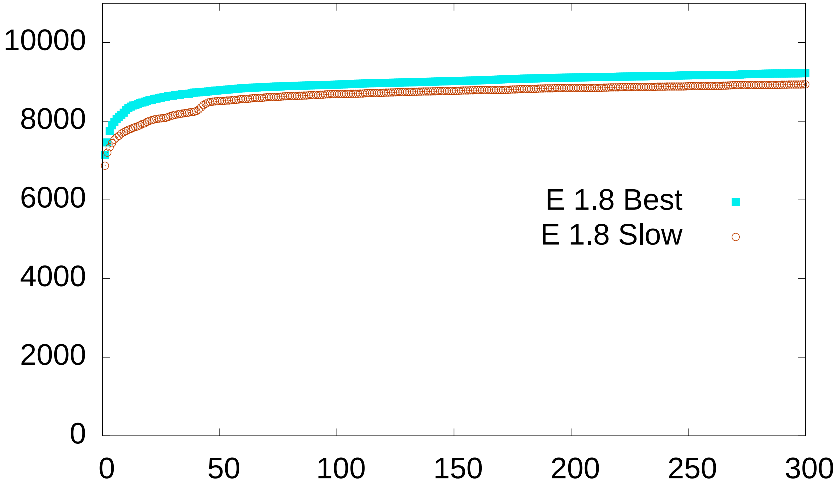
Speed Demon tamed (?)



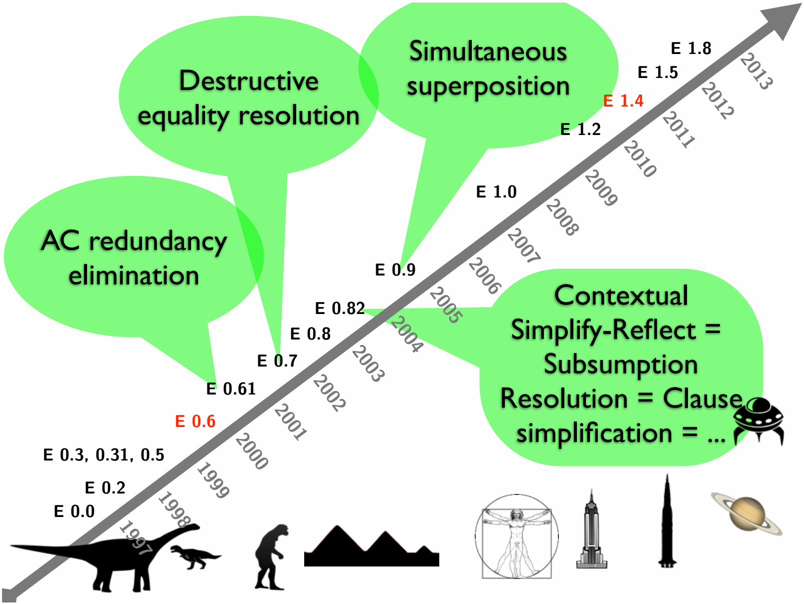
On the other Hand



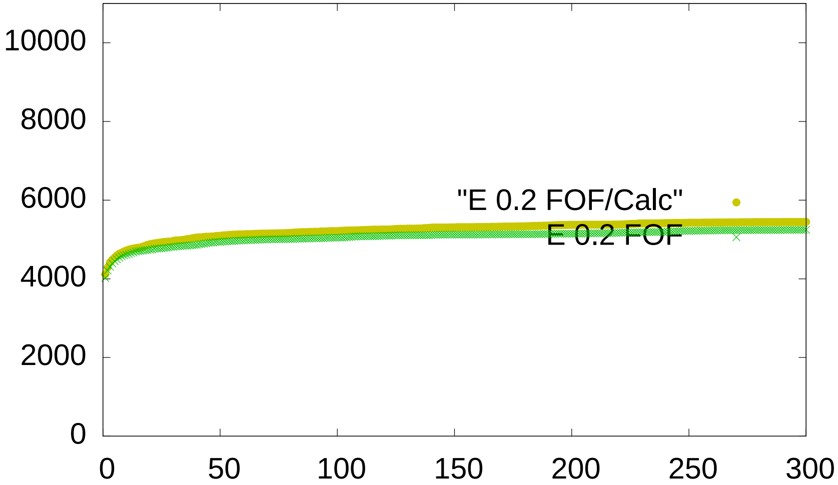
Some Vindication



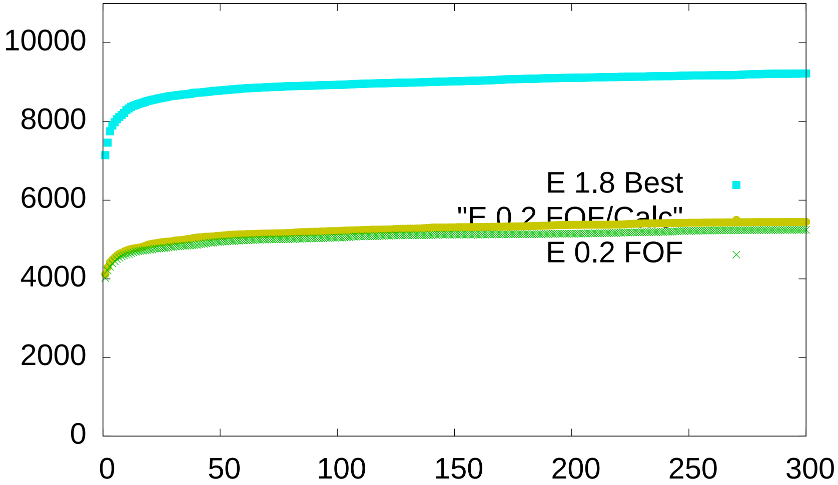
Calculus evolution


Destructive equality resolution
Simultaneous superposition
AC redundancy elimination
Contextual Simplify-Reflect = Subsumption Resolution = Clause simplification = ...
E 0.0
E 0.2
E 0.3, 0.31, 0.5
E 0.6
E 0.61
E 0.7
E 0.8
E 0.82
E 0.9
E 1.0
E 1.2
E 1.4
E 1.5
E 1.8
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013

Calculus evolution alone



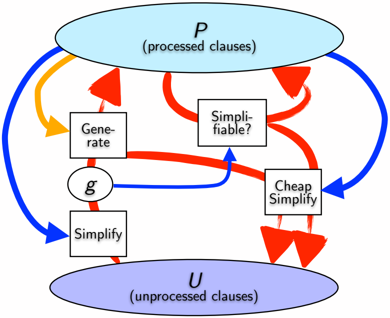
Calculus evolution alone



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
    $P = P \backslash \{c \in P \mid c \text{ redundant w.r.t. } g\}$
    $T = \{c \in P \mid c \text{ simplifiable with } g\}$
    $P = (P \backslash T) \cup \{g\}$
    $T = T \cup \text{generate}(g, P)$
    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
- $|\{f(X) \neq a, P(a) \neq \$true, g(Y) = f(a)\}| = 10$

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

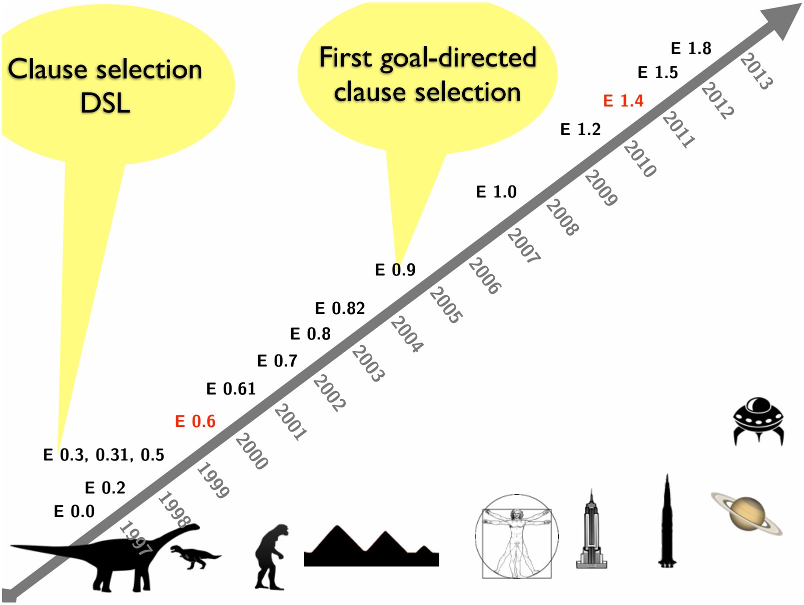
Each queue ordered by:

- Unparameterized priority function
- Parameterized heuristic evaluation function

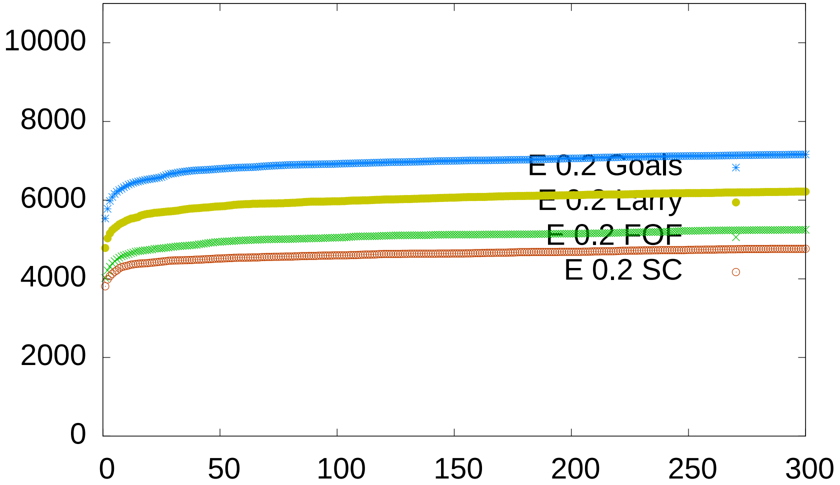
Clauses picked using weighted round-robbin scheme

- Example:
  - 4 clauses from queue 1
  - 2 clauses from queue 2
  - 2 clauses from queue 3
  - Start over at queue 1

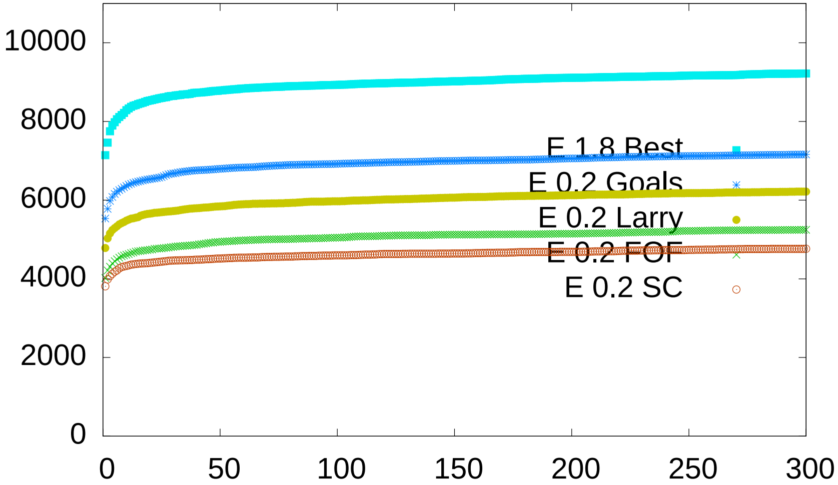
Second-system effect gone wild


Clause selection DSL
First goal-directed clause selection
E 0.0
E 0.2
E 0.3, 0.31, 0.5
E 0.6
E 0.61
E 0.7
E 0.8
E 0.82
E 0.9
E 1.0
E 1.2
E 1.4
E 1.5
E 1.8
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013

The Influence of Clause Selection



The Influence of Clause Selection



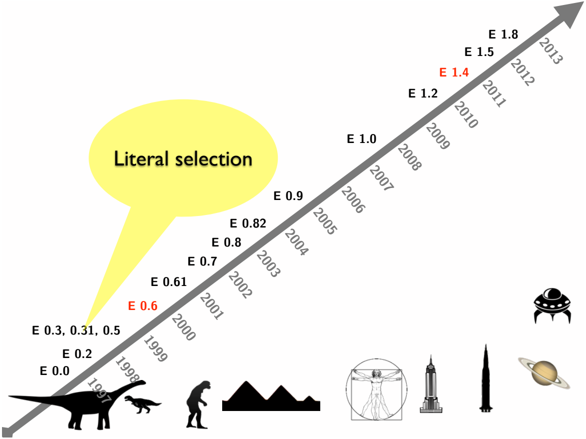
Literal Selection

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:

- $f(X) = a \wedge P(a) \Longrightarrow g(Y) = f(a)$
- We need to solve *all* conditions before the implication becomes relevant
- So start with any one condition...


Literal selection
E 1.8
E 1.5
E 1.4
E 1.2
E 1.0
E 0.9
E 0.82
E 0.8
E 0.7
E 0.61
E 0.6
E 0.3, 0.31, 0.5
E 0.2
E 0.0
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013

Anonymous Reviewers

Literal Selection in E

Ca. umpteen hard-coded strategies

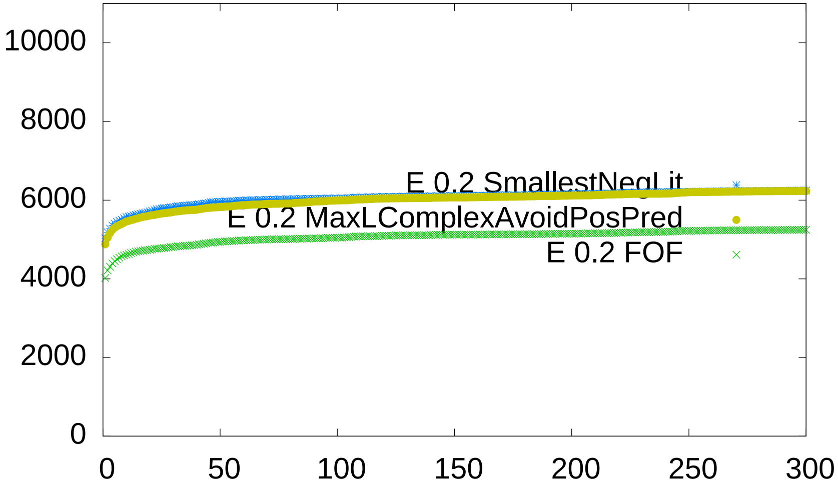
Example 1: SelectSmallestNegLit

- Always select the smallest literal
- Idea: Fewer inferences possible

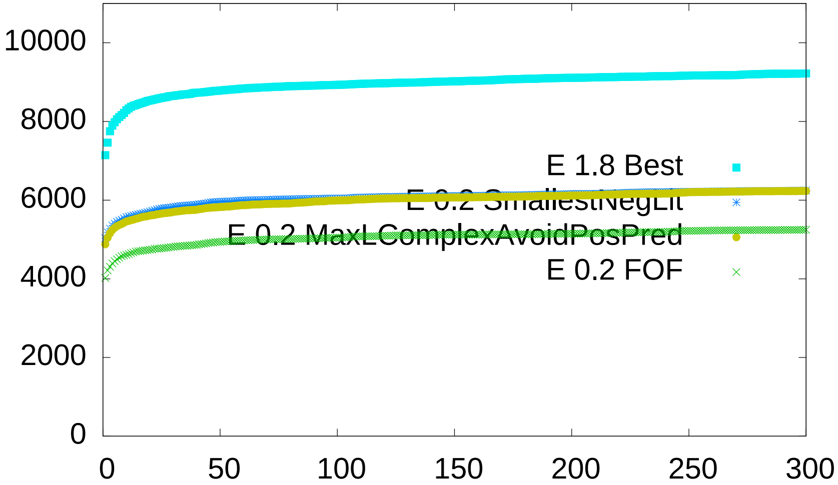
SelectMaxLComplexAvoidPosPred

- 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
  - . . . all things being equal, avoid predicates from positive literals

The Influence of Literal Selection



The Influence of Literal Selection



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

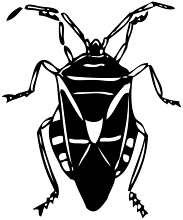
- What makes a good strategy?
- Interaction of literal selection and ordering

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`**