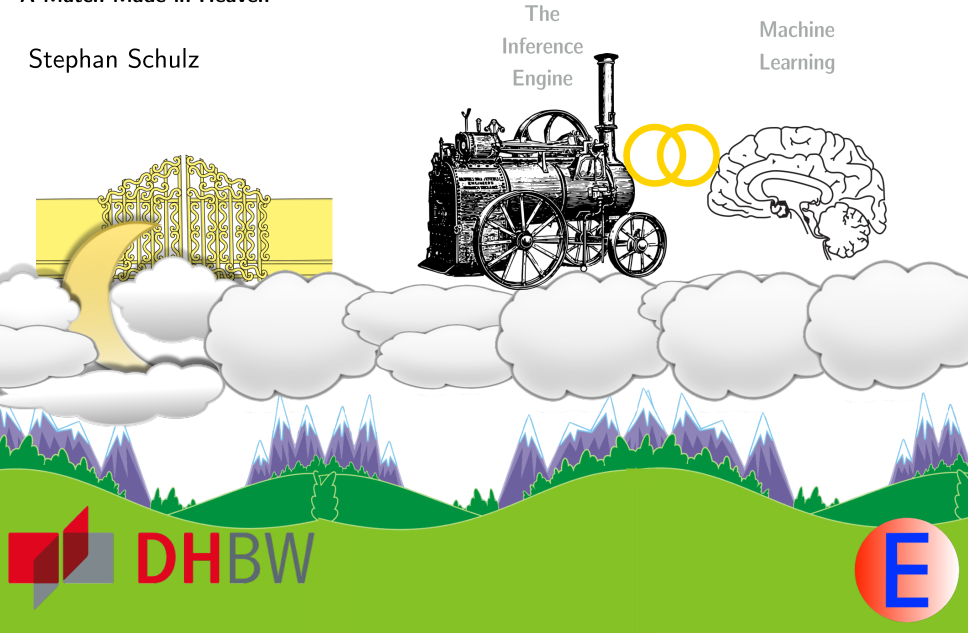
Deduction and Induction

A Match Made in Heaven

Stephan Schulz



Deduction and Induction

A Match Made in Heaven or a Deal with the Devil?

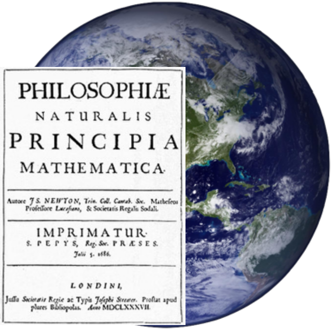
Stephan Schulz

Agenda

- Search and choice points in saturating theorem proving
- Basic questions about learning
- Learning from performance data
  - Classification and heuristic selection
  - Parameters for clause selection
- Learning from proofs and search graphs
  - Proof extraction
  - Learning clause evaluations (?)
- Conclusion

Theorem Proving: Big Picture

Real World Problem



Formalized Problem

$$\forall X : human(X) \rightarrow mortal(X)$$
$$\forall X : philosopher(X) \rightarrow human(X)$$
$$philosopher(socrates)$$

$$\overset{?}{\models}$$

$$mortal(socrates)$$

ATP

Proof Search

Proof

or

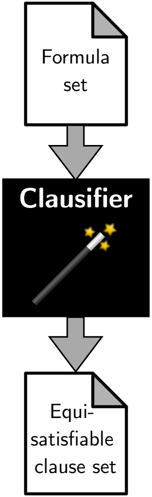
Countermodel

or

Timeout

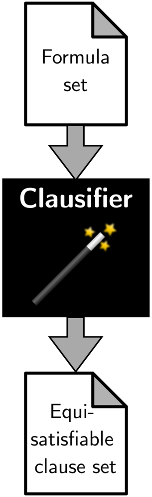
Contradiction and Saturation

- Proof by contradiction
  - Assume negation of conjecture
  - Show that axioms and negated conjecture imply falsity
- Saturation
  - Convert problem to Clause Normal Form
  - Systematically enumerate logical consequences of axioms and negated conjecture
  - Goal: Explicit contradiction (empty clause)
- Redundancy elimination
  - Use contracting inferences to simplify or eliminate some clauses



Contradiction and Saturation

- Proof by contradiction
  - Assume negation of conjecture
  - Show that axioms and negated conjecture imply falsity
- Saturation
  - Convert problem to Clause Normal Form
  - Systematically enumerate logical consequences of axioms and negated conjecture
  - Goal: Explicit contradiction (empty clause)
- Redundancy elimination
  - Use contracting inferences to simplify or eliminate some clauses



Search control problem: How and in which order do we enumerate consequences?

Proof Search and Choice Points

- First-order logic is semi-decidable
  - Provers search for proof in infinite space
  - ...of possible derivations
  - ...of possible consequences
- Major choice points of Superposition calculus:
  - Term ordering (which terms are bigger)
  - (Negative) literal selection
  - Selection of clauses for inferences (with the given clause algorithm)

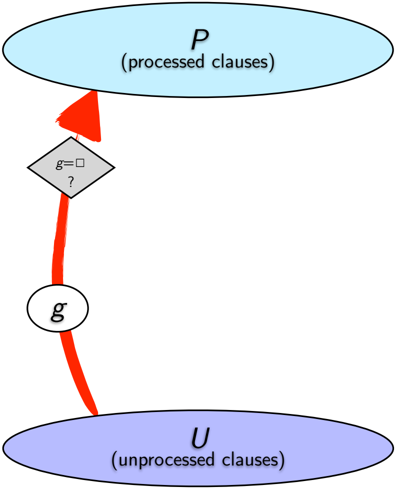
Term Ordering and Literal Selection

- Negative Superposition with selection

$$\frac{C \vee s \simeq t \quad D \vee u \not\simeq v}{(C \vee D \vee u_{[p \leftarrow t]} \not\simeq v)_\sigma}$$

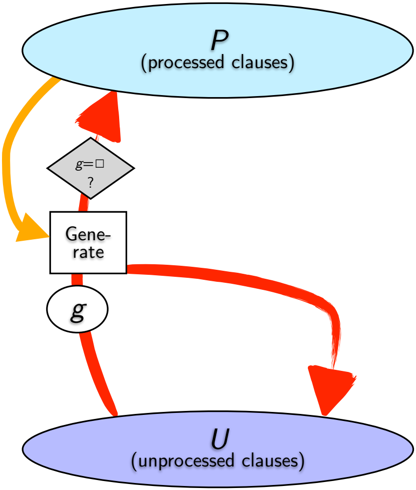
  - if $\sigma = \mathrm{mgu}(u|_p, s)$
    - and $(s \simeq t)_\sigma$ is $\succ$-maximal in $(C \vee s \simeq t)_\sigma$
    - and s is $\succ$-maximal in $(s \simeq t)_\sigma$
    - and $u \simeq v$ is selected in $D \vee u \not\simeq v$
    - and u is $\succ$-maximal in $(s \simeq t)_\sigma$
- Choice points:
  - $\succ$ is a ground-total rewrite ordering
    - Consistent throughout the proof search
    - I.e. in practice determined up-front
  - Any negative literal can be selected
    - Current practice: Fixed scheme picked up-front

The Given-Clause Algorithm



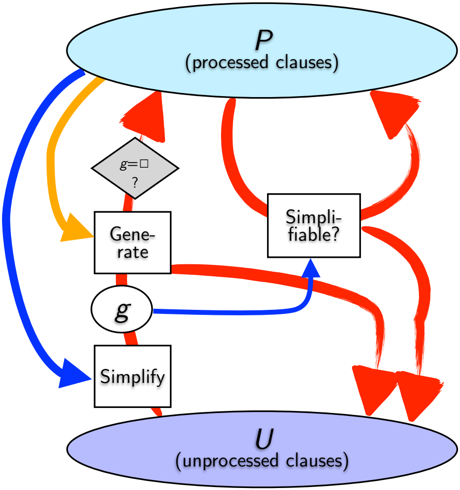
- Aim: Move everything from U to P

The Given-Clause Algorithm



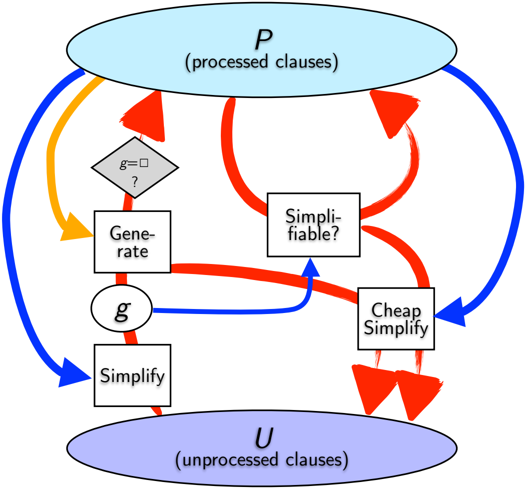
- Aim: Move everything from U to P
- Invariant: All generating inferences with premises from P have been performed

The Given-Clause Algorithm



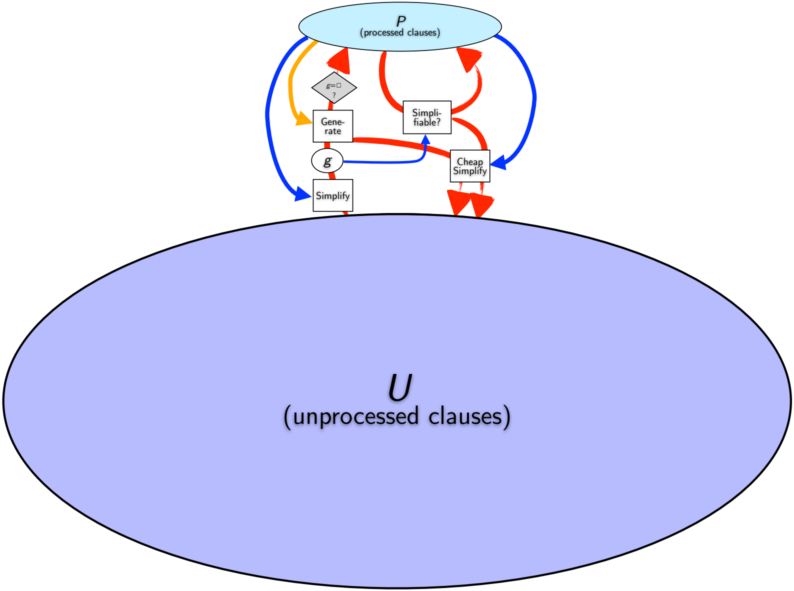
- Aim: Move everything from U to P
- Invariant: All generating inferences with premises from P have been performed
- Invariant: P is interreduced

The Given-Clause Algorithm

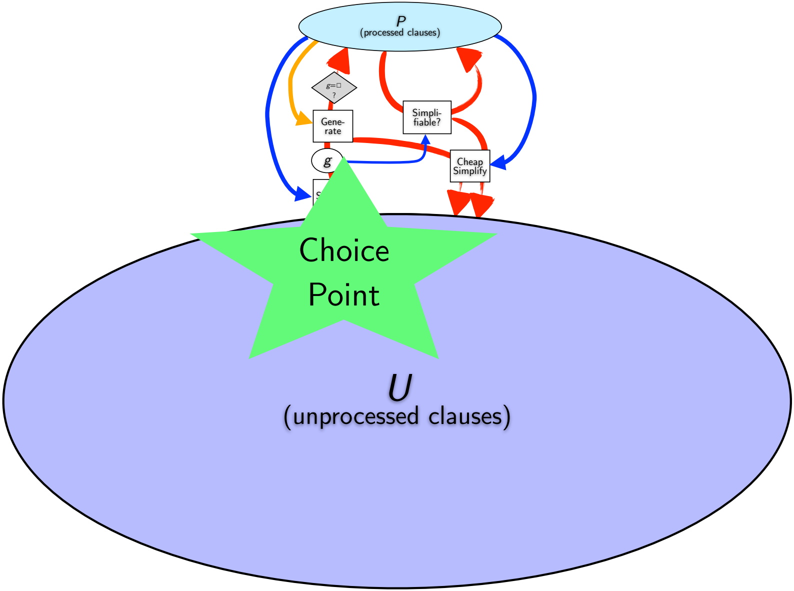


- Aim: Move everything from U to P
- Invariant: All generating inferences with premises from P have been performed
- Invariant: P is interreduced
- Clauses added to U are simplified with respect to P

Choice Point Clause Selection

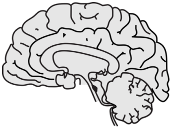


Choice Point Clause Selection



Induction for Deduction

- Question 1: What to learn from?
  - Performance data (prover is a black box)
  - Proofs (only final result of search is visible)
  - Proof search graphs (most of search is visible)
- Question 2: What to learn?
  - Here: Learn strategy selection
  - Here: Learn parameterization for clause selection heuristics
  - Here: Learn new clause evaluation functions
  - ...



Automatic Strategy Selection

Strategy Selection

Definition: A strategy is a collection of all search control parameters

- Term ordering
- Literal selection scheme
- Clause selection heuristic
- ... (minor parameters)

Strategy Selection

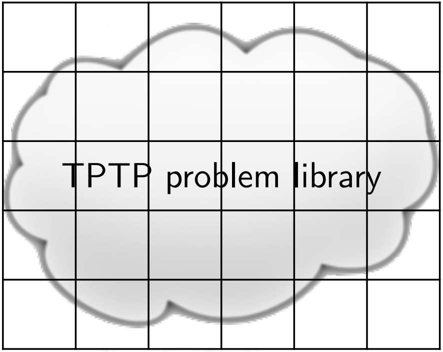
Definition: A strategy is a collection of all search control parameters

- Term ordering
- Literal selection scheme
- Clause selection heuristic
- ... (minor parameters)

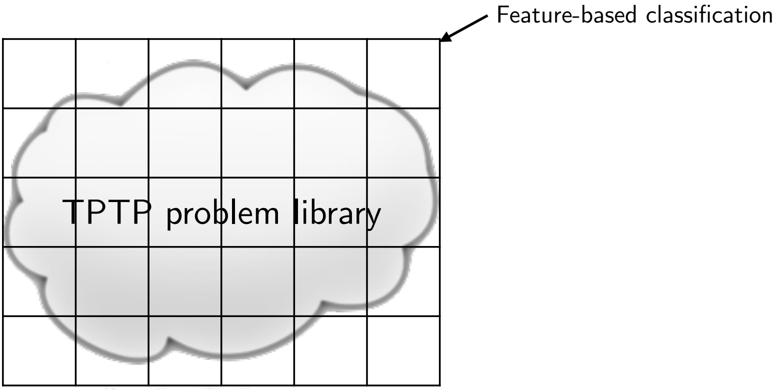
- Observation: Different problems are simple for different strategies
- Question: Can we determine a good heuristic (or set of heuristics) up-front?
- Original: Manually coded automatic modes
  - Based on developer intuition/insight/experience
  - Limited success, high maintenance
- State of the art: Automatic generation of automatic modes

"Learning" Heuristic Selection

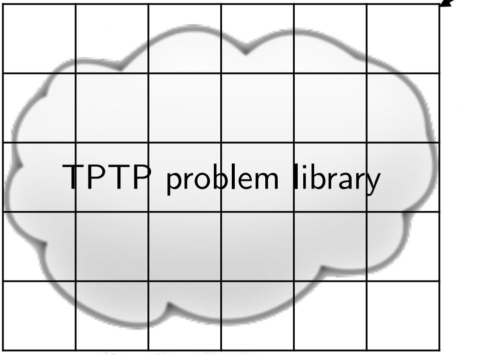
“Learning” Heuristic Selection



“Learning” Heuristic Selection



“Learning” Heuristic Selection

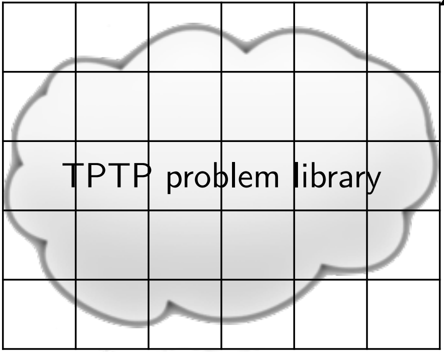


Feature-based classification

Assign strategies to classes based on collected performance data from previous experiments

- Simplest: Always pick best strategy in class
- If no data, pick globally best

“Learning” Heuristic Selection



Feature-based classification

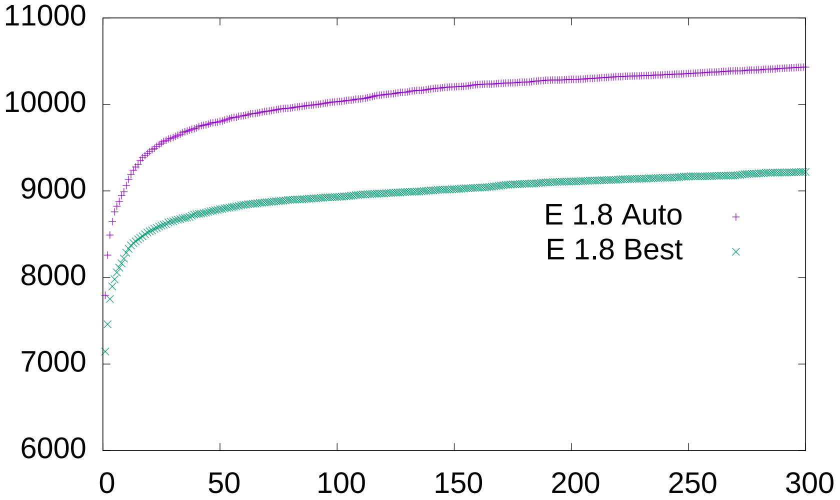
Assign strategies to classes based on collected performance data from previous experiments

- Simplest: Always pick best strategy in class
- If no data, pick globally best

Example features

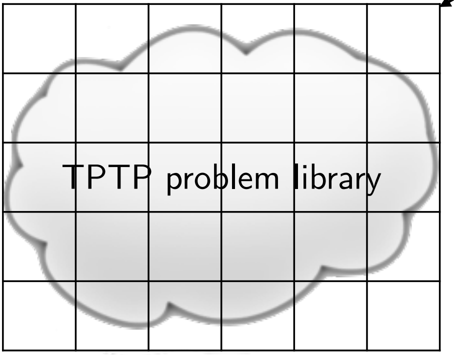
- Number of clausse
- Arity of symbols
- Unit/Horn/Non-horn

Auto Mode Performance



TPTP 5.6.0 CNF&FOF problems

A Caveat



Feature-based classification

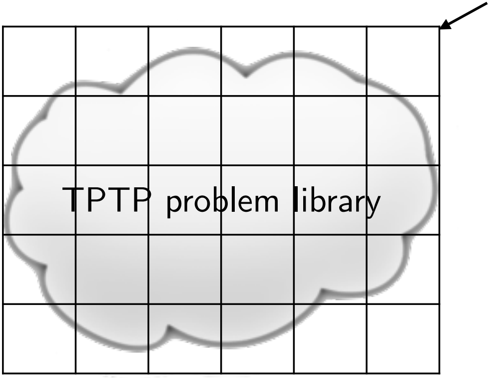
Assign strategies to classes based on collected performance data from previous experiments

- Simplest: Always pick best strategy in class
- If no data, pick globally best

Example features

- Number of clausse
- Arity of symbols
- Unit/Horn/Non-horn

A Caveat



Feature-based classification

Assign strategies to classes based on collected performance data from previous experiments

- Simplest: Always pick best strategy in class
- If no data, pick globally best

Example features

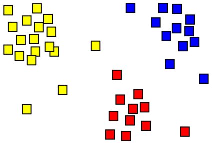
- Number of clausse
- Arity of symbols
- Unit/Horn/Non-horn

Features based on developer...

- ...intuition
- ...insight
- ...experience

Current Work: Learning Classification

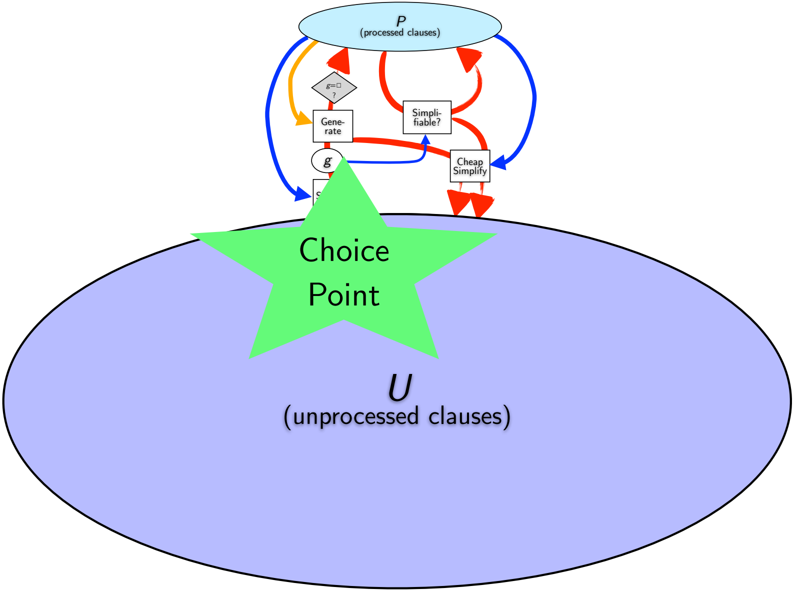
- Characterize problems by performance vectors
  - Which strategy solved the problem how fast?
- Unsupervised clustering of problems based om performance
  - Each cluster contains problems on which the same strategies perform well
- Feature extraction: Try to find characterization of clusters
  - E.g. based on feature set
  - E.g. using nearest-neighbour approaches



My Bachelor Student Ayatallah just started work on this topic - results in 6 months

Learning parameterization for clause selection heuristics

Reminder: Choice Point Clause Selection



Basic Approaches to Clause Selection

- Symbol counting
  - Pick smallest clause in U
  - $|\{f(X) \neq a, P(a) \neq \$true, g(Y) = f(a)\}| = 10$
- FIFO
  - Always pick oldest clause in U
- Flexible weighting
  - Symbol counting, but give different weight to different symbols
  - E.g. lower weight to symbols from goal!
  - E.g. higher weight for symbols in inference positions
- Combinations
  - Interleave different schemes

Given-Clause Selection in E (1)

- Domain Specific Language (DSL) for clause selection scheme
- Arbitrary number of priority queues
- Each queue ordered by:
  - Unparameterized priority function
  - Parameterized heuristic evaluation function
- Clauses picked using weighted round-robin scheme
  - Example (5 queues):

```
(1*ConjectureRelativeSymbolWeight(SimulateSOS,
         0.5, 100, 100, 100, 100, 1.5, 1.5, 1),
 4*ConjectureRelativeSymbolWeight(ConstPrio,
         0.1, 100, 100, 100, 100,1.5, 1.5, 1.5),
 1*FIFOWeight(PreferProcessed),
 1*ConjectureRelativeSymbolWeight(PreferNonGoals,
             0.5, 100, 100, 100,100, 1.5, 1.5, 1),
 4*Refinedweight(SimulateSOS,3,2,2,1.5,2))
```

Given-Clause Selection in E (2)

- Example clause selection heuristic

```
(1*ConjectureRelativeSymbolWeight(SimulateSOS,
         0.5, 100, 100, 100, 100, 1.5, 1.5, 1),
 4*ConjectureRelativeSymbolWeight(ConstPrio,
         0.1, 100, 100, 100, 100,1.5, 1.5, 1.5),
 1*FIFOWeight(PreferProcessed),
 1*ConjectureRelativeSymbolWeight(PreferNonGoals,
             0.5, 100, 100, 100,100, 1.5, 1.5, 1),
 4*Refinedweight(SimulateSOS,3,2,2,1.5,2))
```

- Infinitely many possibilities
  - Several integer and floating point parameters per evaluation function
  - Arbitrary combinations of individual evaluation functions

Given-Clause Selection in E (2)

- Example clause selection heuristic

```
(1*ConjectureRelativeSymbolWeight(SimulateSOS,
         0.5, 100, 100, 100, 100, 1.5, 1.5, 1),
 4*ConjectureRelativeSymbolWeight(ConstPrio,
         0.1, 100, 100, 100, 100,1.5, 1.5, 1.5),
 1*FIFOWeight(PreferProcessed),
 1*ConjectureRelativeSymbolWeight(PreferNonGoals,
             0.5, 100, 100, 100,100, 1.5, 1.5, 1),
 4*Refinedweight(SimulateSOS,3,2,2,1.5,2))
```

- Infinitely many possibilities
  - Several integer and floating point parameters per evaluation function
  - Arbitrary combinations of individual evaluation functions

How do we find good clause selection heuristics (without relying on developer intuition, insight, experience)?

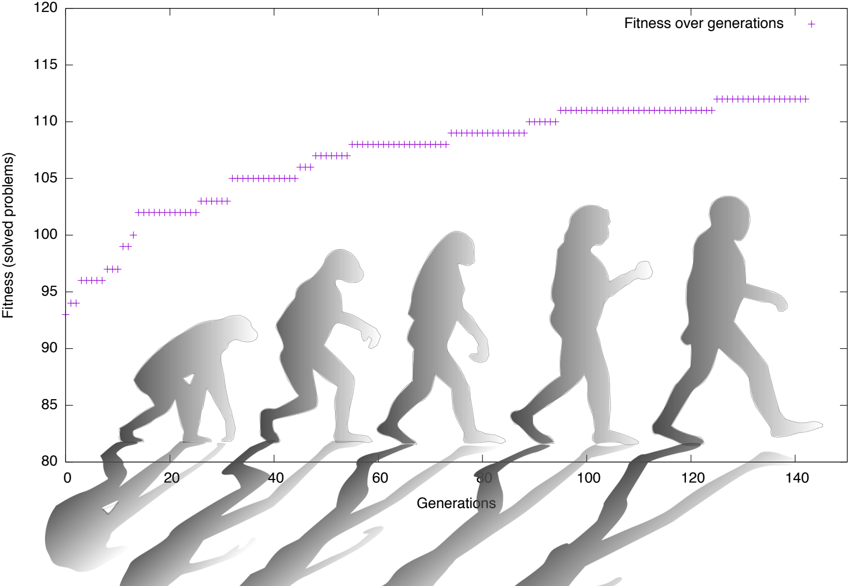
Genetic Algorithms

- Optimization based on evolving population of individuals
  - Optimization is organized in generations
  - In each generation, individuals compete to reproduce
- Each individual is a candidate solution (i.e. search heuristic)
  - Individuals are assigned a fitness score based on performance
  - More fit individuals are more likely to reproduce into the next generation
- The next generation:
  - Mutation - randomly modify individual
  - Crossover - create new individual from two parents
  - Survivors

Applying Genetic Algorithms to Clause Selection

- Encoding: DSL translated into S-Expressions
- Mutation: Randomly modify parameters of one heuristic
- Crossover:
  - Compose individual by randomly inserting evaluation functions from both parents
  - If the same generic evaluation function occurs in both, randomly exchange parameters
- Fitness: How many medium difficulty problems are solved
  - ...on smallish sample set
  - ...with short time limit
- Selection: Tournament selection ($n \approx 5$)

Evolution in Action

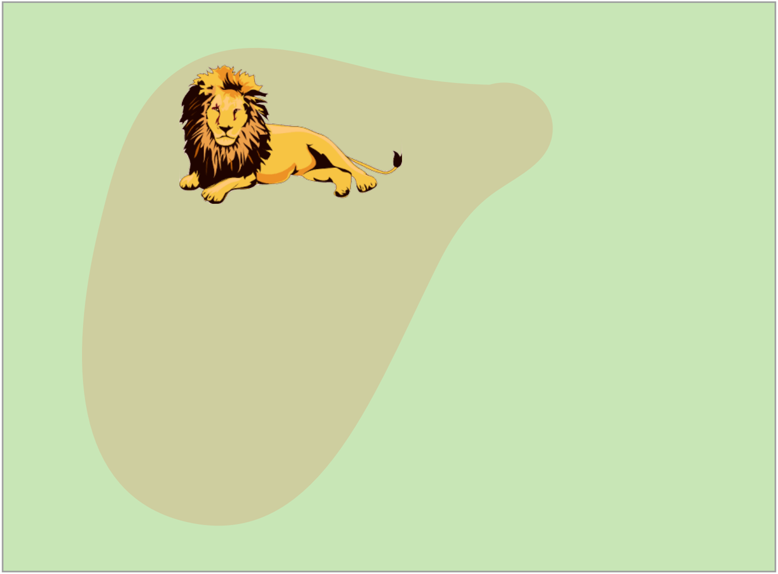


(Very) Preliminary Results

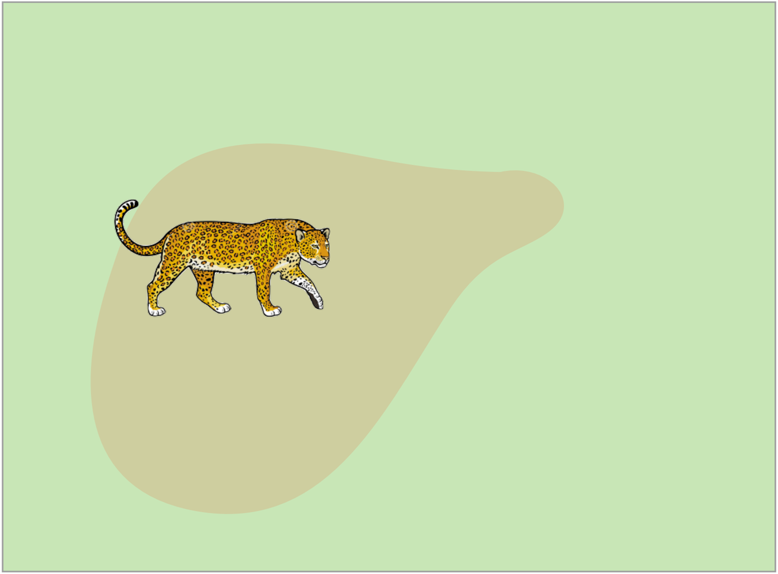
- Evolution finds good clause selection heuristics from random initial population
  - Convergence in ≈ 200 generations
  - Time per generation ≈ 45 CPU hours
  - $\ldots \approx 40$ minutes on 24 core server
- Best evolved heuristic beats best conventional heuristic
  - Evaluation on 15758 problems from TPTP 6.0.0
  - 30 second time limit, 2.6GHz Intel Xeon machines, enough memory
  - Evolved: 8814 solutions found
  - Manual: 8750 solutions found
  - Unique solutions: 466 evolved vs. 386 manual

Current Work: Diversity Beats Ferocity

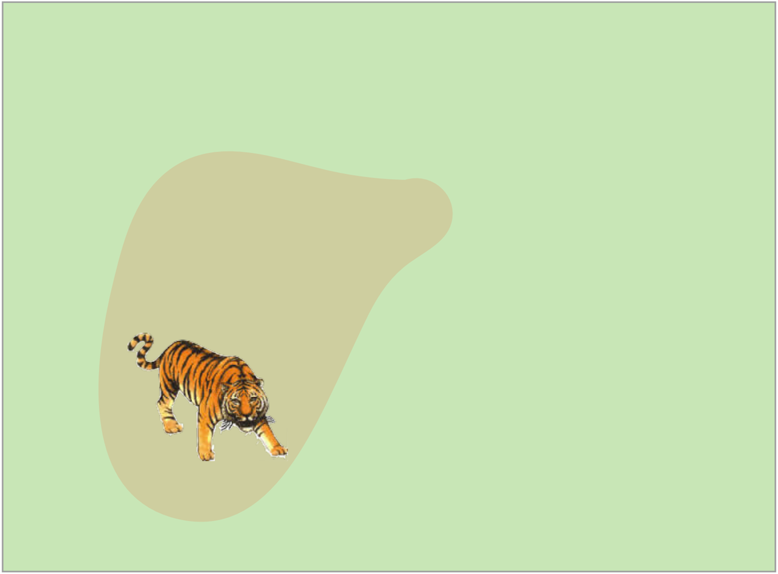
Current Work: Diversity Beats Ferocity



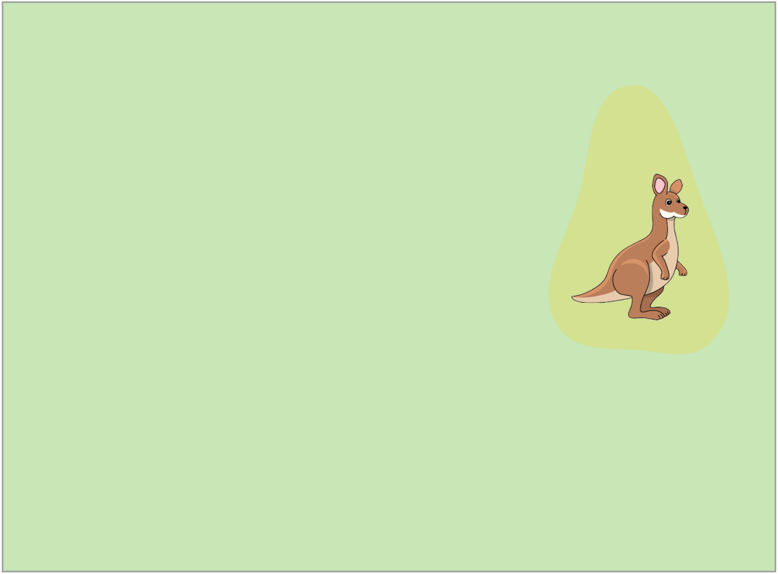
Current Work: Diversity Beats Ferocity



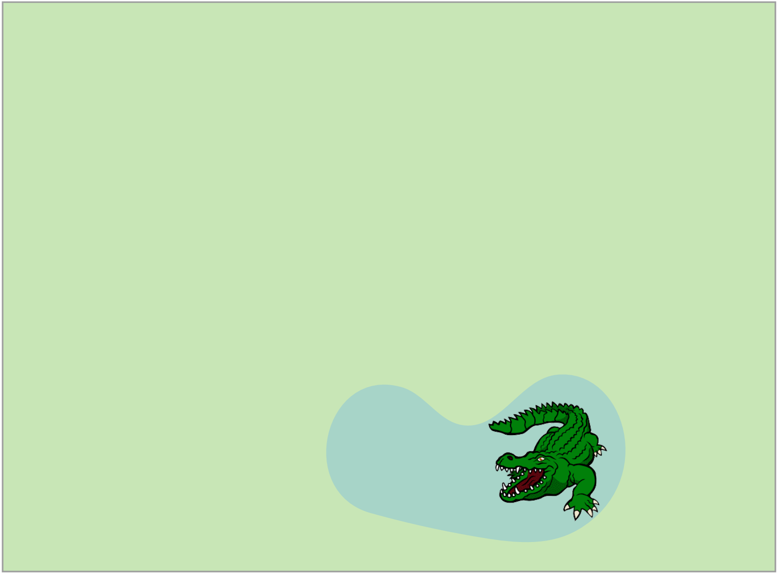
Current Work: Diversity Beats Ferocity



Current Work: Diversity Beats Ferocity

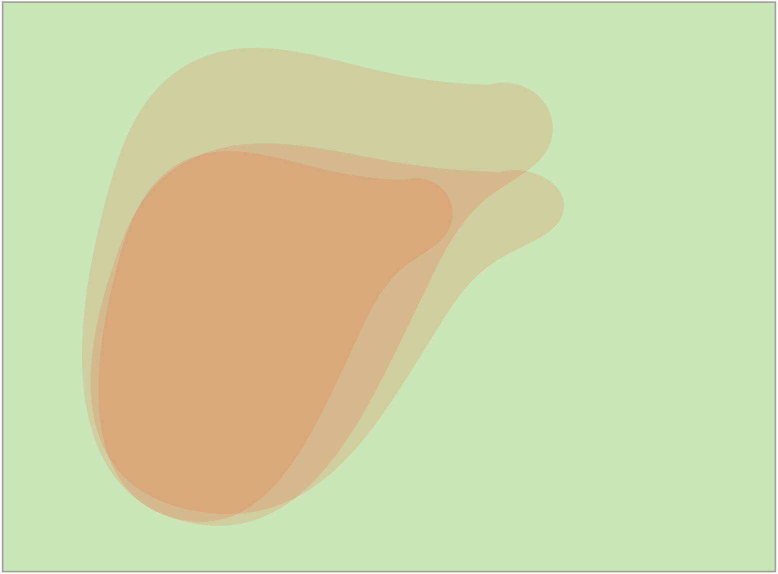


Current Work: Diversity Beats Ferocity



Current Work: Diversity Beats Ferocity

Current Work: Diversity Beats Ferocity



Current Work: Diversity Beats Ferocity

Current Work: Diversity Beats Ferocity

- Idea: Modify fitness function
  - Problems are prey
  - Individual heuristics are predators
  - If several predators catch the same prey, they have to share the benefit
  - $\Longrightarrow$ problems solved by no or few heuristics are more valuable
  - $\Longrightarrow$ Force diversity of the ecosystem

Current Work: Diversity Beats Ferocity

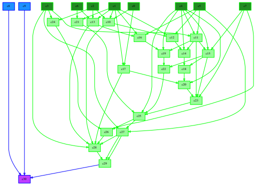
- Idea: Modify fitness function
  - Problems are prey
  - Individual heuristics are predators
  - If several predators catch the same prey, they have to share the benefit
  - $\Longrightarrow$ problems solved by no or few heuristics are more valuable
  - $\Longrightarrow$ Force diversity of the ecosystem

My Bachelor Student Ahmed just started work on this topic - results in 6 months

Proof Extraction and Learning

Learning from Proofs and Proof Search Graphs

- Intuition: Previous proof searches are useful to guide new proof attempts
- Naive approach:
  - Clauses in the proof tree are positive examples
  - (All other clauses are negative examples)
- Initial attempts
  - DISCOUNT (Schulz 1995, Schulz&Denzinger 1996) - UEQ, patterns
  - E (Schulz 2000, 2001) - CNF, patterns
  - Overall, modest successes
  - Mostly with positive examples only - compare Otter's hints

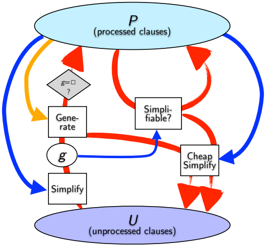


Problems and Solutions

- Problem: Search protocol size
  - Initial approach: Store all intermediate steps
  - Bad time and space performance
  - Borderline impossible in 2000, still hard today
- Problem: Not all examples represent search decisions
  - Many intermediate results
  - Also: Vastly unbalanced ratio of positive/negative examples
- Common solution:
  - Internal proof object (re-)construction
  - Compact representation of the search graph
  - Actually evaluated and picked clauses are recorded
  - Minimal overhead (0.24%) in time
  - Small overhead in memory (due to structure sharing and early discarding of many redundant clauses)

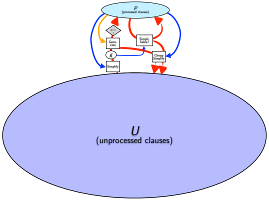
Proof Generation with Limited Archiving

- DISCOUNT loop: Only clauses in P are used for inferences
  - U is subject to simplification, but is passive
  - Only clauses in P need to be available in the proof tree



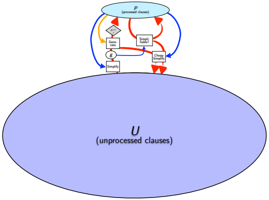
Proof Generation with Limited Archiving

- DISCOUNT loop: Only clauses in P are used for inferences
  - U is subject to simplification, but is passive
  - Only clauses in P need to be available in the proof tree
- Backward simplification is rare
  - Only clauses in P can be backwards-simplified (and P is small)
  - Heuristically, newer clauses are larger (and big clauses rarely simplify small clauses)

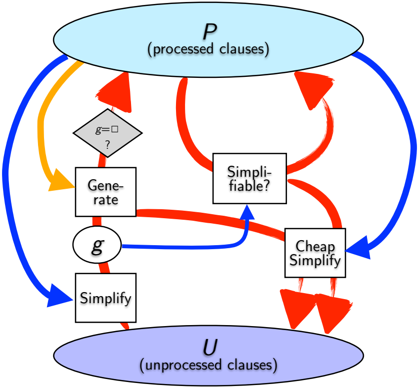


Proof Generation with Limited Archiving

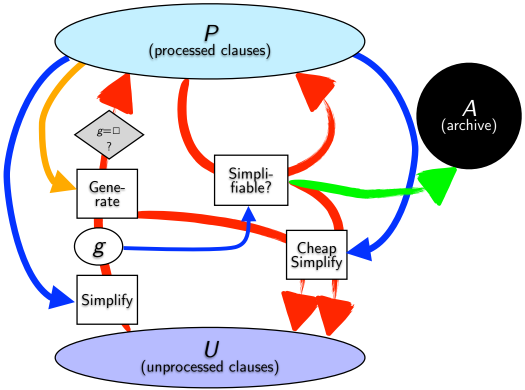
- DISCOUNT loop: Only clauses in P are used for inferences
  - U is subject to simplification, but is passive
  - Only clauses in P need to be available in the proof tree
- Backward simplification is rare
  - Only clauses in P can be backwards-simplified (and P is small)
  - Heuristically, newer clauses are larger (and big clauses rarely simplify small clauses)
- Solution: Non-destructive backwards-simplification
  - Clauses in P are archived on simplification
  - Simplified new clause is build from fresh copy



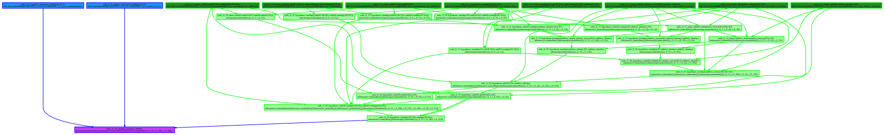
Proof Generation



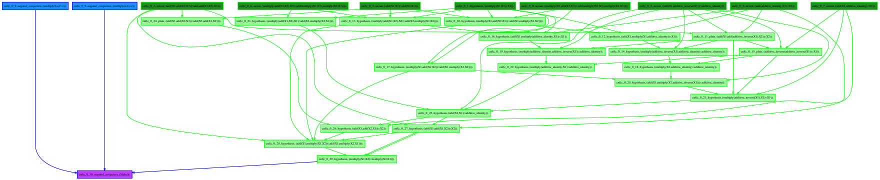
Proof Generation



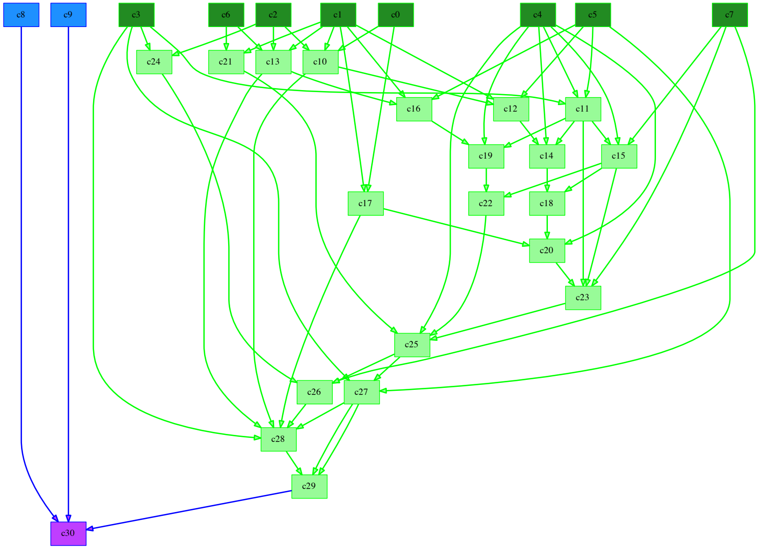
The Structure of Proofs: Commutative Rings



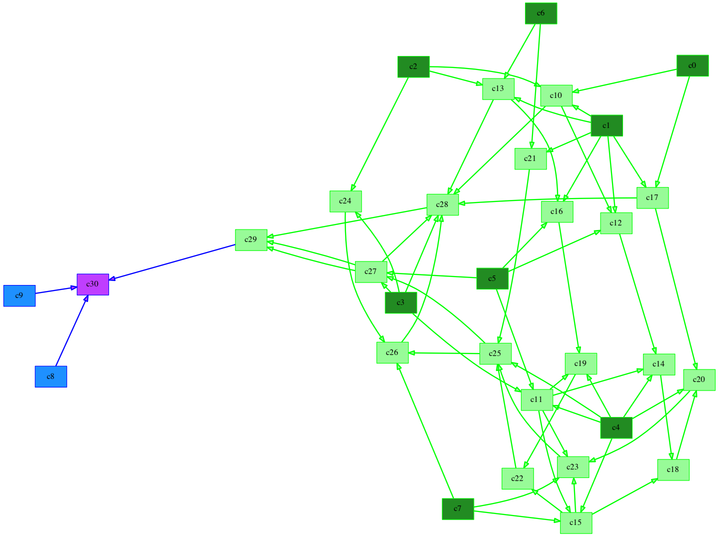
The Structure of Proofs: Commutative Rings



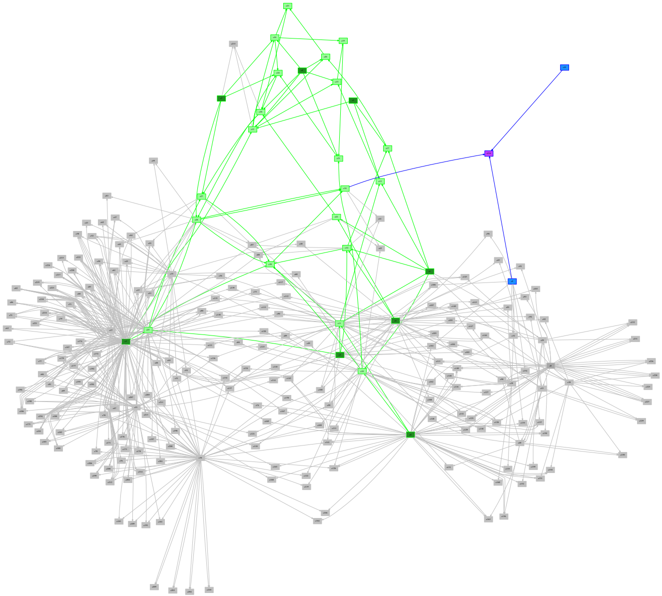
The Structure of Proofs: Commutative Rings



The Structure of Proofs: Commutative Rings

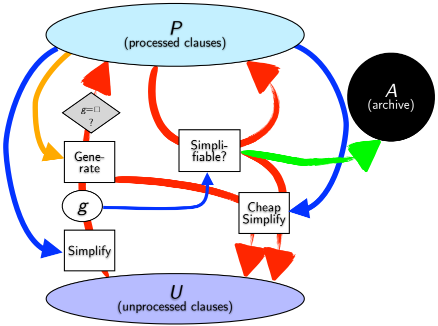


The Structure of Proofs: Commutative Rings



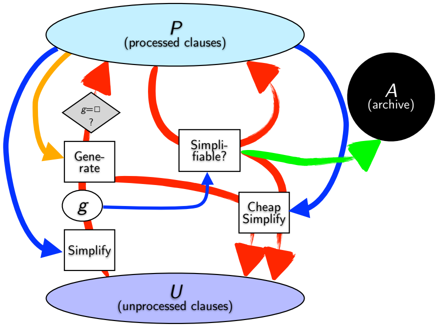
Classification of Search Decisions

- Proof state at success:
  - All proof clauses are in $P \cup A$
  - Clauses in U never contribute
- All clauses in $P \cup A$ have been selected for processing
  - Positive examples: Proof clauses
  - Negative examples: Non-proof clauses



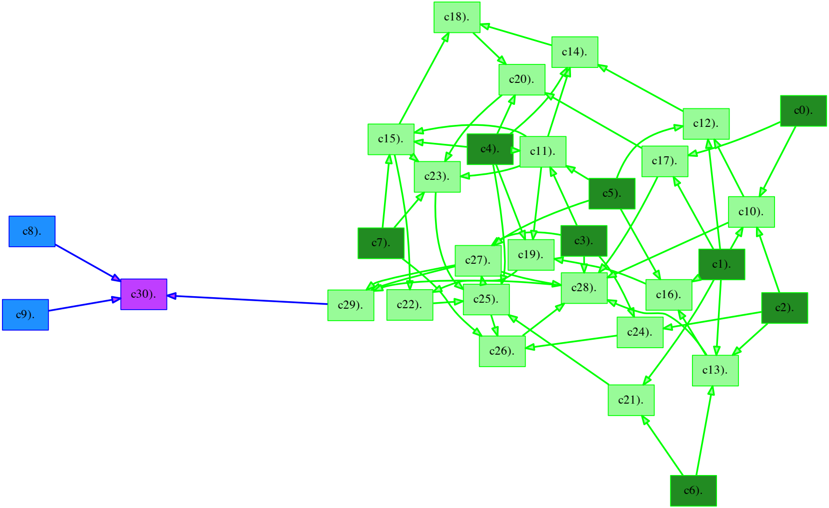
Classification of Search Decisions

- Proof state at success:
  - All proof clauses are in $P \cup A$
  - Clauses in U never contribute
- All clauses in $P \cup A$ have been selected for processing
  - Positive examples: Proof clauses
  - Negative examples: Non-proof clauses

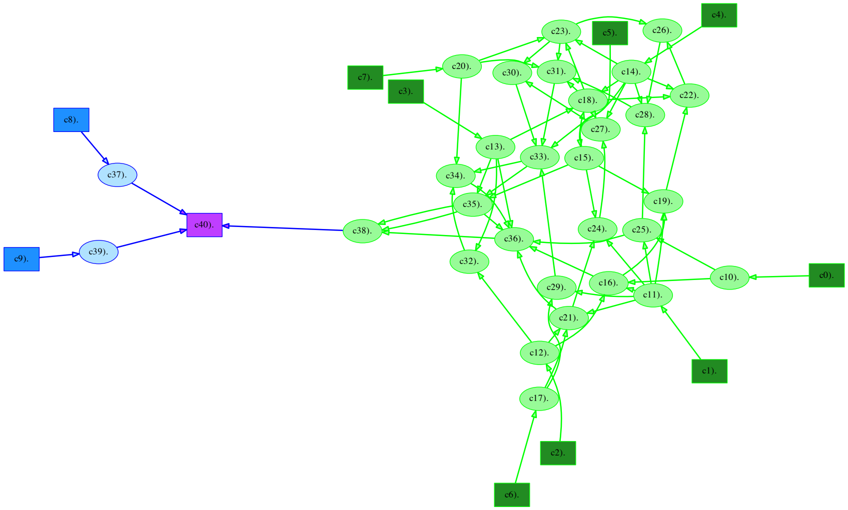


Idea: Apply Machine Learning

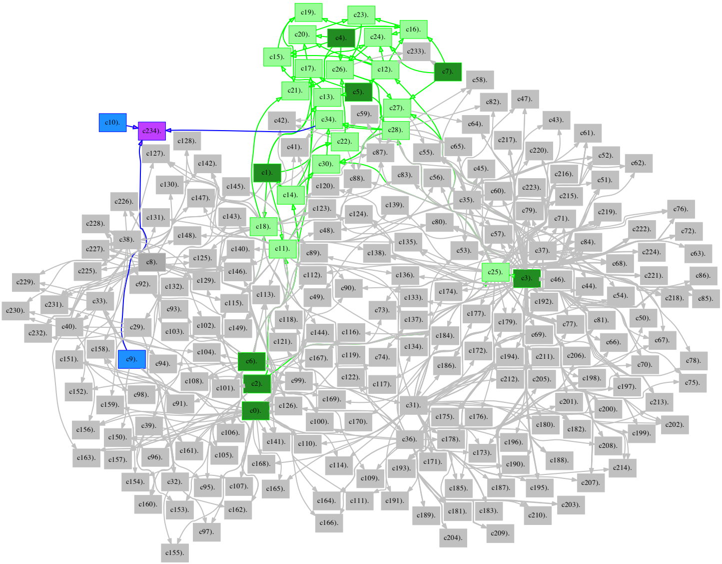
Example: Proof Objects RNG008-7



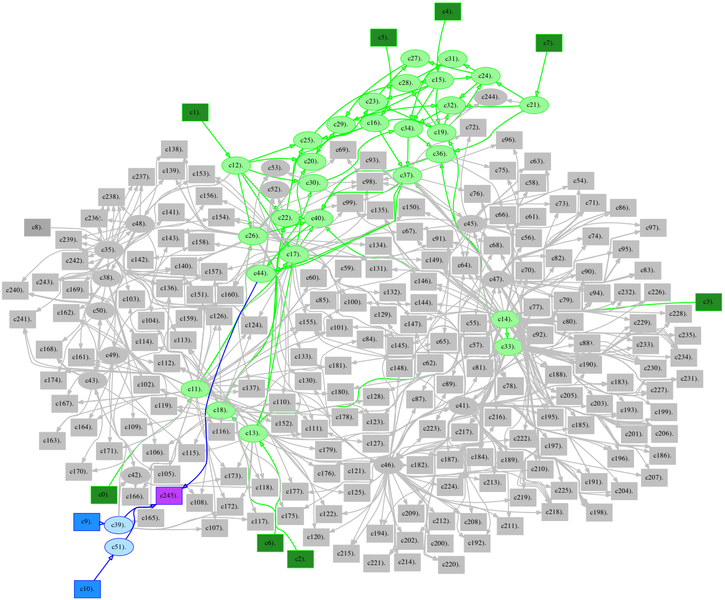
Example: Proof Objects RNG008-7



Example: Proof Objects RNG008-7



Example: Proof Objects RNG008-7

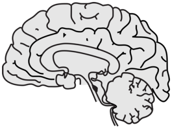


Some Initial Results

- Training examples can be cheaply extracted
- Ratio of utilized to useless given clauses (GCU-ratio) is a good predictor of Heuristic perfomance (Schulz/Möhrmann, IJCAR 2016)
- Positive training examples can be automatically written into a watch list and used as hints
  - Clauses on the watchlist are preferred over all other clauses
  - First experiments
  - Reproving with much better GCU-ratio (and much faster)
  - Some improvement even for related problems

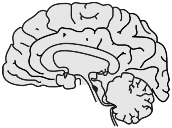
Open Questions

- Abstractions
  - Are concrete function symbols relevant?
  - Is the concrete term structure relevant?
- Learning methods
  - Folding architecture networks?
  - Feature-based numerical methods?
  - Pattern-based learning?
  - Deep learning with convoluted networks?
- Trade-offs
  - Power vs. convenience
  - Speed vs. quality
  - Online vs. offline costs



Open Questions

- Abstractions
  - Are concrete function symbols relevant?
  - Is the concrete term structure relevant?
- Learning methods
  - Folding architecture networks?
  - Feature-based numerical methods?
  - Pattern-based learning?
  - Deep learning with convoluted networks?
- Trade-offs
  - Power vs. convenience
  - Speed vs. quality
  - Online vs. offline costs



Work in Progress

(Nearly) The End

Conclusion

- Controlling proof search for theorem provers is a rich application for machine learning techniques
- Inductive techniques can be applied at several different levels of search control
- Explicit proofs can be generated efficiently
  - ...and mined for training examples!
- Proofs are beautiful and informative
  - Learning from proofs may be the future

Conclusion

- Controlling proof search for theorem provers is a rich application for machine learning techniques
- Inductive techniques can be applied at several different levels of search control
- Explicit proofs can be generated efficiently
  - . . . and mined for training examples!
- Proofs are beautiful and informative
  - Learning from proofs may be the future

Thank you!
Questions?

Image Credit

- Clipart via `http://openclipart.org`
- Hieronimous Bosch via `https://commons.wikimedia.org`