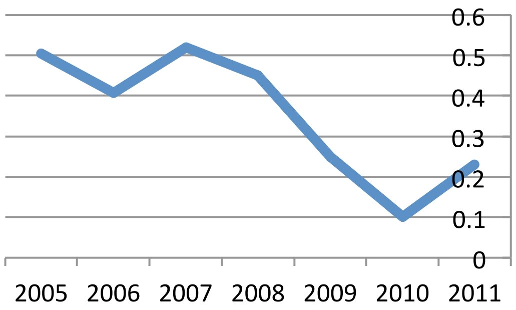
The Vampire Theorem Prover

Krystof Hoder
Andrei Voronkov

Automated First-Order Theorem Proving

- Automated
  - we do not rely on user interaction
  - can be used a black-box by other tools
- First-Order
  - predicate logic with equality
- Extensions
  - sorts
  - arithmetic
- Automated First-Order
  - undecidability – not all can be solved
  - but we keep getting better



relative CASC performance of a reference prover Otter 3.3

TPTP

- TPTP is a universal input language for FO provers
- Also a library of categorized real-life benchmarks

```
fof(kb_SUMOONLY_167,axiom,(
  ! [V__ROW1,V__ROW2] :
   ( ( s__instance(V__ROW2,s__Agent)
     & s__instance(V__ROW1,s__TelecomNumber) )
  => ( s__workPhoneNumber(V__ROW1,V__ROW2)
    => s__telephoneNumber(V__ROW1,V__ROW2) ) ) )).

tff(sum_something_0_samething,conjecture,(
  ! [X: $int] :
   ( ( $less(-1,X)
     & $less(X,1) )
  => $sum(21,X) = 21 ) )).
```

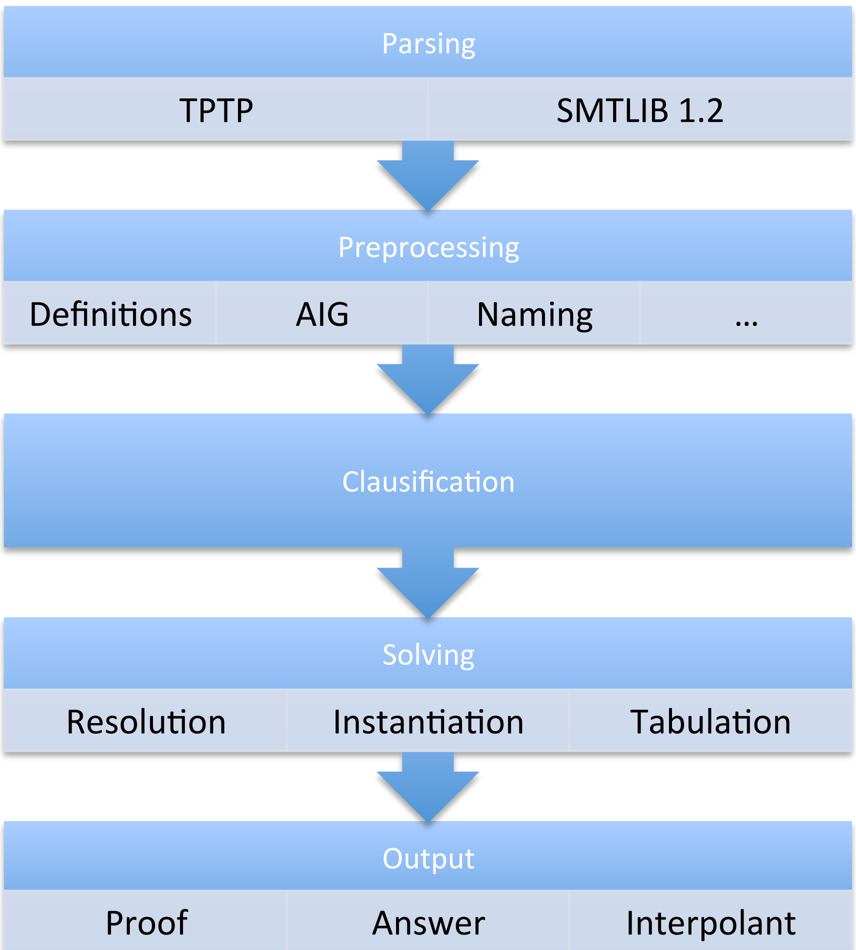
Domains of TPTP benchmarks:

Agents	Logic Calculi
General Algebra	Left Distributive
Analysis	Medicine
Arithmetic	Management
Boolean Algebra	Miscellaneous
Category Theory	Natural Language Processing
Combinatory Logic	Number Theory
Computing Theory	Planning
Commonsense Reasoning	Processes
Data Structures	Puzzles
Fields	Quantales
Geography	Relation Algebra
Geometry	Rings
Graph Theory	Robbins Algebra
Groups	Social Choice Theory
Homological Algebra	Set Theory
Henkin Models	Semantic Web
Hardware Creation	Software Creation
Hardware Verification	**Software Verification**
Kleene Algebra	Syntactic
Knowledge Representation	Topology
Lattices	

CASC

- "World championship" in automated theorem proving
- Several divisions, mostly fragments first-order logic
  - unit equalities, CNF, EPR, general FOF
  - recently also higher-order logic and arithmetic
- Held annually in summer
  - the release dates of theorem provers tend to coincide with the competition date

Vampire Architecture



- Input are general FOF formulas
- Reasoning calculi work with CNF
  - Conjunctive (or Clausal) Normal Form
  - Clause: disjunction of literals
    - p(a) \/ ~q(b) \/ a=b
  - CNF: conjunction of clauses
- Clausification
  - can obscure some information in the problem
  - p <=> (q | r)
  - (~p \/ q \/ r) /\ (p \/ ~q) /\ (p \/ ~r)
  - Preprocessing can exploit this before conversion to CNF

Preprocessing

- Sine axiom selection
  - we will discuss later
- Definition elimination
  - removal of unused
  - inlining
    - may be restricted to avoid blow-up
  - both predicate and function definitions
- Pure predicate removal
- Equality propagation

  x!=a | p(x)

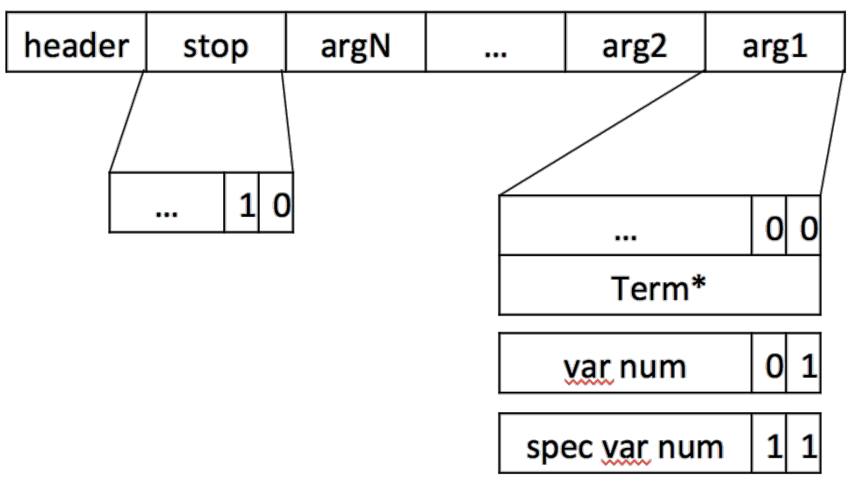
  p(a)

Clausification

- FOF --> ENNF --> NNF
  - NNF has only quantifiers, &, | and literals
- Skolemization
- NNF --> CNF
  - using de Morgan rules
  - (a & b) | (c & d)
    (a | c) & (a | d) & (b | c) & (b | d)
  - Naming can reduce number of generated clauses
  - (a & b) | n
    (c & d) | ~n

Internal representation

- Terms and literals
  - shared by a hash table
  - "prolog" representation



  - fast equality tests
  - pre-computed values (weight, variable count,...)

- Clauses
  - objects with several pre-computed values and a tail array of literals
- Formulas
  - rather naïve implementation, not shared, not garbage collected
  - work in progress on a representation using quantified and-inverter graphs (QAIG)

Resolution and superposition calculus

Most important rules:

- Resolution

$$\frac{A \mid C \qquad \sim B \mid D}{(C \mid D)\sigma}$$

$$\sigma \ldots mgu(A,B)$$

- Subsumption

$$\frac{C \qquad \cancel{D}}{}$$

$$D \underset{multiset}{\supseteq} C\sigma$$

- Superposition

$$\frac{A[s] \mid C \qquad l = r \mid D}{(A[r] \mid C \mid D)\sigma}$$

$$\sigma \ldots mgu(s,l)$$

- Demodulation

$$\frac{l = r \qquad \cancel{C}[l\sigma]}{C[r\sigma]}$$

$$l\sigma \succ r\sigma$$

Some of the ordering and literal selection constraints are omitted.

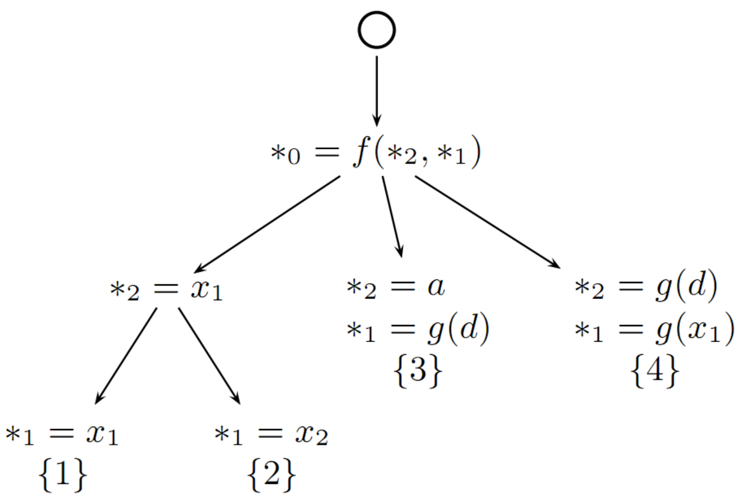
Resolution and superposition calculus

• Resolution $$\frac{A \mid C \qquad \sim B \mid D}{(C \mid D)\sigma}$$ $\sigma \ldots mgu(A,B)$	• Subsumption $$\frac{C \qquad \cancel{D}}{}$$ $D \underset{multiset}{\supseteq} C\sigma$	no equality
• Superposition $$\frac{A[s] \mid C \qquad l = r \mid D}{(A[r] \mid C \mid D)\sigma}$$ $\sigma \ldots mgu(s,l)$	• Demodulation $$\frac{l = r \qquad \cancel{C}[l\sigma]}{C[r\sigma]}$$ $l\sigma \succ r\sigma$	equality
generation rules use unification	simplifying rules use matching	

Indexing

Substitution trees

- Unification, Matching, Instance retrieval



$(1)\ f(x_1, x_1),\ (2)\ f(x_1, x_2),$
$(3)\ f(a, g(d)),\ (4)\ f(g(d), g(x_1)).$

Code trees

- Matching
- Patterns compiled into a tree of abstract machine programs
- Instructions such as

  <bind var 1 to current term and move to next term>

  <check current symbol is f and move to its first argument>

  <check var 2 binding is equal to the current term and move to the next term>

- When a check fails, we backtrack in the tree

Calculus extensions

- Splitting
  - splits long clauses into shorter ones (under some conditions) and does case analysis
- Separate propositional reasoning
  - we can move propositional predicates out of the first-order reasoning and deal with them separately
  - using BDDs and SAT solver
- Unit-resulting hyper-resolution
  - a, b, c, ~a | ~b | ~c | d --> d
- Global subsumption resolution
  - uses SAT solver to find redundant parts of clauses
  - say we have clauses p | b, ~b | ~a and derive p | a
  - from the existing clauses we know that ~p --> ~a, so we can simplify p | a into p

Strategies

- Enabling and disabling various rules and extensions gives a large amount of possible strategies
- We use a computer cluster to explore the strategy space
  - evaluate random strategies on problems from the TPTP library
  - take the best strategies and try to improve them further
- Then we build a "CASC mode"
  - automatically selects a sequence of strategies to use for solving a particular problem
  - puts problem into one of 43 classes, each class has its sequence of strategies

What Matters?

- Features that made a significant improvement
  - Sine
  - DPLL-style splitting
  - Unit hyper-resolution
  - Code trees
- Spider
  - our strategy evaluation system
- The wide variety of strategies
  - it's better to have two **complementary** strategies that each solve 70 distinct problems than one that solves 100

End of the first part

- Any questions?