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

- **Automated First-Order**
  - undecidability – not all can be solved
  - but we keep getting better

- **First-Order**
  - predicate logic with equality

- **Extensions**
  - sorts
  - arithmetic

![relative CASC performance of a reference prover Otter 3.3](image)
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
- General Algebra
- Analysis
- Arithmetic
- Boolean Algebra
- Category Theory
- Combinatory Logic
- Computing Theory
- Commonsense Reasoning
- Data Structures
- Fields
- Geography
- Geometry
- Graph Theory
- Groups
- Homological Algebra
- Henkin Models
- Hardware Creation
- Logic Calculi
- Left Distributive
- Medicine
- Management
- Miscellaneous
- Natural Language Processing
- Number Theory
- Planning
- Processes
- Puzzles
- Quantales
- Relation Algebra
- Rings
- Robbins Algebra
- Social Choice Theory
- Set Theory
- Semantic Web
- Software Creation
  - **Hardware Verification**
  - **Software Verification**
- Syntactic
- Topology
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
    - $p(a) \lor \sim q(b) \lor a=b$
    - CNF: conjunction of clauses
- Clausification
  - can obscure some information in the problem
  - $p \iff (q \lor r)$
  - $(\sim p \lor q \lor r) \land (p \lor \sim q) \land (p \lor \sim r)$
  - 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**
  \[ x \neq a \mid p(x) \]
  \[ p(a) \]

---

**Clausification**

- **FOF --> ENNF --> NNF**
  - NNF has only quantifiers, &, | and literals

- **Skolemization**

- **NNF --> CNF**
  - using de Morgan rules
  - \( (a \& b) \mid (c \& d) \)
    \( (a \mid c) \& (a \mid d) \& (b \mid c) \& (b \mid d) \)
  - Naming can reduce number of generated clauses
  - \( (a \& b) \mid n \)
    \( (c \& d) \mid \lnot n \)
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**

\[
\begin{align*}
A | C & \sim B | D \\
(C | D)_{\sigma} & \\
\sigma \ldots \text{mgu}(A, B)
\end{align*}
\]

- **Superposition**

\[
\begin{align*}
A[s] | C & l = r | D \\
(A[r] | C | D)_{\sigma} & \\
\sigma \ldots \text{mgu}(s, l)
\end{align*}
\]

- **Subsumption**

\[
\begin{align*}
C & D \\
D \supseteq C_{\sigma} & \text{ multiset}
\end{align*}
\]

- **Demodulation**

\[
\begin{align*}
l = r & \subseteq [l_{\sigma}] \\
C[r_{\sigma}] & \\
l_{\sigma} \succ r_{\sigma}
\end{align*}
\]

Some of the ordering and literal selection constraints are omitted.
# Resolution and superposition calculus

<table>
<thead>
<tr>
<th>Resolution</th>
<th>Subsumption</th>
<th>Demodulation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A \mid C \sim B \mid D$</td>
<td>$C \quad D$</td>
<td>$l = r \quad l \sigma \supseteq r \sigma$</td>
</tr>
<tr>
<td>(C \mid D)\sigma</td>
<td>$D \supseteq C \sigma$</td>
<td>$C[r\sigma]$</td>
</tr>
<tr>
<td>$\sigma \ldots \text{mgu}(A, B)$</td>
<td>multiset</td>
<td>$l \sigma \supset r \sigma$</td>
</tr>
<tr>
<td>Superposition</td>
<td>Subsumption</td>
<td>Demodulation</td>
</tr>
<tr>
<td>$A[s] \mid C \quad l = r \mid D$</td>
<td>$C \quad D$</td>
<td>$l = r \quad c[l\sigma]$</td>
</tr>
<tr>
<td>$(A[r] \mid C \mid D)\sigma$</td>
<td>$C[r\sigma]$</td>
<td>$C[l\sigma]$</td>
</tr>
<tr>
<td>$\sigma \ldots \text{mgu}(s, l)$</td>
<td>multiset</td>
<td>$l \sigma \supset r \sigma$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Resolution</th>
<th>Subsumption</th>
<th>Demodulation</th>
</tr>
</thead>
<tbody>
<tr>
<td>generation rules</td>
<td>no equality</td>
<td>equality</td>
</tr>
<tr>
<td>use unification</td>
<td>use matching</td>
<td>use matching</td>
</tr>
</tbody>
</table>

**Simplifying rules**

- Resolution
- Superposition
- Subsumption
- Demodulation

**Unifying rules**

- Resolution
- Superposition
- Subsumption
- Demodulation
Indexing

Substitution trees
• Unification, Matching, Instance retrieval

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

$$\begin{align*}
  *_0 &= f(*_2, *_1) \\
  *_2 &= x_1 \\
  *_1 &= g(d) \\
  *_0 &= f(*_2, *_1) \\
  *_2 &= a \\
  *_1 &= g(d) \\
  *_1 &= g(x_1) \\
  *_1 &= x_1 \\
  *_1 &= x_2 \\
  \{1\} &\quad \{2\}
\end{align*}$$

(1) $f(x_1, x_1)$, (2) $f(x_1, x_2)$, (3) $f(a, g(d))$, (4) $f(g(d), g(x_1))$. 
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**
  – $a, b, c, \sim a | \sim b | \sim c | d \rightarrow d$

• **Global subsumption resolution**
  – uses SAT solver to find redundant parts of clauses
  – say we have clauses $p | b, \sim b | \sim a$ and derive $p | a$
  – from the existing clauses we know that $\sim p \rightarrow \sim a$, 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?