SPASS: Combining Superposition, Sorts and Splitting

Christoph Weidenbach
Max-Planck-Institute for Computer Science
http://spass.mpi-db.mpg.de

Presented by Mooly Sagiv
Bibliography

• **SPASS: Combining Superposition, Sorts and Splitting**  
  C. Weidenbach  
  Handboook of Automated Reasoning

• **Refinements of Resolution** H. de Nivelle

• **Resolution for propositional logic** A. Voronkov

• **A Theory of Resolution** L. Bachmair and H. Ganzinger  
  Handbook of Automated Reasoning

• **A Machine Oriented Logic Based on the Resolution Principle**  
  J.A. Robinson, JACM 1965
General

- The unsatisfiability problem for FOL is undecidable
  - No terminating algorithm which says yes $\leftrightarrow$ the formula is non satisfiable
- The unsatisfiability problem is enumerable
- Resolution is such enumeration procedure
- Implemented in Otter, Spass, Bliksem, Vampire, …
- Succeed in proving interesting theorems
  - Adapts to certain decidable logics
- But predictability is an issue
- Limited practical usage
Clauses

• A literal is an atom or its negation
  – positive literal = atom
  – negative literal = negated atom

• A clause is a finite multiset of literals

• The meaning of \{A_1, A_2, ..., A_n\} is:
  \( \forall X_1, X_2, ..., X_n: (A_1 \lor A_2 \lor ... \ A_n) \)

• The goal is to refute a given finite set of clauses

• Prove that \( C_1 \land C_2 \land ... \land C_n \rightarrow D \) by refuting
  \( \{C_1, C_2, ..., C_n, '\neg D'\} \)
Unifying Terms

• **Substitution**: A mapping $\sigma$ from the set of variables to the terms such that $X\sigma \neq X$ only for finitely many $X$

• Generalizes to terms and literals

• $\sigma$ is a **matcher** for terms $s$ and $t$ if $s \sigma = t$

• $\sigma$ is a **unifier** for terms $s$ and $t$ if $s \sigma = t \sigma$

• $\sigma$ is the **most general unifier** (mgu) of $s$ and $t$ if:
  
  - It is a unifier of $s$ and $t$
  
  - For every unifier $\tau$ of $s$ and $t$ there exists a substitution $\lambda$ such that $\lambda \sigma = \tau$
## Examples

<table>
<thead>
<tr>
<th>Term 1</th>
<th>Term 2</th>
<th>Unifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>X</td>
<td>{X \leftrightarrow a}</td>
</tr>
<tr>
<td>p(a, X)</td>
<td>p(Y, b)</td>
<td>{X \leftrightarrow b, Y \leftrightarrow a}</td>
</tr>
<tr>
<td>p(f(X), g(Z))</td>
<td>p(f(a), Y)</td>
<td>{X \leftrightarrow a, Y \leftrightarrow g(Z)}</td>
</tr>
<tr>
<td>p(f(X), g(Z))</td>
<td>p(f(a), Y)</td>
<td>{X \leftrightarrow a, Y \leftrightarrow g(a), Z \leftrightarrow a}</td>
</tr>
</tbody>
</table>
Resolution

- C and D clauses w/o overlapping variables
- \( \emptyset \neq P \subseteq C \) with positive literals
- \( \emptyset \neq N \subseteq D \) with negative literals
- There exists a substitution \( \sigma \)
  - \( P \sigma = \{A\} \)
  - \( N \sigma = \{\neg A\} \)
- Then: \( ((C - P)\tau \cup (D - N) \tau) \)
  - where \( \tau = \text{mgu}(P, N) \)
Example

1: \{\neg p(X, Y), \ p(Y, X)\}
2: \{\neg p(X, Y), \neg p(Y, Z), \ p(X, Z)\}
3: \{p(X, f(X))\}
4: \{\neg p(a, a)\}
Resolution and Factoring

• Two types of resolution
  – Unify literals within one clause (factoring)
  – Unify literals within different clauses

• Advantage of separation
  – Reduce the cost of resolution
  – Reduce the size of clauses
Resolution

\[
\frac{\Gamma_1, A \rightarrow \Delta_1 \quad \Gamma_2 \rightarrow \Delta_2, B}{(\Gamma_1, \Gamma_2 \rightarrow \Delta_1 \Delta_2) \sigma}
\]

\(\sigma = \text{mgu}(A, B)\)

\[
\begin{align*}
p(f(a), p(f(Y)) \rightarrow p(f(X)) \rightarrow p(X) \\
p(f(a), p(f(f(Y)) \rightarrow \sigma = \{X \mapsto f(Y)\} \\
p(f(a), p(f(f(Y)) \rightarrow
\end{align*}
\]
Factoring

\[
\Gamma \rightarrow \Delta, \ A, B
\]

\[
\frac{\Gamma \rightarrow \Delta, A}{(\Gamma \rightarrow \Delta, A) \sigma}
\]

\[
\sigma = \text{mgu}(A, B)
\]

\[
\Gamma, A, B \rightarrow \Delta
\]

\[
\frac{\Gamma, A \rightarrow \Delta}{(\Gamma, A \rightarrow \Delta) \sigma}
\]

1: \{p(X), p(Y)\}

2: \{\neg p(X), \neg p(Y)\}
Observation

• Simple resolution is easy to implement but does not get very far

• Often diverges due to the inherent complexity of the problem of finding a proof
  – Large possibly infinite search space

• Theorem provers implement refinements (restrictions) to resolution
Refinements of resolution

• Block certain clauses
  – Subsumption & Weight strategies

• Block certain literals in a clause
  – Ordering

• Impose a structure on the resolution
  – Hyperresolution
  – Linear resolution

A refinement is complete if every unsatisfiable set of clauses has a derivation of the empty clause
Subsumption

• Blocks complete clauses from being considered

• If two clauses C and D exist such that $C \subseteq D$ then any conclusion from D can also be obtained from C

• Becomes even more important with equality
Subsumption Deletion

\[ R \frac{\Gamma_1 \rightarrow \Delta_1}{\Gamma_1 \rightarrow \Delta_1} \frac{\Gamma_2 \rightarrow \Delta_2}{\Gamma_1 \rightarrow \Delta_1} \frac{\Gamma_1 \sigma \subseteq \Gamma_2 \text{ and } \Delta_1 \sigma \subseteq \Delta_2}{\Gamma_1 \rightarrow \Delta_1} \]
A Saturation Based Theorem Prover

• Start with an initial set of clauses

• Apply rules and add more clause until either
  – No more clauses can be derives (saturation)
    • The set of clauses is saturated w.r.t. to the inference rules
  – The empty clause $\square$ is derived (refutation)
Simple SPASS rules

\[ \begin{array}{c}
(\Gamma_1, \Delta_1) \rightarrow (\Gamma_2, \Delta_2) \\
\Gamma, \Delta, B \rightarrow \Delta
\end{array} \]

\[ \begin{array}{c}
\sigma = \text{mgu}(A, B)
\end{array} \]
A Simple Resolution Based TP

• A worklist algorithm
• Remember which inference rules have been tried
• Prefer reductions over inferences
• Prefer small clauses
Input reduction

Forward subsumption

Backward subsumption

A Simple Resolution Based TP

ResolutionProver1(N)

\[ W_0 := \emptyset; \]
\[ U_s := \text{taut}(\text{strictsub}(N, N)); \quad \text{Input reduction} \]

while \((U_s \neq \emptyset \text{ and } \nabla \notin U_s)\) {

\((\text{Given, } U_s) = \text{choose}(U_s);\)
\[ W_0 := W_0 \cup \{\text{Given}\}; \]

\[ N_{\text{ew}} := \text{res}(\text{Given, } W_0) \cup \text{fac}(\{\text{Given}\}); \]
\[ N_{\text{ew}} := \text{taut}(\text{strictsub}(N_{\text{ew}}, N_{\text{ew}})); \]

\[ N_{\text{ew}} := \text{sub}(\text{sub}(N_{\text{ew}}, W_0), U_s); \quad \text{Forward subsumption} \]
\[ W_0 := \text{sub}(W_0, N_{\text{ew}}); \quad \text{Backward subsumption} \]
\[ U_s := \text{sub}(U_s, N_{\text{ew}}) \cup N_{\text{ew}}; \]

} if \((U_s = \emptyset)\) then print “Completion Found” ;
If \((\nabla \in U_s)\) then print “Proof found”;
A Simple Example

1: $\rightarrow p(f(a))$
2: $p(f(X)) \rightarrow p(X)$
3: $p(f(a)), p(f(X))$
Fair selection

• ResutionProver1 is complete when choose is fair
  – No clauses stays in Us forever

• A simple fair selection
  – Chose the lightest clause smaller size
  – Finitely many clauses of a given size in a given vocabulary

• Unfair selection may also be useful
  – Ignore clauses which are too big
  – Restart few times with larger bounds
Maintained Invariants

- Any inference conclusion (resolution, factoring) from \( Wo \) is either a tautology or contained/subsumed by a clause in \( Wo, Us \)

- \( Wo \) and \( Us \) are completely inter-reduced
  - \( \text{taut}(Wo \cup Us) = Wo \cup Us \)
  - \( \text{strictsub}(Wo \cup Us, Wo \cup Us) = Wo \cup Us \)

- Partial correctness
  - Upon termination \( Wo \) is saturated or \( \square \in Us \)
Other properties of ResolutionProver1

• In case a $N' \subseteq N$ is known to be satisfiable, initialized with
  – $Wo := N'$;
  – $Us' := (N - N')$

• The initial order of $N$ may be important
Subsumption

- On non-trivial examples $|W_0| \ll |U_s|$
- Subsumption test w.r.t. $U_s$ becomes the bottleneck (95%)
A Second Resolution Based TP

ResolutionProver2(N)

\[ Wo := \emptyset; \]
\[ Us := \text{taut(strictsub}(N, N)) ; \]

while (Us \neq \emptyset and \Box \notin Us) {
    \( (\text{Given, Us}) = \text{choose}(Us); \)
    if (sub(Given), Wo) \neq \emptyset) {
        Wo := sub(Wo, \{Given\});
        Wo := Wo \cup \{Given\};
        New := \text{res}(Given, Wo) \cup \{Given\};
        New := \text{taut}(\text{strictsub}(New, New));
        New := sub(New, Wo);
        Us := Us \cup \text{New};
    }
    if (Us = \emptyset) then print “Completion Found” ;
    if (\Box \in Us) then print “Proof found”;
Maintained Invariants

• Any inference conclusion (resolution, factoring) from $Wo$ is either a tautology or contained/subsumed by a clause in $Wo$, $Us$

• $Wo$ is completely inter-reduced
  – $\text{taut}(Wo) = Wo$
  – $\text{strictsub}(Wo, Wo) = Wo$

• Partial correctness
  – Upon termination $Wo$ is saturated or $\square \in Us$
Ordering

- Block certain literals from consideration
- Impose an order $<$ on literals
- Apply resolution/factoring only on maximal literals
- Drastically reduces the number of applied rules
- Completeness may be an issue
- Can guarantee termination for certain decidable class of logics
Resolution with ordering

\[ \frac{\Gamma_1, A \rightarrow \Delta_1 \quad \Gamma_2 \rightarrow \Delta_2, B}{(\Gamma_1, \Gamma_2 \rightarrow \Delta_1 \Delta_2 )\sigma} \]

\[ \sigma = \text{mgu}(A, B) \]

A is maximal in \( \Gamma_1, A \rightarrow \Delta_1 \)

B is maximal in \( \Gamma_2 \rightarrow \Delta_2, B \)
Propositional example

1: \{a, b\}
2: \{a, \neg b\}
3: \{\neg a, b\}
4: \{\neg a, \neg b\}

\[ a < b < \neg a < \neg b \]
Completeness

• In the propositional case any order results in a complete refinement (Theorem 2.7: De Nivelle)

• In predicate logic the situation is more complicated
  \[ C = \{p(X), q(X), r(X)\} \text{ where } p(X) < q(X) < r(X) \]
  \[ D = \{\neg r(0)\} \]

• An order is **liftable** if \( A < B \) implies \( A \theta \leq B \theta \)

• An order \( < \) on literals is **descending** if
  - \( A < B \Rightarrow A \theta_1 < B \theta_2 \)
  - \( A \theta < A \) when \( \theta \) is not a renaming of \( A \)

• For liftable and descending orders resolution is complete
Orders in Spass

• Knuth-Benedix Ordering (KBO)
  – Invented as part of the Knuth-Benedix completion algorithm
  – Based on orders on functions/predicates
  – Total order on ground terms
  – Useful with handling equalities

• Recursive path ordering with Status
  [Dershowitz 82]
  – Useful for orienting distributivity
Other rules in Spass

• Sort constraint resolution
• Hyperresolution
• Paramodulation
• Splitting
Missing

• The automatic Spass loop (Table 4)
• The overall loop with splitting (Table 7)
• Data structures and algorithms
Conclusion

• Resolution based decision procedures can prove interesting theorems
• Refinements of resolution are essential
• Decidability of certain classes of first order logic is possible
• Combing with specialized decision procedures is a challenge
• Other issues:
  – Scalability
  – Counterexamples