Splitting and Propositional Variables in Resolution Theorem Provers
Splitting and Propositional Variables in Resolution Theorem Provers

Andrei Voronkov (The University of Manchester)
Splitting and Propositional Variables in Resolution Theorem Provers

Krystof Hoder (The University of Manchester)
Andrei Voronkov (The University of Manchester)
Outline

Introduction. Resolution Theorem Proving

Propositional Variables. RePro

Splitting

Experiments

Results
Introduction

- FO problems often contain propositional variables;
- Long clauses can be generated;

Solutions:
- DPLL-style splitting (SPASS)
- Splitting without backtracking (Vampire)

Problem: no extensive evaluation.
Introduction

- FO problems often contain propositional variables;
- Long clauses can be generated;
- Treating propositional variables as ordinary atoms slows down the prover;
- Long clauses slow it down even more.
Introduction

- FO problems often contain propositional variables;
- Long clauses can be generated;
- Treating propositional variables as ordinary atoms **slows down the prover**;
- Long clauses **slow it down even more**.

Solutions:

- **DPLL-style splitting** (SPASS)
- **Splitting without backtracking** (Vampire)
Introduction

- FO problems often contain propositional variables;
- Long clauses can be generated;
- Treating propositional variables as ordinary atoms slows down the prover;
- Long clauses slow it down even more.

Solutions:

- DPLL-style splitting (SPASS)
- Splitting without backtracking (Vampire)

Problem: no extensive evaluation.
Saturation algorithms

1. **Simplifying inferences** replace a clause by another clause that is simpler in some strict sense.
2. **Deletion inferences** delete clauses from the search space.
3. **Generating inferences** derive a new clause from clauses in the search space. This new clause can then be immediately simplified and/or deleted by other kinds of inference.
Long Clauses

They degrade performance considerably.

1. Some inference rules have complexity linear in the size of clauses (for example, rewriting by unit equalities). Some deletion rules (subsumption) and simplification rules (subsumption resolution) are NP-complete. Algorithms for these deletion rules are exponential in the number of literals in clauses.
Long Clauses

They degrade performance considerably.

1. Some inference rules have complexity linear in the size of clauses (for example, rewriting by unit equalities). Some deletion rules (subsumpion) and simplification rules (subsumption resolution) are NP-complete. Algorithms for these deletion rules are exponential in the number of literals in clauses.

2. Generating inferences applied to heavy clauses usually generate heavy clauses. Generating inferences applied to long clauses usually generate even longer clauses. For example, resolution applied to two clauses containing $n_1$ and $n_2$ literals normally gives a clause with $n_1 + n_2 - 2$ literals.
Known Solutions?

- Limited Resource Strategy (Vampire);
- Splitting (SPASS, Vampire, E)
Outline

Introduction. Resolution Theorem Proving

Propositional Variables. RePro

Splitting

Experiments

Results
Propositional Variables: Calculus RePro

This calculus:

- separates propositional reasoning from non-propositional;
- works with augmented clauses of the form $C|P$, where $C$ is a clause having no propositional formulas at all and $P$ is a propositional formula.
Propositional Variables: Calculus RePro

This calculus:

- separates propositional reasoning from non-propositional;
- works with augmented clauses of the form $C|P$, where $C$ is a clause having no propositional formulas at all and $P$ is a propositional formula.

- $C|P$ is logically equivalent to $C \lor P$;
- RePro is a family of calculi, depending on the underlying resolution calculus.
Generating inferences

For every generating inference

\[
\begin{array}{c}
C_1 \quad \cdots \quad C_n \\
\hline \\
C
\end{array}
\]

of the resolution calculus the following is an inference rule of RePro:

\[
\begin{array}{c}
C_1|P_1 \quad \cdots \quad C_n|P_n \\
\hline \\
C|(P_1 \lor \cdots \lor P_n)
\end{array}
\]
Simplifying inferences

For every simplifying inference

\[
\frac{C_1 \cdots C_n \quad \mathcal{D}}{C}
\]

of the resolution calculus, if \( P_1 \lor \ldots \lor P_n \rightarrow P \) is a tautology, then the following is a simplifying inference rule of RePro:

\[
\frac{C_1|P_1 \quad \cdots \quad C_n|P_n \quad D|P}{C|(P_1 \lor \ldots \lor P_n)}
\]
Deletion inferences

For every deletion inference

\[ C_1 \cdots C_n \vdash \emptyset \]

of the resolution calculus, if \( P_1 \lor \cdots \lor P_n \rightarrow P \) is a tautology, then the following is a deletion inference of RePro:

\[ C_1 \upharpoonright P_1 \cdots C_n \upharpoonright P_n \vdash \emptyset \upharpoonright P. \]
Completeness

It is not hard to derive soundness and completeness of RePro assuming the same properties of the underlying resolution calculus.

By completeness here we mean that every fair sequence of sets starting with an unsatisfiable set will contain a set with an empty clause in it, see for details.
Propositional tautology deletion:

\[ \not \models P, \]

where \( P \) is a propositional tautology.
More Rules

Propositional tautology deletion:

\[ D \vdash P, \]

where \( P \) is a propositional tautology.

The merge rule of RePro:

\[
\frac{C \vdash P_1 \quad C \vdash P_2}{C \vdash (P_1 \land P_2)}
\]

Note that so far this is the only rule that introduces propositional formulas other than clauses.
Propositional tautology deletion:

\[ D \vdash P, \]

where \( P \) is a propositional tautology.

The merge rule of RePro:

\[
\frac{C \vdash P_1 \quad C \vdash P_2}{C \vdash (P_1 \land P_2)}
\]

Note that so far this is the only rule that introduces propositional formulas other than clauses.

The merge subsumption rule:

\[
\frac{C \vdash P_1 \quad D \vdash P_2}{D \vdash (P_1 \land P_2)}
\]

where \( C \) subsumes \( D \).
Alternative Formulation

For every simplifying inference

\[
\frac{C_1 \cdots C_n \mathcal{D}}{C}
\]

of the resolution calculus, consider

\[
\frac{C_1 | P_1 \cdots C_n | P_n \quad D | \mathcal{P}}{C | (P_1 \lor \cdots \lor P_n) \quad D | (P_1 \lor \cdots \lor P_n \rightarrow P)}
\]
Alternative Formulation

For every simplifying inference

\[
\frac{C_1 \cdots C_n \mathcal{D}}{C}
\]

of the resolution calculus, consider

\[
\frac{C_1 \mid P_1 \cdots C_n \mid P_n \mathcal{D} \mid \mathcal{P}}{C \mid (P_1 \lor \cdots \lor P_n) \quad \mathcal{D} \mid (P_1 \lor \cdots \lor P_n \rightarrow P)}.
\]

The previously defined simplifying rule is a special case of this one, since, if \(P_1 \lor \cdots \lor P_n \rightarrow P\) is a tautology, so the second inferred clause can be removed.
Alternative Formulation

For every simplifying inference

\[
\begin{array}{c}
C_1 \cdots C_n \quad \emptyset \\
\hline \\
C
\end{array}
\]

of the resolution calculus, consider

\[
\begin{array}{c}
C_1 | P_1 \cdots C_n | P_n \quad D | P \\
\hline \\
C | (P_1 \lor \cdots \lor P_n) \quad D | (P_1 \lor \cdots \lor P_n \rightarrow P)
\end{array}
\]

The previously defined simplifying rule is a special case of this one, since, if \( P_1 \lor \cdots \lor P_n \rightarrow P \) is a tautology, so the second inferred clause can be removed.

One can also reformulate the deletion rules in the same way.
Advantages?

A clause $A \lor B | (p \land q)$ is redundant in the presence if $A|p$ and $B|q$
Advantages?

A clause $A \lor B \models (p \land q)$ is redundant in the presence if $A \models p$ and $B \models q$ using the following sequence of deletion rules:

\[
\begin{align*}
A \models p & \quad A \lor B \models (p \land q) \\
B \models q & \quad A \lor B \models (p \rightarrow (p \land q)) \\
A \lor B \models (q \rightarrow (p \rightarrow (p \land q)))
\end{align*}
\]

whose conclusion is a tautology.
Outline

Introduction. Resolution Theorem Proving

Propositional Variables. RePro

Splitting

Experiments

Results
Observation

Suppose that $S$ is a set of clauses and $C_1 \lor C_2$ a clause such that the variables of $C_1$ and $C_2$ are disjoint.
Observation

Suppose that $S$ is a set of clauses and $C_1 \lor C_2$ a clause such that the variables of $C_1$ and $C_2$ are disjoint.

Then the set $S \cup \{C_1 \lor C_2\}$ is unsatisfiable if and only if both $S \cup \{C_1\}$ and $S \cup \{C_2\}$ are unsatisfiable.
Splittable Clause

Let $C_1, \ldots, C_n$ be clauses such that $n \geq 2$ and all the $C_i$’s have pairwise disjoint sets of variables. Then we say that the clause $C \overset{\text{def}}{=} C_1 \lor \ldots \lor C_n$ is splittable into components $C_1, \ldots, C_n$. There may be more than one way to split a clause, however there is always a unique splitting such that each component $C_i$ is non-splittable: we call this splitting maximal. ▶ A maximal splitting has the largest number of components and every splitting with the largest number of components is the maximal one. ▶ There is a simple algorithm for finding the maximal splitting of a clause, which is, essentially, the union-find algorithm.
Splittable Clause

Let $C_1, \ldots, C_n$ be clauses such that $n \geq 2$ and all the $C_i$'s have pairwise disjoint sets of variables. Then we say that the clause $C \overset{\text{def}}{=} C_1 \lor \ldots \lor C_n$ is splittable into components $C_1, \ldots, C_n$.

There may be more than one way to split a clause, however there is always a unique splitting such that each component $C_i$ is non-splittable: we call this splitting maximal.
Let $C_1, \ldots, C_n$ be clauses such that $n \geq 2$ and all the $C_i$’s have pairwise disjoint sets of variables. Then we say that the clause $C \overset{\text{def}}{=} C_1 \lor \ldots \lor C_n$ is splittable into components $C_1, \ldots, C_n$.

There may be more than one way to split a clause, however there is always a unique splitting such that each component $C_i$ is non-splittable: we call this splitting maximal.

- a maximal splitting has the largest number of components and every splitting with the largest number of components is the maximal one.
- There is a simple algorithm for finding the maximal splitting of a clause, which is, essentially, the union-find algorithm.
Two Ways of Splitting

- DPLL-like (with backtracking)
- Without backtracking (using naming).
Splitting With Backtracking

- **DPLL-like.** Clauses are marked by a splitting history.
- **A lot of work upon backtracking.**
Splitting Without Backtracking

\[
\frac{C_1 \lor C_2}{C_1 \lor p \quad C_2 \lor \neg p},
\]

where

- \( C_1 \) is a minimal component in \( C_1 \) and \( C_2 \);
- \( C_1 \) has no propositional variables;
- \( C_2 \) has a non-propositional atom;
- \( p \) is fresh.

If this rule was previously applied to \( C_1 \), we can consider \( p \) as a name for \( \neg \forall C_1 \) so we have \( \neg p \leftrightarrow \forall C_1 \).
Splitting Without Backtracking

\[
\frac{C_1 \lor C_2}{\begin{array}{c} C_1 \lor p \\ C_2 \lor \neg p \end{array}},
\]

where

- \( C_1 \) is a minimal component in \( C_1 \) and \( C_2 \);
- \( C_1 \) has no propositional variables;
- \( C_2 \) has a non-propositional atom;
- \( p \) is fresh.

\[
\frac{C_1 \lor C_3}{\begin{array}{c} C_3 \lor \neg p \end{array}},
\]

if this rule was previously applied to \( C_1 \).
Splitting Without Backtracking

\[ \frac{C_1 \lor C_2}{C_1 \lor p} \quad \frac{C_2 \lor \neg p}{C_1 \lor p} \quad \frac{C_3 \lor \neg p}{C_3 \lor \neg p} \],

where

- \( C_1 \) is a minimal component in \( C_1 \) and \( C_2 \);
- \( C_1 \) has no propositional variables;
- \( C_2 \) has a non-propositional atom;
- \( p \) is fresh.

\[ \frac{C_1 \lor C_3}{C_3 \lor \neg p} \],

if this rule was previously applied to \( C_1 \).

We can consider \( p \) as a name for \( \neg \forall C_1 \) so we have \( \neg p \leftrightarrow \forall C_1 \).
Splitting Without Backtracking

- Easy to implement, not many changes to a resolution prover.
- Thousands of names can be generated.
Splitting and Saturation Algorithms

Both ways of splitting affect saturation algorithms.

- Redundancy elimination;
- Term indexing;
- Clause selection.
There are 14 different options, 13 of them are boolean and one has 3 values.

However, not every combination of options makes sense, so there are “only” 505 different combinations of splitting-related parameters.

The main option `splitting` has 3 possible values:

- **off**: no splitting
- **backtracking**: splitting with backtracking
- **nobacktracking**: splitting without backtracking
What to Split?

- `split_goal_only`: split only clauses derived from the goal
- `split_input_only`: split only input clauses
- `split_at_activation`: split clause when it is selected for a generating inference
- `split_positive`: split only to components that contain at least one positive literal
Propositional Part

(Only for splitting without backtracking)

- `propositional_to_bdd`: use BDD to represent the propositional part
Inference Selection

- **nonliterals_in_clause_weight**: in $C\bar{P}$, count not only the weight of $C$, but add some additional weight depending on $P$.
- **splitting_with_blocking** (without backtracking): select the introduced negative literal.
Empty Clauses

(Without backtracking).

- `sat_solver_for_empty_clause`: convert the empty clause BDD to a set of clauses and use a SAT solver to deal with them.
- `sat_solver_with_naming`: introduce names for some BDD nodes to avoid an exponential number of clauses.
- `bdd_marking_subsumption`: approximation of subsumption by the empty clause.
- `empty_clause_subsumption`: use a simple test for subsumption of a parent BDD by the empty child BDD.
Other Options

- **sat_solver_with_subsumption_resolution**: use a simple test for subsumption of a parent BDD by the child BDD.
- **split_add_ground_negation**: If one of the components is a ground literal $L$, upon backtracking add the complementary literal.
Outline

Introduction. Resolution Theorem Proving

Propositional Variables. RePro

Splitting

Experiments

Results
Experiments

There are 505 different combinations of splitting-related parameters.
Experiments

There are 505 different combinations of splitting-related parameters.

4,869 TPTP Problems:

- all TPTP problems having non-unit clauses and rating greater than 0.2 and less than 1.
- all rating 1 problems solvable by Vampire;
- excluding very large problems;
Experiments

There are 505 different combinations of splitting-related parameters.

4,869 TPTP Problems:

- all TPTP problems having non-unit clauses and rating greater than 0.2 and less than 1.
- all rating 1 problems solvable by Vampire;
- excluding very large problems;

Strategy:

- take the principal strategy: the one that is believed to solve the largest number of problems;
- create the 505 strategies obtained from the principal one by varying only the splitting-related parameters.
- use time limit of 30 seconds.

This gives 2,458,845 runs, which roughly corresponds to 1.5 years CPU time on a single core computer.
Introduction. Resolution Theorem Proving

Propositional Variables. RePro

Splitting

Experiments

Results
Results

- 4,869 problems;
- 3,598 problems (about 74% of all problems) were solved by at least one strategy;
- 1,053 problems were solved by all 505 strategies;
- this gives us 2,545 “interesting” problems.
Results

- 4,869 problems;
- 3,598 problems (about 74% of all problems) were solved by at least one strategy;
- 1,053 problems were solved by all 505 strategies;
- this gives us 2,545 “interesting” problems.

All selected problems

<table>
<thead>
<tr>
<th>splitting</th>
<th>strategies</th>
<th>worst</th>
<th>average</th>
<th>best</th>
</tr>
</thead>
<tbody>
<tr>
<td>off</td>
<td>25</td>
<td>2,708</td>
<td>2,720</td>
<td>2,737</td>
</tr>
<tr>
<td>backtracking</td>
<td>64</td>
<td>1,825</td>
<td>2,710</td>
<td>3,143</td>
</tr>
<tr>
<td>non-backtracking</td>
<td>416</td>
<td>1,756</td>
<td>2,608</td>
<td>2,929</td>
</tr>
</tbody>
</table>
Results

- 4,869 problems;
- 3,598 problems (about 74% of all problems) were solved by at least one strategy;
- 1,053 problems were solved by all 505 strategies;
- this gives us 2,545 “interesting” problems.

All selected problems

<table>
<thead>
<tr>
<th>splitting</th>
<th>strategies</th>
<th>worst</th>
<th>average</th>
<th>best</th>
</tr>
</thead>
<tbody>
<tr>
<td>off</td>
<td>25</td>
<td>2,708</td>
<td>2,720</td>
<td>2,737</td>
</tr>
<tr>
<td>backtracking</td>
<td>64</td>
<td>1,825</td>
<td>2,710</td>
<td>3,143</td>
</tr>
<tr>
<td>non-backtracking</td>
<td>416</td>
<td>1,756</td>
<td>2,608</td>
<td>2,929</td>
</tr>
</tbody>
</table>

Interesting problems

<table>
<thead>
<tr>
<th>splitting</th>
<th>strategies</th>
<th>worst</th>
<th>average</th>
<th>best</th>
</tr>
</thead>
<tbody>
<tr>
<td>off</td>
<td>25</td>
<td>1,655</td>
<td>1,667</td>
<td>1,684</td>
</tr>
<tr>
<td>backtracking</td>
<td>64</td>
<td>772</td>
<td>1,657</td>
<td>2,090</td>
</tr>
<tr>
<td>non-backtracking</td>
<td>416</td>
<td>703</td>
<td>1,555</td>
<td>1,876</td>
</tr>
</tbody>
</table>
# Best and Worst Strategies

<table>
<thead>
<tr>
<th></th>
<th>worst</th>
<th>best</th>
</tr>
</thead>
<tbody>
<tr>
<td>splitting</td>
<td>nobacktracking on</td>
<td>backtracking</td>
</tr>
<tr>
<td>propositional_to_bdd</td>
<td></td>
<td></td>
</tr>
<tr>
<td>split_at_activation</td>
<td>off</td>
<td>on</td>
</tr>
<tr>
<td>split_goal_only</td>
<td>off</td>
<td>off</td>
</tr>
<tr>
<td>split_input_only</td>
<td>off</td>
<td>off</td>
</tr>
<tr>
<td>split_positive</td>
<td>off</td>
<td>off</td>
</tr>
<tr>
<td>nonliterals_in_clause_weight</td>
<td>off</td>
<td>off</td>
</tr>
<tr>
<td>bdd_marking_subsumption</td>
<td>off</td>
<td>off</td>
</tr>
<tr>
<td>empty_clause_subsumption</td>
<td>on</td>
<td></td>
</tr>
<tr>
<td>sat_solver_for_empty_clause</td>
<td>off</td>
<td></td>
</tr>
<tr>
<td>split_add_ground_negation</td>
<td>off</td>
<td>on</td>
</tr>
</tbody>
</table>
Splitting

Problems solved **only by a single value of an option**

<table>
<thead>
<tr>
<th></th>
<th>nobacktracking</th>
<th>backtracking</th>
</tr>
</thead>
<tbody>
<tr>
<td>off</td>
<td>128</td>
<td>198</td>
</tr>
<tr>
<td>0</td>
<td>128</td>
<td>198</td>
</tr>
</tbody>
</table>
What to split

<table>
<thead>
<tr>
<th></th>
<th>split_at_activation</th>
<th>split_input_only</th>
</tr>
</thead>
<tbody>
<tr>
<td>splitting on</td>
<td>on</td>
<td>off</td>
</tr>
<tr>
<td>backtracking</td>
<td>147</td>
<td>73</td>
</tr>
<tr>
<td>nobacktracking</td>
<td>91</td>
<td>93</td>
</tr>
<tr>
<td>both</td>
<td>145</td>
<td>113</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>split_goal_only</th>
<th>split_positive</th>
</tr>
</thead>
</table>
### What to split

<table>
<thead>
<tr>
<th>split_at_activation</th>
<th>split_input_only</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>splitting</strong></td>
<td><strong>splitting</strong></td>
</tr>
<tr>
<td>Backtracking</td>
<td>147</td>
</tr>
<tr>
<td>Nobacktracking</td>
<td>91</td>
</tr>
<tr>
<td>Both</td>
<td>145</td>
</tr>
</tbody>
</table>

### split_goal_only

<table>
<thead>
<tr>
<th>split_goal_only</th>
<th>split_positive</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>splitting</strong></td>
<td><strong>splitting</strong></td>
</tr>
<tr>
<td>Backtracking</td>
<td>31</td>
</tr>
<tr>
<td>Nobacktracking</td>
<td>21</td>
</tr>
<tr>
<td>Both</td>
<td>17</td>
</tr>
</tbody>
</table>
## What to split

<table>
<thead>
<tr>
<th>split_at_activation</th>
<th>on</th>
<th>off</th>
</tr>
</thead>
<tbody>
<tr>
<td>backtracking</td>
<td>147</td>
<td>73</td>
</tr>
<tr>
<td>nobacktracking</td>
<td>91</td>
<td>93</td>
</tr>
<tr>
<td>both</td>
<td>145</td>
<td>113</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>split_input_only</th>
<th>on</th>
<th>off</th>
</tr>
</thead>
<tbody>
<tr>
<td>backtracking</td>
<td>43</td>
<td>414</td>
</tr>
<tr>
<td>nobacktracking</td>
<td>67</td>
<td>302</td>
</tr>
<tr>
<td>both</td>
<td>33</td>
<td>384</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>split_goal_only</th>
<th>on</th>
<th>off</th>
</tr>
</thead>
<tbody>
<tr>
<td>backtracking</td>
<td>31</td>
<td>155</td>
</tr>
<tr>
<td>nobacktracking</td>
<td>21</td>
<td>207</td>
</tr>
<tr>
<td>both</td>
<td>17</td>
<td>159</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>split_positive</th>
<th>on</th>
<th>off</th>
</tr>
</thead>
<tbody>
<tr>
<td>backtracking</td>
<td>37</td>
<td>262</td>
</tr>
<tr>
<td>nobacktracking</td>
<td>28</td>
<td>146</td>
</tr>
<tr>
<td>both</td>
<td>35</td>
<td>181</td>
</tr>
</tbody>
</table>
### What to split

<table>
<thead>
<tr>
<th>split_at_activation</th>
<th>splitting</th>
<th>on</th>
<th>off</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>backtracking</td>
<td>147</td>
<td>73</td>
</tr>
<tr>
<td></td>
<td>nobacktracking</td>
<td>91</td>
<td>93</td>
</tr>
<tr>
<td></td>
<td>both</td>
<td>145</td>
<td>113</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>split_input_only</th>
<th>splitting</th>
<th>on</th>
<th>off</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>backtracking</td>
<td>43</td>
<td>414</td>
</tr>
<tr>
<td></td>
<td>nobacktracking</td>
<td>67</td>
<td>302</td>
</tr>
<tr>
<td></td>
<td>both</td>
<td>33</td>
<td>384</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>split_goal_only</th>
<th>splitting</th>
<th>on</th>
<th>off</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>backtracking</td>
<td>31</td>
<td>155</td>
</tr>
<tr>
<td></td>
<td>nobacktracking</td>
<td>21</td>
<td>207</td>
</tr>
<tr>
<td></td>
<td>both</td>
<td>17</td>
<td>159</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>split_positive</th>
<th>splitting</th>
<th>on</th>
<th>off</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>backtracking</td>
<td>37</td>
<td>262</td>
</tr>
<tr>
<td></td>
<td>nobacktracking</td>
<td>28</td>
<td>146</td>
</tr>
<tr>
<td></td>
<td>both</td>
<td>35</td>
<td>181</td>
</tr>
</tbody>
</table>
Propositional Part

<table>
<thead>
<tr>
<th>splitting</th>
<th>on</th>
<th>off</th>
</tr>
</thead>
<tbody>
<tr>
<td>off</td>
<td>62</td>
<td>45</td>
</tr>
<tr>
<td>nobacktracking</td>
<td>227</td>
<td>107</td>
</tr>
<tr>
<td>both</td>
<td>226</td>
<td>106</td>
</tr>
</tbody>
</table>
Inference Selection

**nonliterals_in_clause_weight** : in $C\bar{P}$, count not only the weight of $C$, but add some additional weight depending on $P$.

<table>
<thead>
<tr>
<th>splitting</th>
<th>on</th>
<th>off</th>
</tr>
</thead>
<tbody>
<tr>
<td>off</td>
<td>17</td>
<td>11</td>
</tr>
<tr>
<td>nobacktracking</td>
<td>55</td>
<td>45</td>
</tr>
<tr>
<td>nobacktracking</td>
<td>23</td>
<td>62</td>
</tr>
<tr>
<td>all</td>
<td>33</td>
<td>91</td>
</tr>
</tbody>
</table>
Inference Selection

nonliterals_in_clause_weight: in $C\bar{P}$, count not only the weight of $C$, but add some additional weight depending on $P$.

<table>
<thead>
<tr>
<th>splitting</th>
<th>on</th>
<th>off</th>
</tr>
</thead>
<tbody>
<tr>
<td>off</td>
<td>17</td>
<td>11</td>
</tr>
<tr>
<td>nobacktracking</td>
<td>55</td>
<td>45</td>
</tr>
<tr>
<td>nobacktracking</td>
<td>23</td>
<td>62</td>
</tr>
<tr>
<td>all</td>
<td>33</td>
<td>91</td>
</tr>
</tbody>
</table>

splitting_with_blocking: select the introduced negative literal.

<table>
<thead>
<tr>
<th>splitting</th>
<th>on</th>
<th>off</th>
</tr>
</thead>
<tbody>
<tr>
<td>nobacktracking</td>
<td>20</td>
<td>290</td>
</tr>
</tbody>
</table>
Empty Clauses

<table>
<thead>
<tr>
<th>sat_solver_for_empty_clause</th>
<th>bdd_marking_subsumption</th>
</tr>
</thead>
<tbody>
<tr>
<td>splitting</td>
<td>on</td>
</tr>
<tr>
<td>off</td>
<td>8</td>
</tr>
<tr>
<td>nobacktracking</td>
<td>34</td>
</tr>
<tr>
<td>both</td>
<td>34</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sat_solver_with_naming</th>
<th>empty_clause_subsumption</th>
</tr>
</thead>
<tbody>
<tr>
<td>splitting</td>
<td>on</td>
</tr>
<tr>
<td>off</td>
<td>2</td>
</tr>
<tr>
<td>nobacktracking</td>
<td>22</td>
</tr>
<tr>
<td>both</td>
<td>22</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sat_solver_with_subsumption_resolution</th>
</tr>
</thead>
<tbody>
<tr>
<td>splitting</td>
</tr>
<tr>
<td>on</td>
</tr>
<tr>
<td>off</td>
</tr>
<tr>
<td>nobacktracking</td>
</tr>
<tr>
<td>both</td>
</tr>
</tbody>
</table>
Other Options

\[
\text{sat\_solver\_with\_subsumption\_resolution}
\]

<table>
<thead>
<tr>
<th>splitting</th>
<th>on</th>
<th>off</th>
</tr>
</thead>
<tbody>
<tr>
<td>off</td>
<td>8</td>
<td>17</td>
</tr>
<tr>
<td>nobacktracking</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>both</td>
<td>30</td>
<td>30</td>
</tr>
</tbody>
</table>
### Other Options

**sat_solver_with_subsumption_resolution**

<table>
<thead>
<tr>
<th>splitting</th>
<th>on</th>
<th>off</th>
</tr>
</thead>
<tbody>
<tr>
<td>off</td>
<td>8</td>
<td>17</td>
</tr>
<tr>
<td>nobacktracking</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>both</td>
<td>30</td>
<td>30</td>
</tr>
</tbody>
</table>

**split_add_ground_negation**

<table>
<thead>
<tr>
<th>splitting</th>
<th>on</th>
<th>off</th>
</tr>
</thead>
<tbody>
<tr>
<td>backtracking</td>
<td>191</td>
<td>6</td>
</tr>
</tbody>
</table>
Summary

- Calculi for separating the propositional part of clauses;
- Implementation and comparison of two ways of splitting.
- Implementation and comparison of various splitting-related options.