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

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

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 perfomance 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.

Long Clauses

They degrade perfomance 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

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 $C \vee P$;
- RePro is a family of calculi, depending on the underlying resolution calculus.

Generating inferences

For every generating inference

$$\frac{C_1 \quad \cdots \quad C_n}{C}$$

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

$$\frac{C_1|P_1 \quad \cdots \quad C_n|P_n}{C|(P_1 \vee \ldots \vee P_n)} \ .$$

Simplifying inferences

For every simplifying inference

$$\frac{C_1 \quad \cdots \quad C_n \quad \not{D}}{C}$$

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

$$\frac{C_1|P_1 \quad \cdots \quad C_n|P_n \quad \not{D}|\not{P}}{C|(P_1 \vee \ldots \vee P_n)} \ .$$

Deletion inferences

For every deletion inference

$$C_1 \quad \cdots \quad C_n \quad \cancel{D}$$

of the resolution calculus, if $P_1 \vee \ldots \vee P_n \to P$ is a tautology, then the following is a deletion inference of RePro:

$$C_1|P_1 \quad \cdots \quad C_n|P_n \quad \cancel{D|P}.$$

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.

More Rules

Propositional tautology deletion:

$$\cancel{D \mid P},$$

where P is a propositional tautology.

More Rules

Propositional tautology deletion:

$$\cancel{D|P},$$

where P is a propositional tautology.

The merge rule of RePro:

$$\frac{\cancel{C|P_1} \quad \cancel{C|P_2}}{C|(P_1 \wedge P_2)}$$

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

More Rules

Propositional tautology deletion:

$$\cancel{D|P},$$

where P is a propositional tautology.

The merge rule of RePro:

$$\frac{\cancel{C|P_1} \quad \cancel{C|P_2}}{C|(P_1 \wedge P_2)}$$

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

The merge subsumption rule:

$$\frac{C|P_1 \quad \cancel{D|P_2}}{D|(P_1 \wedge P_2)} \ ,$$

where C subsumes D.

Alternative Formulation

For every simplifying inference

$$\frac{C_1 \quad \cdots \quad C_n \quad \cancel{D}}{C}$$

of the resolution calculus, consider

$$\frac{C_1|P_1 \quad \cdots \quad C_n|P_n \quad \cancel{D|P}}{C|(P_1 \vee \ldots \vee P_n) \quad D|(P_1 \vee \ldots \vee P_n \rightarrow P)} \ .$$

Alternative Formulation

For every simplifying inference

$$\frac{C_1 \quad \cdots \quad C_n \quad \not{D}}{C}$$

of the resolution calculus, consider

$$\frac{C_1|P_1 \quad \cdots \quad C_n|P_n \quad \not{D|P}}{C|(P_1 \vee \ldots \vee P_n) \quad D|(P_1 \vee \ldots \vee P_n \rightarrow P)} \;.$$

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

Alternative Formulation

For every simplifying inference

$$\frac{C_1 \quad \cdots \quad C_n \quad \not{D}}{C}$$

of the resolution calculus, consider

$$\frac{C_1|P_1 \quad \cdots \quad C_n|P_n \quad D|\not{P}}{C|(P_1 \vee \ldots \vee P_n) \quad D|(P_1 \vee \ldots \vee P_n \rightarrow P)} \; .$$

The previously defined simplifying rule is a special case of this one, since, if $P_1 \vee \ldots \vee P_n \rightarrow P$ 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 $A \vee B|(p \wedge q)$ is redundant in the presence if $A|p$ and $B|q$

Advantages?

A clause $A \vee B | (p \wedge q)$ is redundant in the presence if $A|p$ and $B|q$ using the following sequence of deletion rules:

$$\dfrac{B|q \qquad \dfrac{A|p \qquad \cancel{A \vee B | (p \wedge q)}}{\cancel{A \vee B | (p \rightarrow (p \wedge q))}}}{A \vee B | (q \rightarrow (p \rightarrow (p \wedge q)))}$$

whose conclusion is a tautology.

Outline

Observation

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

Observation

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

Then the set $S \cup \{C_1 \vee C_2\}$ 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 $C \stackrel{\text{def}}{=} C_1 \vee \ldots \vee C_n$ is splittable into components $C_1, \ldots, C_n$.

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 $C \stackrel{\text{def}}{=} C_1 \vee \ldots \vee C_n$ 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.

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 $C \stackrel{\text{def}}{=} C_1 \vee \ldots \vee C_n$ 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 lof of work upon backtracking.

Splitting Without Backtracking

$$\frac{\cancel{C_1 \vee C_2}}{C_1 \vee p \qquad C_2 \vee \neg p},$$

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.

Splitting Without Backtracking

$$\frac{\cancel{C_1 \vee C_2}}{C_1 \vee p \qquad C_2 \vee \neg p} \; ,$$

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.

$$\frac{\cancel{C_1 \vee C_3}}{C_3 \vee \neg p} \; ,$$

if this rule was previously applied to C_1.

Splitting Without Backtracking

$$\frac{\cancel{C_1 \vee C_2}}{C_1 \vee p \qquad C_2 \vee \neg p}\ ,$$

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.

$$\frac{\cancel{C_1 \vee C_3}}{C_3 \vee \neg p}\ ,$$

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.

Options Related to Splitting and Propositional Literals

- 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

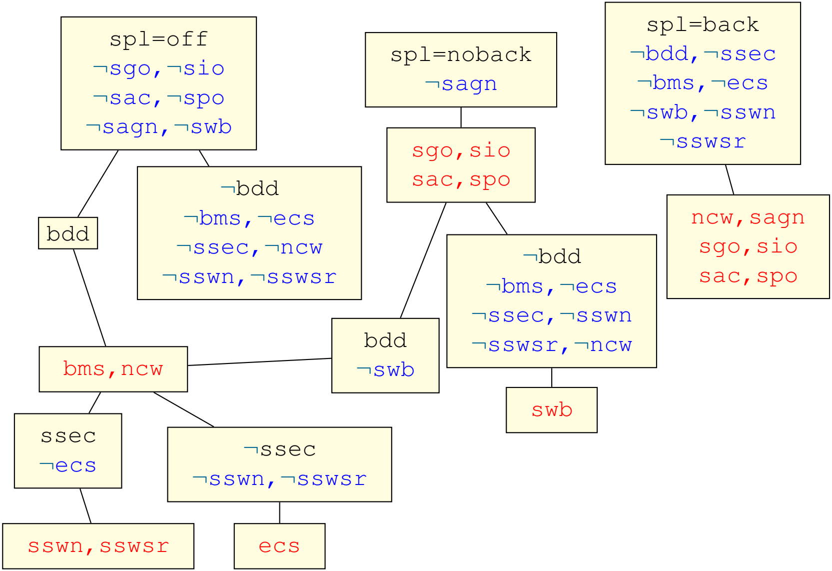
(Without backtracking).

- `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.

Dependency Tree



Outline

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.

Outline

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

splitting	strategies	worst	average	best
off	25	2,708	2,720	2,737
backtracking	64	1,825	2,710	3,143
non-backtracking	416	1,756	2,608	2,929

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

splitting	strategies	worst	average	best
off	25	2,708	2,720	2,737
backtracking	64	1,825	2,710	3,143
non-backtracking	416	1,756	2,608	2,929

Interesting problems

splitting	strategies	worst	average	best
off	25	1,655	1,667	1,684
backtracking	64	772	1,657	2,090
non-backtracking	416	703	1,555	1,876

Best and Worst Strategies

	worst	best
splitting	nobacktracking	backtracking
propositional_to_bdd	on	
split_at_activation	off	on
split_goal_only	off	off
split_input_only	off	off
split_positive	off	off
nonliterals_in_clause_weight	off	off
bdd_marking_subsumption	off	
empty_clause_subsumption	on	
sat_solver_for_empty_clause	off	
split_add_ground_negation		on

Splitting

Problems solved only by a single value of an option

off	nobacktracking	backtracking
0	128	198

What to split

split_at_activation

splitting	on	off
backtracking	147	73
nobacktracking	91	93
both	145	113

split_input_only

split_goal_only

split_positive

What to split

split_at_activation

splitting	on	off
backtracking	147	73
nobacktracking	91	93
both	145	113

split_input_only

split_goal_only

splitting	on	off
backtracking	31	155
nobacktracking	21	207
both	17	159

split_positive

What to split

split_at_activation

splitting	on	off
backtracking	147	73
nobacktracking	91	93
both	145	113

split_input_only

splitting	on	off
backtracking	43	414
nobacktracking	67	302
both	33	384

split_goal_only

splitting	on	off
backtracking	31	155
nobacktracking	21	207
both	17	159

split_positive

What to split

split_at_activation

splitting	on	off
backtracking	147	73
nobacktracking	91	93
both	145	113

split_input_only

splitting	on	off
backtracking	43	414
nobacktracking	67	302
both	33	384

split_goal_only

splitting	on	off
backtracking	31	155
nobacktracking	21	207
both	17	159

split_positive

splitting	on	off
backtracking	37	262
nobacktracking	28	146
both	35	181

Propositional Part

propositional_to_bdd:

splitting	on	off
off	62	45
nobacktracking	227	107
both	226	106

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	on	off
off	17	11
nobacktracking	55	45
nobacktracking	23	62
all	33	91

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	on	off
off	17	11
nobacktracking	55	45
nobacktracking	23	62
all	33	91

`splitting_with_blocking`: select the introduced negative literal.

splitting	on	off
nobacktracking	20	290

Empty Clauses

sat_solver_for_empty_clause

splitting	on	off
off	8	5
nobacktracking	34	21
both	34	21

bdd_marking_subsumption

splitting	on	off
off	62	45
nobacktracking	227	107
both	226	106

sat_solver_with_naming

splitting	on	off
off	2	0
nobacktracking	22	0
both	22	0

empty_clause_subsumption

splitting	on	off
off	5	7
nobacktracking	18	46
both	18	46

sat_solver_with_subsumption_resolution

splitting	on	off
off	2	1
nobacktracking	1	2
both	2	2

Other Options

sat_solver_with_subsumption_resolution

splitting	on	off
off	8	17
nobacktracking	30	30
both	30	30

Other Options

sat_solver_with_subsumption_resolution

splitting	on	off
off	8	17
nobacktracking	30	30
both	30	30

split_add_ground_negation

splitting	on	off
backtracking	191	6

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.