Splitting through New Proposition Symbols

Hans de Nivelle

Max Planck Institut für Informatik
Stuhlsatzenhausweg 85
66123 Saarbrücken, Germany
nivelle@mpi-sb.mpg.de

Abstract. The splitting rule is a tableau-like rule, that is used in the resolution context. In case the search state contains a clause $C_1 \vee C_2$, which has no shared variables between C_1 and C_2, the prover splits the search state, and tries to refute C_1 and C_2 separately.
Instead of splitting the state of the theorem prover, one can create a new proposition symbol α, and replace $C_1 \vee C_2$ by $C_1 \vee \alpha$ and $\neg\alpha \vee C_2$. In the first clause α is the least preferred literal. In the second clause α is selected. In this way, nothing can be done with C_2 as long as C_1 has not been refuted.
This way of splitting simulates search state splitting only partially, because a clause that inherits from $C_1 \vee \alpha$ cannot subsume or simplify a clause that does not inherit from C_1. With search state splitting, a clause that inherits from C_1 can in principle subsume or simplify clauses that do not derive from C_1. As a consequence, splitting through new symbols is less powerfull than search state splitting. In this paper, we present a solution for this problem.

1 Introduction

It is an empirical fact that methods with state splitting are better on propositional problems than resolution-like methods. When there are variables, state splitting becomes difficult due to the fact that it is necessary to maintain shared variables between states. It seems reasonable to keep as much as possible from state splitting, but avoiding the problems with shared variables.

The *splitting rule* is the following rule: Suppose that the search state of a resolution theorem prover contains a clause $C_1 \vee C_2$, where both C_1 and C_2 are non-trivial and have no variables in common. In that case the prover splits its search state into two states, one for C_1 and one for C_2. When properly applied, the splitting rule can improve the chance of finding a proof significantly. In addition to increased efficiency when finding a proof, the splitting rule increases the chance of finding a saturation. This is due to the fact that C_1 or C_2 may backward subsume clauses that are not subsumed by $C_1 \vee C_2$. Similarly it may be possible to derive an equality from C_1 or C_2 that simplifies the search state significantly, but which could not be derived from $C_1 \vee C_2$.

Some resolution decision procedures rely on the splitting rule. Examples are the procedure for E^+ of [FLTZ93] and [dN94], and the procedure for the 2-variable fragment of [dNPH01].

The splitting rule is practically useful, but difficult to implement in the resolution context. Currently (2001), Spass ([Wb01]) is the only implementation of resolution with search state splitting. It is not practical to make two copies of the search state. Therefore splitting has to be implemented through backtracking. The system deletes $C_1 \vee C_2$ from the search state, and replaces it by C_1. After that, search continues until either a saturation or a proof is found. If it finds a saturation then the system can terminate. If the empty clause is derived, then the system has to backtrack to the point on which C_1 was introduced, and replace it by C_2. In order to be able to restore the search state, the theorem prover has to do a complex administration. In particular, when a clause is deleted because it subsumed or rewritten, it has to be be done in such a way that it can be restored.

Because of these implementation difficulties, some resolution provers implement the following approximation: If the clause $C_1 \vee C_2$ can be split, then it can be replaced by the two clauses $C_1 \vee \alpha$ and $\neg\alpha \vee C_2$. Here α is a new propositional atom. It is ensured that α ist the least preferred literal in $C_1 \vee \alpha$, and that $\neg\alpha$ is selected in $\neg\alpha \vee C_2$. The only way in which $\neg\alpha \vee C_2$ can be used is by resolving it with a clause in which α is maximal. Such a clause will be derived only when C_1 is refuted. This way of splitting has been implemented in the Saturate System ([GNN98]), and in Vampire ([RV01]). We call this way of splitting *splitting through new symbols.*

Structural clause transformations ([NW01], [BFL94]) are closely related to splitting through new symbols. Assume that $F_1 \vee F_2$ is *antiprenexed*, i.e. that quantifiers are factored inwards as much as possible. In that case, every free variable of F_1 or F_2 is a free variable of both F_1 and F_2. A structural clause transformation would replace $F_1 \vee F_2$ by $F_1 \vee \alpha(\overline{x})$ and $\forall x(\alpha(\overline{x}) \rightarrow F_2)$. Here $\overline{x}$ are the free variables of F_2 and α is a new predicate symbol. In case F_1 and F_2 have no shared variables, symbol α will be propositional.

Unfortunately, splitting through new symbols only partially simulates search state splitting, because the new symbols hinder backward subsumption and simplification. Every clause that inherits from C_1 contains the literal α. Because of this, it cannot simplify or subsume a clause that does not inherit from C_1. This makes that splitting through new symbols fails to have one of the major advantages of splitting. In this paper, we give the following solution for this problem: If there are two clauses $C_1 \vee \alpha$ and D, and C_1 subsumes D, then we replace D by $\neg\alpha \vee D$. In this way D is switched off, until α has been derived.

Simplification can be handled in the same way. Suppose that C_1 can simplify the clause D_1 into D_2, but that the prover has the clause $C_1 \vee \alpha$ instead of C_1 in its search state. Then D_1 is replaced by $D_2 \vee \alpha$, and $\neg\alpha \vee D_1$. Doing this, the simplification D_2 is available on the current branch, and D_1 is switched off until α has been derived. We give an example:

Example 1. Suppose that one wants to refute the following clause set

$$p(0), \quad \neg p(s^{16}(0)), \quad s(s(X)) \approx X \vee s(Y) \approx Y,$$

and assume that one is using the Knuth-Bendix order. The equalities in the last clause are uncomparable under the Knuth-Bendix order, so both equalities have to be used for paramodulation.

The last clause can be split into $s(s(X)) \approx X$ and $s(Y) \approx Y$. Splitting results in the following two states, both of which have a trivial refutation using simple rewriting and one resolution step. The states are:

$$p(0), \quad \neg p(s^{16}(0)), \quad s(s(X)) \approx X,$$

and

$$p(0), \quad \neg p(s^{16}(0)), \quad s(Y) \approx Y.$$

Instead of search state splitting, one can split through a new symbol. Using α as new symbol, the last clause can be replaced by

$$s(s(X)) \approx X \vee \alpha \text{ and } \neg\alpha \vee s(Y) \approx Y.$$

After that, α can be derived by iterated paramodulation from $s(s(X)) \approx X \vee \alpha$. When this is done, $s(Y) \approx Y$ becomes available again. This equality will simplify $\neg p(s^{16}(0))$ into $\neg p(0)$.

In Example 1, splitting through new symbols is better than no splitting at all, but it is still not as good as state splitting. In the first search state, $s(s(X)) \approx X$ can simplify $\neg p(s^{16}(0))$ into $\neg p(0)$. However $s(s(X)) \approx X \vee \alpha$ cannot simplify $\neg p(s^{16}(0))$, because of the extra literal α.
In the following example, we use extended backward simplification:

Example 2. With extended backward simplification, Example 1 is handled as follows: After the split, the search state consists of the clauses

$$p(0), \quad \neg p(s^{16}(0)), \quad s(s(X)) \approx X \vee \alpha, \quad \neg\alpha \vee s(X) \approx X.$$

With $s(s(X)) \approx X \vee \alpha$, the clause $\neg p(s^{16}(0))$ is simplified into

$$\neg p(s^{14}(0)) \vee \alpha \text{ and } \neg\alpha \vee \neg p(s^{16}(0)).$$

Seven more rewrites result in

$$\neg p(0) \vee \alpha \text{ and } \neg\alpha \vee \neg p(s^{16}(0)).$$

Now the complete search state consists of the clauses

$$p(0), \quad \neg p(0) \vee \alpha, \quad s(s(X)) \approx X \vee \alpha,$$

$$\neg\alpha \vee \neg p(s^{16}(0)), \quad \neg\alpha \vee s(X) \approx X.$$

The first two clauses resolve into the clause α. After that, the two last clauses resolve with α, which results in the clauses

$$\neg p(s^{16}(0)) \text{ and } s(X) \approx X.$$

Now $\neg p(s^{16}(0))$ is simplified into $\neg p(0)$, which resolves with $p(0)$ into the empty clause.

In order to handle cases where more than one splitting symbol is missing, it is necessary to extend the clause format. Suppose that C_1 subsumes C_2, but the search state contains $C_1 \vee \alpha \vee \beta$. In that case C_2 has to be restored, when either of α or β is derived. One could obtain this by replacing C_2 by $\neg\alpha \vee C_2$ and $\neg\beta \vee C_2$. However, when there are many splitting symbols, this would result in too many copies. For this reason, it is better to extend the clause format by allowing negated disjunctions of splitting symbols. Using the extension, C_2 can be replaced by $\neg(\alpha \vee \beta) \vee C_2$. Each of the symbols α or β can restore C_2. Note that only a minor extension of the clause format is needed. Negated disjunctions need to be added only for splitting symbols, not for usual atoms. Both the positive disjunctions of splitting literals, and the negative disjunctions can be easily represented by bit strings, so they can be manipulated very efficiently.

If one does not want to extend the clause format, it is also possible to replace C_2 by

$$\neg\ \gamma \vee C_2, \neg\alpha \vee \gamma, \neg\beta \vee \gamma,$$

where γ is a new literal. However we think that it is better to make a small extension to the clause format.

In the next section we formally define search state splitting, and we give a couple of variants of splitting through new propositional symbols. After that, in Section 3, we prove a general completeness result, that is general enough to prove completeness of all types of splitting, combined with all restrictions of resolution.

2 Splitting through New Symbols

We first define two variants of the splitting rule, usual splitting and extended splitting. It is possible to add the negation of one of the branches in the other branch. This is called *extended splitting.* Extended splitting is possible because $C_1 \vee C_2$ is logically equivalent with $C_1 \vee (C_2 \wedge \neg C_1)$.

Definition 1. *Let C be a clause, not the empty clause. Let A be a literal in C. The* component *of A in C is the smallest subset C_1 of C, s.t*

- *$A \in C_1$, and*
- *if literals B_1, B_2 have some variables in common, $B_1 \in C_1$, $B_2 \in C$, then $B_2 \in C_1$.*

Write $C = C_1 \vee C_2$, where C_2 are the literals of C that are not in C_1. If C_2 is non-empty, then clause $C_1 \vee C_2$ can be split. Write Γ for the remaining clauses in the search state of the theorem prover.

- Splitting *is obtained by replacing the search state $\Gamma, C_1 \vee C_2$ by two search states Γ, C_1 and Γ, C_2.*
- Extended splitting *is defined by replacing the search state by the two search states Γ, C_1 and $\Gamma, \neg C_1, C_2$. In the second proof state, variables in $\neg C_1$ have to be Skolemized.*

The original search state is refuted if both of the split search states are refuted. The original search state has a saturation if one of the split search states has a saturation.

Note that our definition implies that C_1 cannot be split another time. It is possible however that C_2 can be split another time. In practive, one does not always split when it is possible, because search state splitting is costly. Spass uses heuristics to decide whether or not a possible split should take place, and whether extended or simple splitting should be used. Typically both components need to contain non-propositional, positive literals. It is reported in [Wb01] that extended splitting is not always better than non-extended splitting. See [Wb01] for details. Next we define splitting through new symbols. We do this in two ways. The first way is the way that we described in the introduction. The second way tries to reduce the number of splitting symbols by exploiting dependencies between them. We call this splitting with *literal suppression.*

Definition 2. *We first define splitting without literal suppression. Let $C_1 \vee C_2$ be a clause that can be split.*

- *Non-extended splitting replaces the clause by*

$$C_1 \vee \beta, \quad \neg\beta \vee C_2.$$

- *Extended splitting replaces the clause by*

$$C_1 \vee \beta, \quad \neg\beta \vee C_2, \quad \neg\beta \vee \neg C_1.$$

  The negation of C_1 has to be Skolemized.

If some clause has more than one split clause among its parents, it will contain more than one splitting symbol. If one would split such a clause using Definition 2, then one more splitting symbol will be added. The following way of splitting makes it possible to drop other splitting atoms when a clause is split, that already contains splitting atoms.

Definition 3. *Let $C_1 \vee C_2 \vee \alpha_1 \vee \cdots \vee \alpha_p$ be a clause that can be split, where $\alpha_1, \ldots, \alpha_p$ $p \geq 0$ are positive splitting symbols, resulting from earlier splits. We define splitting with* symbol suppression.

- *Simple splitting replaces the clause by*

$$C_1 \vee \beta, \quad \neg\beta \vee C_2 \vee \alpha_1 \vee \cdots \vee \alpha_p,$$

$$\neg\alpha_1 \vee \beta, \ldots, \neg\alpha_p \vee \beta.$$

- *Extended splitting replaces the clause by*

$$C_1 \vee \beta, \quad \neg\beta \vee C_2 \vee \alpha_1 \vee \cdots \vee \alpha_p, \quad \neg\beta \vee \neg C_1 \vee \alpha_1 \vee \cdots \vee \alpha_p,$$

$$\neg\alpha_1 \vee \beta, \cdots, \neg\alpha_p \vee \beta.$$

It may seem that the effect of splitting with literal suppression can be also obtained by splitting $C_1 \vee C_2 \vee \alpha_1 \vee \cdots \vee \alpha_p$ into $C_1 \vee \beta$ and $\neg\beta \vee C_2 \vee \alpha_1 \vee \cdots \vee \alpha_p$. However if the first clause resolves with a clause containing some of the α_i, then both the α_i and β have to be kept in the resulting clause. If one has literal supression, then any clause of form $D \vee \alpha_{i_1} \vee \cdots \vee \alpha_{i_k} \vee \beta$ can be simplified into $D \vee \beta$, due to presence of the $\neg\alpha_i \vee \beta$ clauses.

For all four ways of splitting, it can be easily verified that the clauses resulting from the split imply the original clause. This does not imply completeness for the case where splitting is done eagerly, but it does imply that splitting with fresh literals can be done finitely often without losing completeness.

We will now prove that the four ways of splitting are sound. We do this by proving that in each case there exists a first order formula F, which can be substituted for β, such that the resulting clauses become logical consequences of the original clause. This makes the splitting rules provably sound in higher order logic. This makes it possible to verify resolution proofs that use splitting through new symbols, see [dN01]. In first order logic, the splitting rules are satisfiability preserving. Instead of substituting F for β, one can extend the interpretation with β, and copy the truth value for β from F.

Theorem 1. *For all four ways of splitting, there exists a formula F which can be substituted for β, s.t. the resulting clauses are logical consequences of the original clause.*

Proof.

- First we consider splitting without literal suppression. For non-extended splitting, one can simply take

$$\beta := C_2.$$

  For extended splitting, one can take

$$\beta := \neg C_1 \wedge C_2.$$

- Next we consider splitting with literal suppression. For non-extended splitting, one can take

$$\beta := C_2 \vee \alpha_1 \vee \cdots \vee \alpha_p.$$

  Substituting in the formulas that result from the split, gives

$$C_1 \vee (C_2 \vee \alpha_1 \vee \cdots \vee \alpha_p),$$

$$\neg(C_2 \vee \alpha_1 \vee \cdots \vee \alpha_p) \vee C_2 \vee \alpha_1 \vee \cdots \vee \alpha_p,$$

$$\neg\alpha_1 \vee (C_2 \vee \alpha_1 \vee \cdots \vee \alpha_p), \ldots, \neg\alpha_p \vee (C_2 \vee \alpha_1 \vee \cdots \vee \alpha_p).$$

  The first formula equals $C_1 \vee C_2 \vee \alpha_1 \vee \cdots \vee \alpha_p$. The other formulae are tautologies.
  In the case of extended splitting with literal suppression, one can take

$$\beta := (\neg C_1 \wedge C_2) \vee \alpha_1 \vee \cdots \vee \alpha_p.$$

Substituting in the formulas, resulting from the split, gives

$$C_1 \vee (\neg C_1 \wedge C_2) \vee \alpha_1 \vee \cdots \vee \alpha_p,$$
$$\neg((\neg C_1 \wedge C_2) \vee \alpha_1 \vee \cdots \vee \alpha_p) \vee C_2 \vee \alpha_1 \vee \cdots \vee \alpha_p,$$
$$\neg((\neg C_1 \wedge C_2) \vee \alpha_1 \vee \cdots \vee \alpha_p) \vee \neg C_1 \vee \alpha_1 \vee \cdots \vee \alpha_p,$$
$$\neg\alpha_1 \vee (\neg C_1 \wedge C_2) \vee \alpha_1 \vee \cdots \vee \alpha_p, \cdots, \neg\alpha_p \vee (\neg C_1 \wedge C_2) \vee \alpha_1 \vee \cdots \vee \alpha_p.$$

It is easily checked that all of these formulas are tautologies or logical consequences of $C_1 \vee C_2 \vee \alpha_1 \vee \cdots \vee \alpha_p$.

3 A Meta Calculus

We prove a general completeness result, which applies to all splitting strategies described so far. We do not want the result to be restricted to one calculus, or to one type of redundancy. The completeness result has to be applicable to all resolution decision procedures that need the splitting rule, and also to other refinements of resolution, that are used for full first order.

One could try to give separate completeness proofs for the various calculi, but this is too complicated. The completeness proofs are rather heterogeneous. Some of them rely on the completeness of the superposition calculus ([BG01]), others are based on the resolution game ([dN94]) or lock resolution. ([B71])

In order to obtain a general completeness result, we define a *meta calculus* which extends some refinement of resolution (called *calculus* here) by adjoining the splitting atoms to it. We then prove relative completeness: Every derivation, that fulfills certain conditions, will either derive the empty clause, or construct in the limit a saturation of the original calculus.

The rules of the meta calculus are obtained by modifying the rules of the original calculus. When a rule of the original calculus is applied on clauses containing splitting symbols, the resulting clause inherits the splitting symbols from the parents in the meta calculus. Redundancy in the meta calculus is obtained by combining the redundancy of the original calculus with propositional implication on the splitting symbols.

It is necessary to keep the splitting symbols apart from the calculus literals. Using the propositional redundancy techniques in full generality on all literals would result in incompleteness.

Definition 4. *We identify a calculus $\mathcal{C}$ by its derivation rules and its redundancy rules. A* calculus *is characterized by an ordered tuple $\mathcal{C} = (A, D, R, e)$ in which*

- *A is the set of clauses,*
- *$D \subseteq A^* \times A$ is the set of derivation rules,*
- *$R \subseteq A^* \times A$ is the set of redundancy rules. R must be* reflexive *and* transitive. *Reflexive means that $R(a, a)$ for all $a \in A$. Transitive means that $R(a_1, \ldots, a_n, a)$ and $R(b_1, \ldots, b_{i-1}, a, b_{i+1}, \ldots, b_m, b)$ imply*

$$R(b_1, \ldots, b_{i-1}, a_1, \ldots, a_n, b_{i+1}, \ldots, b_m, b).$$

– *$e \in A$ is the empty clause of $\mathcal{C}$.*

In examples, clauses of $\mathcal{C}$ will be between square brackets, to stress that we see them as closed objects. For example, if one would have ordinary resolution with subsumption, one would have

$$([p(X) \vee q(X)], [\neg p(X) \vee r(Y)]; [q(X) \vee r(Y)]) \in D,$$

$$([p(X,Y) \vee p(Y,X) \vee r(X,Y)]; [p(X,X) \vee r(X,X)] \in D.$$

In R, one would have

$$([p(X) \vee q(X)]; [p(0) \vee q(0)]) \in R,$$

$$([p(X)]; [p(f(X))]) \in R.$$

The calculus $\mathcal{C}$ is considered on the predicate level. Clauses are not replaced by their ground instances. However, since the clauses of $\mathcal{C}$ are closed objects, the meta calculus is a propositional calculus.

All natural redundancy criteria are transitive, because they are based on implication and on some ordering conditions.

There is no need to specify what the splitting rules of $\mathcal{C}$ exactly are. The reason for this fact is that, as far as completeness is concerned, splitting can be handled by redundancy. When a clause $C_1 \vee C_2$ is split into C_1 and C_2, both of the components subsume the original clause.

The method can handle any form of splitting, as long as the clauses obtained by splitting subsume the split clause. We are not concerned about soundness in this section. The soundness of most of the possible ways of splitting has been proven in Theorem 1.

Definition 5. *A* saturated set *is a set $M \subseteq A$, such that for each set of clauses $a_1, \ldots, a_n \in M$, $(n \geq 0)$ and clause $a \in A$, for which*

$$D(a_1, \ldots, a_n, a),$$

there are clauses $b_1, \ldots, b_m \in M$, $(m \geq 0)$ such that

$$R(b_1, \ldots, b_m, a).$$

A set $M \subseteq A$ is a saturation of *some set of initial clauses I if M is a saturated set, and for each $a \in I$, there are clauses $a_1, \ldots, a_m \in M$, such that $R(a_1, \ldots, a_m, a)$.*

We use the letter M for saturations, because they play the role of models. If the calculus $\mathcal{C}$ is complete, then M (in principle) represents a model of the clauses it contains.
We now extend calculus $\mathcal{C}$ with splitting atoms:

Definition 6. *Let $\mathcal{C} = (A, D, R, e)$ be a calculus. Let $(\Sigma, \prec)$ be a well-ordered set of propositional atoms, non-overlapping with any object in A. Clauses of the* extended *calculus $\mathcal{C}^\Sigma$ have form*

$$(\neg\sigma_1 \vee \cdots \vee \neg\sigma_p) \vee a \vee \tau.$$

It must be the case that $a \in A$. It is possible that $a = e$. Each σ_i $(1 \leq i \leq p)$ is a disjunction of splitting literals. If $p > 0$, then the clause is blocked by the sequence $(\neg\sigma_1, \ldots, \neg\sigma_p)$. If $p = 0$, then the clause is not blocked. When the clause is not blocked, we write $a \vee \tau$ instead of $(\ ) \vee a \vee \tau$.

The τ is a disjunction of splitting atoms, representing the splitting context. We assume that τ is sorted by $\prec$, with the maximal atom first, and that repeated splitting atoms are deleted. If τ is empty, we omit it from the clause.
Similarly we write $(\neg\sigma_1 \vee \cdots \vee \neg\sigma_p) \vee \tau$ instead of $(\neg\sigma_1 \vee \cdots \vee \neg\sigma_p) \vee e \vee \tau$.
The empty clause of the extended calculus is the clause

$$(\ ) \vee e \vee \bot,$$

where $\bot$ is the empty disjunction.

Next we define the rules of the calculus $\mathcal{C}^\Sigma$.

Definition 7. *The derivation rules D^Σ of the extended calculus are defined as follows:*

CONTEXT: *If $D(a_1, \ldots, a_n, a)$ in the original calculus $\mathcal{C}$, then for all splitting contexts $\tau_1, \ldots, \tau_n$,*

$$D^\Sigma(a_1 \vee \tau_1, \ldots, a_n \vee \tau_n, a \vee (\tau_1 \vee \cdots \vee \tau_n)\ ).$$

On clauses that are not blocked, we simply apply the rules of $\mathcal{C}$. The splitting contexts of the parents are collected into the splitting context of the new clause.

RESTORE: *Let*

$$c = (\neg\sigma_1 \vee \cdots \vee \neg\sigma_p) \vee a \vee \tau$$

be a blocked $\mathcal{C}^\Sigma$-clause with $p > 0$. For each i, $1 \leq i \leq p$, let the clause c_i have form $\alpha_i \vee \tau_i$. (It consists only of splitting atoms) It must be the case that α_i is the maximal splitting atom of the clause $\alpha_i \vee \tau_i$. If each α_i occurs in σ_i, then we put $D^\Sigma(c, c_1, \ldots, c_p, d)$ for the $\mathcal{C}^\Sigma$-clause

$$d = a \vee (\tau \vee \tau_1 \vee \cdots \vee \tau_p).$$

Definition 8. *The redundancy rule R^Σ is defined as follows: Assume that*

$$R(a_{1,1}, \ldots, a_{1,m_1}, b_1), \ldots, R(a_{k,1}, \ldots, a_{k,m_k}, b_k)$$

in the original calculus $\mathcal{C}$. If

$$\neg a_{1,1} \vee \cdots \vee \neg a_{1,m_1} \vee b_1, \ldots, \neg a_{k,1} \vee \cdots \vee \neg a_{k,m_k} \vee b_k,\ c_1, \ldots, c_n \models c$$

in propositional logic, treating the clauses $a_{i,j}$, b_i $(1 \leq i \leq k,\ 1 \leq j \leq m_i)$ as propositional atoms, then

$$R^\Sigma(c_1, \ldots, c_n, c).$$

What this rule says is that the calculus $\mathcal{C}^{\Sigma}$ inherits the redundancy from $\mathcal{C}$ through propositional implication on the splitting literals. We give a couple of examples in order to show that Definition 8 does what it is supposed to do:

Example 3. We show how R^{Σ} handles subsumption. Let $\mathcal{C}$ be the simple resolution calculus. Clause $p(X) \vee q(X)$ subsumes $p(s(X)) \vee q(s(X))$.
Then $[p(X) \vee q(X)]$ makes $[p(s(X)) \vee q(s(X))]$ redundant in R^{Σ}, because

$$\neg[p(X) \vee q(X)] \vee [p(s(X)) \vee q(s(X))], [p(X) \vee q(X)] \models [p(s(X)) \vee q(s(X))].$$

Similarly $[p(X) \vee q(X)] \vee \alpha$ makes $[p(s(X)) \vee q(s(X))] \vee \alpha$ redundant, because of the propositional implication

$$\neg[p(X) \vee q(X)] \vee [p(s(X)) \vee q(s(X))],\ [p(X) \vee q(X)] \vee \alpha \models$$

$$[p(s(X)) \vee q(s(X))] \vee \alpha.$$

In the presence of $[p(X) \vee q(X)] \vee \alpha$, it is possible to replace $[p(s(X)) \vee q(s(X))]$ by $\neg\alpha \vee [p(s(X)) \vee q(s(X))]$, because $[p(X) \vee q(X)] \vee \alpha$ and $\neg\alpha \vee [p(s(X)) \vee q(s(X))]$ make $[p(s(X)) \vee q(s(X))]$ redundant in R^{Σ}. This fact follows from the following propositional implication

$$\neg[p(X) \vee q(X)] \vee [p(s(X)) \vee q(s(X))],$$

$$[p(X) \vee q(X)] \vee \alpha,\ \neg\alpha \vee [p(s(X)) \vee q(s(X))] \models [p(s(X)) \vee q(s(X))].$$

The following example demonstrates how R^{Σ} handles simplification:

Example 4. If $\mathcal{C}$ is the superposition calculus, then the clauses

$$c_1 = [s(X) \approx X] \vee \alpha \text{ and } c_2 = [t(Y) \approx Y] \vee \beta$$

can simplify $d = [p(s(X), t(Y))]$ into

$$d_1 = [p(X, Y)] \vee \alpha \vee \beta \text{ and } d_2 = \neg(\alpha \vee \beta) \vee [p(s(X), t(Y))].$$

In order to justify this simplification, we need to show that c_1, c_2, d_1, d_2 make d redundant in $\mathcal{C}^{\Sigma}$. This follows from the implication

$$\neg[s(X) \approx X] \vee \neg[t(Y) \approx Y] \vee \neg[p(X, Y)] \vee [p(s(X), t(Y))],$$

$$[s(X) \approx X] \vee \alpha,\ [t(Y) \approx Y] \vee \beta,$$

$$[p(X, Y)] \vee \alpha \vee \beta, \neg(\alpha \vee \beta) \vee [p(s(X), t(Y))] \models$$

$$[p(s(X), t(Y))].$$

The following example shows how Definition 8 handles splitting through fresh literals.

Example 5. Suppose we want to split $p(X) \vee q(Y)$. Both $p(X)$ and $q(Y)$ make $p(X) \vee q(Y)$ redundant in the original calculus $\mathcal{C}$. Because of the implication

$$\neg[p(X)] \vee [p(X) \vee q(Y)],\ \neg[q(Y)] \vee [p(X) \vee q(Y)],$$

$$[p(X)] \vee \alpha,\ \neg\alpha \vee [q(Y)] \ \models$$

$$[p(X) \vee q(Y)],$$

the clauses $p(X) \vee \alpha$ and $\neg\alpha \vee q(Y)$ make $p(X) \vee q(Y)$ redundant in R^{Σ}.

The last example gives a simplification that is allowed by Definition 8:

Example 6. Suppose there is a clause

$$(\neg\sigma_1 \vee \cdots \vee \neg\sigma_p) \vee a \vee \tau,$$

and one of the splitting symbols in τ occurs in one of the σ_i. Call this splitting atom α. Then the clause can be replaced by

$$(\neg\sigma_1 \vee \cdots \vee \neg\sigma_i' \vee \cdots \vee \neg\sigma_p) \vee a \vee \tau.$$

where σ_i' is obtained by deleting α from σ_i. If σ_i' is empty, then the clause can be removed completely.

It is clear that all variants of Definition 2 and Definition 3 can be handled, because the clauses resulting from the split logically imply the original clause.

Definition 9. *A saturated set $M^{\Sigma} \subseteq A^{\Sigma}$ of $\mathcal{C}^{\Sigma}$ is defined as follows:*

- *For each clause c, which is derivable by rule CONTEXT from clauses $c_1, \ldots, c_n \in M^{\Sigma}$, there are clauses $d_1, \ldots, d_m \in M^{\Sigma}$, such that*

$$R^{\Sigma}(d_1, \ldots, d_m, c).$$

- *For each clause c, which is derivable by rule RESTORE from clauses $c_1, \ldots, c_n \in M^{\Sigma}$, there are clauses $d_1, \ldots, d_m \in M^{\Sigma}$ that do not contain negative Σ-literals, and*

$$R^{\Sigma}(d_1, \ldots, d_m, c)$$

A set $M^{\Sigma} \subseteq A^{\Sigma}$ is a saturation of some set of initial clauses $I^{\Sigma} \subseteq A^{\Sigma}$ if it is a saturated set of $\mathcal{C}^{\Sigma}$ and for each $c \in I^{\Sigma}$, there are $d_1, \ldots, d_m \in M^{\Sigma}$, such that

$$R^{\Sigma}(d_1, \ldots, d_m, c).$$

It is necessary to restrict R^{Σ} in the definition of a saturated set, because the full combination of D^{Σ} and R^{Σ} would have been incomplete, even when $\mathcal{C}$ is complete.

Theorem 2. *If $\mathcal{C}^{\Sigma}$ has a saturation that does not contain the empty clause, then $\mathcal{C}$ has a saturation not containing the empty clause.*

Proof. Let M^{Σ} be a saturation of $\mathcal{C}^{\Sigma}$.

We construct an interpretation $M = M_1 \cup M_2$, s.t. M_1 consists of Σ-atoms, M_2 consists of clauses from A, and all clauses of M^{Σ} are true in M. A clause $(\neg\sigma_1 \vee \cdots \vee \neg\sigma_p) \vee a \vee \tau$ is true in M, if one of the following holds:

- For one of the σ_i, none of the literals in σ_i occurs in M_1.
- There are clauses $a_1, \ldots, a_m \in M_2$, that make a redundant in the original calculus.
- One of the symbols of τ occurs in M_1.

First put

$$C_1 = \{c \in M^{\Sigma} \mid c \text{ has form } (\ ) \vee e \vee \tau\}.$$

These are the clauses containing only positive atoms from Σ. Next put

$$C_2 = \{c \in M^{\Sigma} \mid c \text{ has form } (\ ) \vee a \vee \sigma \text{ and } a \neq e\}.$$

We construct the set M_1 from a sequence $\Sigma_0, \Sigma_1, \ldots$ The set of symbols Σ is well-ordered by $\prec$. Let k_1 be the ordinal length of $\prec$ on Σ. Write σ_λ for the λ-th element of Σ, based on $\prec$.

- For a limit ordinal λ, put $\Sigma_\lambda = \bigcup_{\mu<\lambda} \Sigma_\mu$. This implies that $\Sigma_0 = \{\ \}$.
- For a successor ordinal $\lambda + 1$, put $\Sigma_{\lambda+1} = \Sigma_\lambda \cup \{\sigma_\lambda\}$ if there is a clause of the form $(\sigma_\lambda \vee \tau) \in C_1$, in which σ_λ is $\prec$-maximal and $\sigma_\lambda \vee \tau$ is false in Σ_λ. If there is no such clause, then put $\Sigma_{\lambda+1} = \Sigma_\lambda$.
- Finally put $M_1 = \Sigma_{k_1}$.

Each clause C_1 is true in M_1 and for each symbol $\sigma \in M_1$ there is a clause $c \in C_1$, such that σ is the maximal literal in c, and σ is the unique true literal of c.
M_2 is constructed essentially similar to M_1, but there is no need to do an iteration, because every clause contains at most one clause from A. Let

$$M_2 = \{a \mid (\ ) \vee a \vee \tau \in C_2, \text{ and } \tau \text{ is false in } M_1\}.$$

We want to show that all clauses in M^{Σ} are true in M. For the clauses of C_1 and C_2, this is immediate. For the remaining clauses, we use the following argument: Let c be a clause of form

$$(\neg\sigma_1 \vee \cdots \vee \neg\sigma_p) \vee a \vee \tau,$$

with $p > 0$. Suppose that all σ_i are true in M_1. For each σ_i, there is a symbol $\alpha_i \in \Sigma$, such that $\alpha_i \in M_1$. There must be clauses

$$\alpha_1 \vee \tau_1, \ldots, \alpha_p \vee \tau_p,$$

s.t. each α_i is maximal in $\alpha_i \vee \tau_i$, and each τ_i is false in M_1. By rule RESTORE, one can derive

$$d = a \vee (\tau \vee \tau_1 \vee \cdots \vee \tau_p).$$

There are clauses $b_1, \ldots, b_m \in C_1 \cup C_2$, which are true in M, s.t.

$$D^{\Sigma}(b_1, \ldots, b_m, d).$$

Because of this, d must be true in M. Since none of the τ_i is true, $a \vee \tau$ must be true in M. This makes

$$(\neg\sigma_1 \vee \cdots \vee \neg\sigma_p) \vee a \vee \tau$$

true.
It remains to show that M_2 is a saturated set of $\mathcal{C}$ and that M_2 does not contain e. Clearly, by the way M_2 is constructed, $e \notin M_2$.
Suppose that there are clauses $a_1, \ldots, a_n \in M_2$ from which an inference $R(a_1, \ldots, a_n, a)$ is possible. There are clauses $a_1 \vee \tau_1, \ldots, a_n \vee \tau_n \in M^{\Sigma}$ for which the τ_i are false. By rule CONTEXT, it is possible to derive

$$a \vee (\tau_1 \vee \cdots \vee \tau_n).$$

If we can prove that this clause is true in M, then we are ready, because then a must be true. If a is true there must be clauses in M_2, that make it redundant.

Because M^{Σ} is a saturated set of $\mathcal{C}^{\Sigma}$, there are clauses $d_1, \ldots, d_m \in M^{\Sigma}$, which make $a \vee (\tau_1 \vee \cdots \vee \tau_n)$ redundant. The clauses $d_1, \ldots, d_m$ are true in M. By definition of redundancy, there exist $\mathcal{C}$-clauses

$$a_{1,1}, \ldots, a_{1,m_1}, b_1, \ldots, a_{k,1}, \ldots, a_{k,m_k}, b_k,$$

such that

$$R(a_{1,1}, \ldots, a_{1,m_1}, b_1), \ldots, R(a_{k,1}, \ldots, a_{k,m_k}, b_k),$$

and

$$\neg a_{1,1} \vee \cdots \vee \neg a_{1,m_1} \vee b_1, \ldots, \neg a_{k,1} \vee \cdots \vee \neg a_{k,m_k} \vee b_k,$$
$$d_1, \ldots, d_m \models a \vee (\tau_1 \vee \cdots \vee \tau_n)$$

in propositional logic.

We know already that the clauses $d_1, \ldots, d_m$ are true in M. The other clauses $\neg a_{j,1} \vee \cdots \vee \neg a_{j,m_j} \vee b_j$ $(1 \leq j \leq k)$ are true by the fact that R is a transitive relation. From this it follows that $a \vee (\tau_1 \vee \cdots \vee \tau_n)$ is true.

4 Conclusions

We have presented a way for simulating search state splitting by splitting through new symbols. The method preserves redundancy elimination, what is particularly important if one is looking for a saturated set.

In our method negative splitting literals are always selected, so that they block the clause. In [RV01], an interesting alternative was introduced: Simply use an A-order which makes both the positive and the negative splitting atom minimal. In this way, different splitting branches are explored in parallel. Different branches cannot interact, because any inference between clauses of the form $C_1 \vee \alpha$ and $C_2 \vee \neg\alpha$ will result in a tautology. It would be interesting to see if our method of redundancy elimination could be combined with this style of splitting.

5 Acknowledgements

The author became aware of the problem with splitting through proposition symbols after a discussion with Ullrich Hustadt. Harald Ganzinger read a draft version. The present presentation has benefitted from his comments.

References

[BFL94] Baaz, M., Fermüller, C. Leitsch, A.: A non-elementary speed up in proof length by structural clause form transformation, In LICS 94.

[BG01] Bachmair, L., Ganzinger, H.: Resolution Theorem Proving, pp. 19–99, in the Handbook of Automated Reasoning, **2001**, Edited by A. Robinson and A. Voronkov, Elsevier Science, Amsterdam, the Netherlands.

[B71] Boyer, R.S.: Locking: A Restriction of Resolution (Ph. D. Thesis). University of Texas at Austin, (1971).

[FLTZ93] Fermüller, C., Leitsch, A., Tammet, T., Zamov, N.: Resolution Methods for the Decision Problem. LNCS 679, Springer Verlag Berlin Heidelberg New York, (1993).

[GNN98] Ganzinger, H., Nieuwenhuis, R., Nivela, P., The Saturate System, `www.mpi-sb.mpg.de/SATURATE/Saturate.html`, **1998**.

[GdN99] Ganzinger, H., de Nivelle, H.: A superposition procedure for the guarded fragment with equality. LICS **14**, IEEE Computer Society Press, (1999), 295–303.

[HdNS00] Hustadt, U., de Nivelle, H., Schmidt, R.: Resolution-Based Methods for Modal Logics, Journal of the IGPL **8-3**, (2000), 265-292.

[dN94] de Nivelle, H.: Resolution Games and Non-Liftable Resolution Orderings, in Computer Science Logic **1994**, Selected Papers, (1995) pp. 279–293, LNCS 933, Springer Verlag.

[dN00] de Nivelle, H.: Deciding the E^+-Class by an A Posteriori, Liftable Order. Annals of Pure and Applied Logic **104-(1-3)**, (2000), pp. 219–232, Elsevier Science, Amsterdam.

[dNPH01] de Nivelle, H., Pratt-Hartmann, I.: A Resolution-Based Decision Procedure for the Two-Variable Fragment with Equality, Proceedings IJCAR **2001**, Springer Verlag (2001), 211-225.

[NR01?] de Nivelle, H., de Rijke, M.: Deciding the Guarded Fragments by Resolution, to appear in the Journal of Symbolic Computation.

[dN01] de Nivelle, H.: Translation of Resolution Proofs into Higher Order Natural Deduction, **2001**, unpublished, to appear.

[NW01] Nonnengart, A., Weidenbach, C.: Computing Small Clause Normal Forms, pp. 335–370, in the Handbook of Automated Reasoning, **2001**, Edited by A. Robinson and A. Voronkov, Elsevier Science, Amsterdam, the Netherlands.

[RV01] Riazanov, A., Voronkov, A.: Splitting Without Backtracking, Preprint of the Department of Computer Science, University of Manchester, CSPP-10.

[Wb01] Weidenbach, C.: SPASS: Combining Superposition, Sorts and Splitting, pp. 1967–2012, in the Handbook of Automated Reasoning, **2001**, Edited by A. Robinson and A. Voronkov, Elsevier Science, Amsterdam, the Netherlands.