Ann Math Artif Intell (2009) 55:3–34
DOI 10.1007/s10472-009-9150-9

Labelled splitting

Arnaud Fietzke · Christoph Weidenbach

Published online: 23 July 2009

Abstract We define a superposition calculus with explicit splitting on the basis of labelled clauses. For the first time we show a superposition calculus with an explicit non-chronological backtracking rule sound and complete. The new backtracking rule advances backtracking with branch condensing known from Spass. An experimental evaluation of an implementation of the new rule shows that it improves considerably on the previous Spass splitting implementation. Finally, we discuss the relationship between labelled first-order splitting and DPLL style splitting with intelligent backtracking and clause learning.

Keywords Superposition · Splitting · Labels

Mathematics Subject Classification (2000) 68T15

1 Introduction

Splitting is an inference rule for case analysis. It is well-known from the Davis-Putnam-Logemann-Loveland (DPLL) [6] decision procedure for propositional logic, where a propositional clause set N is split into the clause sets $N \cup \{A\}$ and $N \cup \{\neg A\}$ for some propositional variable A occurring positively and negatively in N. Obviously, N is satisfiable iff one of the two split clause sets $N \cup \{A\}$ or $N \cup \{\neg A\}$ is satisfiable. Furthermore, both split clause sets are strict subsets of N[1] after unit propagation and subsumption with A and $\neg A$, respectively. Both clause sets are also smaller in the sense of the standard superposition ordering for clauses and

[1] We compare clauses by the subset relation as well.

A. Fietzke (✉) · C. Weidenbach
Max-Planck-Institut für Informatik, Campus E1 4, 66123 Saarbrücken, Germany
e-mail: fietzke@mpi-inf.mpg.de

C. Weidenbach
e-mail: weidenbach@mpi-inf.mpg.de

clause sets [2, 15]. Therefore, from a superposition perspective splitting can be considered as a simplification rule as well and has in particular shown to be useful in turning the superposition calculus into a decision procedure for certain first-order fragments [3, 9, 11]. This is our main motivation to have explicit splitting in Spass [20–22].

The DPLL decision procedure does not consider the two split clause sets in parallel by duplicating N, but traverses the tree generated by splitting and backtracking in a depth-first way. By appropriate implementation mechanisms, backtracking then becomes quite cheap. As any clause set resulting from splitting is a subset of subclauses after reduction with the split variable, updates can be made by marking techniques and there is no need to generate new clauses.[2] Nieuwenhuis et al. [14] presented the DPLL procedure performing depth-first search by an abstract calculus. The main contribution of this paper is a calculus for first-order superposition with explicit splitting, generalizing the DPLL procedure to the full first-order case with equality.

In first-order logic the DPLL splitting style does typically not make sense, because for a given first-order atom A there exist infinitely many ground instances $A\sigma$ of A and it is not a priori known which instances eventually contribute to a proof or a model. In case of models having infinite domains, such a style of splitting won't terminate for satisfiable clause sets. Furthermore, the split clause sets $N \cup \{A\sigma\}$ and $N \cup \{\neg A\sigma\}$ are only strict subsets of N after unit propagation and subsumption if $A\sigma$ already occurs in N. Therefore, in the context of clauses with variables a different style of splitting is used: given a clause $C \in N$ that can be decomposed into two non-trivial variable disjoint subclauses C_1, C_2, we split into the clause sets $N \cup \{C_1\}$ and $N \cup \{C_2\}$. As C_1 and C_2 both subsume C, the split clause sets are again strict subsets of N and smaller with respect to the standard superposition ordering for clause sets. Furthermore, both clauses C_1 and C_2 contain strictly less different variables than C. This property is quite useful in making superposition a decision procedure for certain first-order clause fragments. The superposition calculus is a decision procedure for a particular clause fragment if the maximal term depth as well as the number of different variables in clauses is finitely bounded in the course of a superposition saturation. If this is the case for a fixed finite signature of a given clause set, then superposition only generates finitely many clauses modulo subsumption, i.e., saturation terminates and superposition becomes a decision procedure. As the above style of splitting reduces the number of different variables in a clause, it is often an indispensable ingredient of superposition based decision procedures [3, 9, 11].

In the current Spass implementation the rule is further restricted to require that both C_1 and C_2 contain at least one positive literal, i.e., we split into clause sets that are closer to Horn. The first rationale behind this restriction is that for Horn clause sets decidability results are typically "easier" to establish and more efficient algorithms exist. For example, satisfiability of propositional Horn clause sets can be decided in linear time, whereas satisfiability of arbitrary propositional clause sets is an NP-complete problem. Secondly, as argued above, we always like to limit the number of different variables in clauses. The superposition inference rules that potentially generate clauses with a growing number of different variables

[2] Learning clauses is a separate issue.

rely on inference steps with positive literals. Therefore, the restriction to non-trivial of positive literals typically suffices to prevent the generation of clauses with more different variables.

A further major difference between first-order splitting and propositional splitting is that in the first-order case effective theorem proving typically relies on the generation of new clauses, either via inferences or simplifications. Therefore, the bookkeeping for backtracking of a depth-first approach to splitting gets more involved, because marking algorithms on existing clauses are no longer sufficient to track changes. We need to extend the labels used in the abstract DPLL calculus that are sequences of decision and propagation literals, to a sequence of split elements, called the split stack, where any split element stores in particular the potential second part of the split clause and all clauses that became redundant in the course of splitting and may have to be reactivated. A further consequence of the generation of new clauses is that the number of possible splits cannot be finitely bounded for the first order case. Therefore, any calculus preferring splitting rules over other inference rules has to take into account a new dimension of fairness, going beyond the fairness notions developed for superposition without splitting. Our second contribution is an extended notion of fairness enabling the soundness and completeness results for labelled splitting to be proven. In particular, to the best of our knowledge, this represents the first soundness and completeness result for a proof procedure using a form of non-chronological backtracking outside the areas of constraint solving and propositional logic.

There are two previously presented "splitting" calculi for first-order logic [7, 17]. They suggest to replace a clause C in some clause set N that can be decomposed into two variable disjoint components C_1, C_2 by two clauses $q \vee C_1$ and $\neg q \vee C_2$ where q is a new propositional symbol. If then $\neg q$ is selected in $\neg q \vee C_2$ and q is made the smallest atom in the ordering no inference step between q and $\neg q$ is done as long as C_1 or C_2 have not been "resolved away", giving a flavor of explicit splitting. Obviously, the transformation preserves satisfiability, but does not constitute an explicit case analysis, nor is the new clause set $(N \setminus \{C\}) \cup \{q \vee C_1, \neg q \vee C_2\}$ a strict subset of N. Therefore, from the superposition perspective this rule is not a simplification but an inference that introduces a new propositional variable. Actually, this technique is much closer to definitional extensions [16, 19] than to splitting. This was already pointed out by one of the authors [7]. As a consequence this technique cannot replace the explicit splitting needed for decidability results, because the introduction of new propositional symbols violates the termination argument. Related to the introduction of new propositional symbols there are also fairness and completeness issues that have not been considered by the authors. First, the introduction of up to infinitely many new propositional symbols potentially violates the well-foundedness requirement for orderings in the superposition context, hence leading to completeness issues. This can be resolved, but the price to pay is that the propositional symbols must not be totally ordered, leading to a prolific search space or a loss of completeness, if one doesn't a priori finitely limit the number of definitional extensions. If the number of definitional extensions is not limited, then also fairness becomes an issue similar to the way it shows up in our calculus. This is not discussed in the above mentioned papers but an original contribution of ours. Definitional extensions are a very useful technique for finding proofs but not for obtaining decision results or finding models.

The model evolution calculus [5] also includes an explicit splitting rule. As a different generalization of DPLL to what is suggested in our contribution, the calculus splits a clause set N into the clause sets $N \cup \{A\}$ and $N \cup \{skolem(\neg A)\}$ where A is a first-order atom potentially including variables that are replaced by fresh Skolem constants in $skolem(\neg A)$. The new Skolem constants are not treated naively but special refinements of the calculus ensure that they can be reused to a certain extend. The clause set $N \cup \{skolem(\neg A)\}$ is not a strict subset of N,[3] so this rule is also not a simplification but an inference rule, and it is therefore not obvious whether the model evolution calculus can be turned into a decision procedure for all the fragments for which we know superposition with explicit splitting to be a decision procedure. Eventually, the paper does not take a deep investigation into fairness issues, but rather suggests an iterative deepening approach, where the number of splits for any round is finitely bounded.

For the general methodology of labelled clauses there is a huge literature, see [4] for an overview. In particular, the use of clause labels to model explicit case analysis was first suggested in [12], which provided the basis for the work presented here. We refined the abstract labelled splitting rule presented there with explicit backtracking and redundancy handling.

Our starting point is the splitting approach as it was implemented in Spass [20, 21]. On the basis of this calculus we develop the labelled splitting calculus that in particular refines the previously implemented one with an improved backtracking rule, Right-collapse (Section 2), which allows larger parts of the split tree to be pruned (see Fig. 2). We show the labelled splitting calculus to be sound and complete where we introduce a new notion of fairness, taking into account an infinite path in the split tree. Labelled splitting is implemented in Spass (http://spass-prover.org/) and improves significantly on the previous implementation (Section 3). In addition to the above discussion on related work, we compare the calculus in detail to the DPLL approach with intelligent backtracking and clause learning (Section 3). The paper ends with a summary of the achieved results and directions for future work (Section 4). This article is an extended version of [10].

2 Labelled splitting

2.1 Preliminaries

We employ the usual notions and notations of first-order logic and superposition in a way consistent with [20]. When traversing the split tree, the conclusions of splits that were used in deriving a clause determine the clause's scope in the tree, i.e., those parts of the tree in which the clause may participate in proof search. In order to capture this information, we identify each case of a splitting application on a given path through the tree with a unique integer, its *split level*, and label each clause with a set of integers, representing all splits that contributed to the derivation of the clause.

Formally, a labelled clause $L : C$ consists of a finite set $L \subseteq \mathbb{N}$ and a clause $C = \Gamma \rightarrow \Delta$ where Γ and Δ contain the negatively and positively occurring atoms,

[3] In our usual sense, i.e., after redundancy elimination.

respectively. The empty clause with label L is denoted by $L\colon \square$. We call the greatest element in L the *split level* of the clause. We say that $L\colon C$ *depends* on l if $l \in L$. We extend the usual notions for clause sets to sets of labelled clauses by ignoring the labels: for example, we will say that a set N of labelled clauses entails some formula ϕ (written $N \models \phi$) if and only if the corresponding set of unlabelled clauses entails ϕ. In the same way, we also extend clause orderings [20] to labelled clauses.

A *labelled clause set* is of the form $\Psi : N$ where N is a set of labelled clauses and Ψ is the *split stack*. Split stacks are sequences $\Psi = \langle \psi_n, \ldots, \psi_1 \rangle$ of length $n \geq 0$ and correspond to paths in the split tree. The ψ_i are tuples of the form $(l_i, B_i, D_i, \varphi_i)$ called *splits*, where $l_i \in \mathbb{N}$ is the split level, B_i is the set of *blocked* clauses, D_i is the set of *deleted* clauses and $\varphi_i \in \{\emptyset, \{L\}\}$ with $L \subseteq \mathbb{N}$, is the *leaf marker*. Splitting a clause results in a new split being put onto the stack, which can be thought of as entering the corresponding left branch in the split tree. Once the branch has been refuted, the split is removed and possibly replaced by a split representing the right branch of the splitting step, as shown in Fig. 1. We call i the *index* of split ψ_i in Ψ. We define the function level_Ψ that for any valid index of Ψ returns the split level $\mathrm{level}_\Psi(i) = l_i$ of the corresponding split ψ_i. We refer to the inverse of level_Ψ as index_Ψ, i.e., $\mathrm{index}_\Psi(l_i) = i$. Label-validity (Proposition 1) ensures that index_Ψ is well-defined. The reason we distinguish split levels from indices is that the levels of Ψ in general will not form a contiguous sequence (contrary to the indices). This is because splits are removed from the stack during backtracking, leaving "holes" in the level sequence. Splits corresponding to left branches will be assigned odd levels, while splits corresponding to right branches will have even levels. This is captured by two predicates, $\mathrm{left}(l) = \mathrm{true}$ iff $l \bmod 2 = 1$ and $\mathrm{right}(l) = \mathrm{true}$ iff $l \bmod 2 = 0$. We call $\psi = (l, B, D, \varphi)$ a *left split* if $\mathrm{left}(l)$, and a *right split* otherwise. Furthermore we define the set $\mathrm{levels}(\Psi)$ to be the set of all split levels of splits in Ψ, i.e., $\mathrm{levels}(\Psi) := \{\mathrm{level}_\Psi(1), \ldots, \mathrm{level}_\Psi(n)\}$. For any set N of labelled clauses and set of split levels $K \subseteq \mathbb{N}$ we define $N\big|_K := \{L\colon \Gamma \to \Delta \in N \mid L \subseteq K\}$ and $N\big|_{\overline{K}} := \{L\colon \Gamma \to \Delta \in N \mid L \cap K = \emptyset\}$.

The leaf marker φ records which splits were involved in refuting a branch and is used during backtracking to get rid of unnecessary splits. In a left split, the leaf marker will be empty, and the set B will contain clauses to be reinserted when entering the right branch. In the present framework, B will always consist of just the second split clause. However using a set allows for additional clauses to be added to the right branch, for example the negation of the first split clause in case it is ground. In a right split, φ will contain the label of the empty clause with which the corresponding left branch was refuted. We distinguish $\varphi = \{\emptyset\}$ from $\varphi = \emptyset$ because the former implies the existence of an empty clause that does not depend on any splits (the *toplevel* empty clause), while the latter means no empty clause was derived yet.

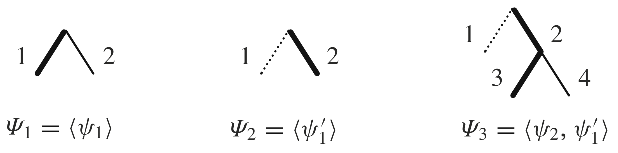
1
2
$\Psi_1 = \langle \psi_1 \rangle$
1
2
$\Psi_2 = \langle \psi_1' \rangle$
1
2
3
4
$\Psi_3 = \langle \psi_2, \psi_1' \rangle$

Fig. 1 Split trees and corresponding stacks. ψ_1 and ψ_2 are left splits, ψ_1' is a right split, and $\mathrm{level}_{\Psi_1}(1) = 1$, $\mathrm{level}_{\Psi_2}(1) = \mathrm{level}_{\Psi_3}(1) = 2$ and $\mathrm{level}_{\Psi_3}(2) = 3$

The set D is used to record reduced clauses that may have to be reinserted when the split is removed. For better readability, we use the notation $\psi[x := v]$ where x is one of l, B, D, φ to denote a split identical to ψ up to component x, which has value v. We write $\psi[x_1 := v_1, x_2 := v_2]$ instead of $\psi[x_1 := v_1][x_2 := v_2]$.

For the definition of the labelled calculus we distinguish inference rules

$$\mathcal{I}\,\frac{\Psi : N \qquad L_1\colon \Gamma_1 \to \Delta_1 \quad \ldots \quad L_n\colon \Gamma_n \to \Delta_n}{\Psi' : N' \qquad\qquad K\colon \Pi \to \Lambda}$$

which produce new clauses, and reduction rules

$$\mathcal{R}\,\frac{\Psi : N \qquad L_1\colon \Gamma_1 \to \Delta_1 \quad \ldots \quad L_n\colon \Gamma_n \to \Delta_n}{\begin{array}{ll}\Psi' : N' \qquad\qquad & K_1\colon \Pi_1 \to \Lambda_1 \\ & \vdots \\ & K_k\colon \Pi_k \to \Lambda_k\end{array}}$$

which replace or remove clauses. The clauses $L_i\colon \Gamma_i \to \Delta_i$ are called the *premises* and the clauses $K_{(i)}\colon \Pi_{(i)} \to \Lambda_{(i)}$ the *conclusions* of the respective rule. A rule is applicable to a labelled clause set $\Psi : N$ if the premises of the rule are contained in N. In the case of an inference, the resulting labelled clause set is $\Psi' : (N' \cup \{K\colon \Pi \to \Lambda\})$. In the case of a reduction, the resulting labelled clause set is $\Psi' : (N' \setminus \{L_i\colon \Gamma_i \to \Delta_i \mid 1 \leq i \leq n\} \cup \{K_j\colon \Pi_j \to \Lambda_j \mid 1 \leq j \leq k\})$. Reduction rules which replace clauses, i.e., where $k > 0$, are often called *simplifications*. Because simplifications can be viewed as combinations of inferences and reductions (removing some of the premises, while adding some new clauses), we will not distinguish simplifications from reductions and instead refer to all rules of the second form as reductions. Furthermore, we say that a clause C is *redundant* in N if it follows from smaller clauses in N, and an inference is redundant in N, if its conclusion follows logically from clauses in N that are smaller than its maximal premise.

2.2 Labelled calculus

We present a basic set of inference and reduction rules which together yield a sound and refutationally complete calculus for first-order logic without equality. All other rules, i.e., equality inference rules or more advanced reduction rules [20] can be easily defined accordingly and are contained in our implementation. Since our emphasis lies on the modelling of the splitting process, we omit these rules here. In fact, any inference or reduction can be integrated into our framework analogously to the rules presented here, as long as that rule satisfies the following basic assumptions: an inference or reduction should only produce logical consequences of its premises, while a reduction may only remove clauses that are redundant with respect to its premises. The two exceptions to this requirement are the splitting rule (Definition 1) and the backtracking rule (Definition 12), which have no unlabelled equivalent. Since these rules not only operate on clauses but also manipulate the stack, their soundness follows from the fact that they preserve labelled clause set satisfiability (Definition 13). This is a notion of satisfiability lifted to labelled clause sets to take the structure of the split stack into account.

Definition 1 (Splitting) The inference

$$\mathcal{I}\frac{\Psi : N \quad L\colon \Gamma_1, \Gamma_2 \to \Delta_1, \Delta_2}{\Psi' : N \qquad L'\colon \Gamma_1 \to \Delta_1}$$

where $\Psi = \langle \psi_n, \ldots, \psi_1 \rangle$ ($l_n = 0$ if $n = 0$), $l_{n+1} = 2\left\lceil \frac{l_n}{2} \right\rceil + 1$, $L' = L \cup \{l_{n+1}\}$, $L'' = L \cup \{l_{n+1} + 1\}$, $\Psi' = \langle \psi_{n+1}, \psi_n, \ldots, \psi_1 \rangle$ with $\psi_{n+1} = (l_{n+1}, \{L''\colon \Gamma_2 \to \Delta_2\}, \emptyset, \emptyset)$, $vars(\Gamma_1 \to \Delta_1) \cap vars(\Gamma_2 \to \Delta_2) = \emptyset$, and $\Delta_1 \neq \emptyset$ and $\Delta_2 \neq \emptyset$ is called *splitting*.

Splitting creates a new split representing the left branch $\Gamma_1 \to \Delta_1$ on the stack. The remainder is kept in the new split's blocked clause set to be restored upon backtracking. The new split level l_{n+1} is the smallest odd number larger than l_n, so that left(l_{n+1}) and right($l_{n+1} + 1$) hold. Furthermore, note that splitting is an inference and the parent clause $\Gamma_1, \Delta_1 \to \Gamma_2, \Delta_2$ is not removed from the clause set. A concrete proof strategy may require to apply subsumption deletion to the parent clause immediately after each splitting step (and after each backtracking step, when the corresponding right branch is entered), thus turning splitting into a reduction. In the current SPASS implementation, splitting is a reduction in this sense.

In case the first split part $L'\colon \Gamma_1 \to \Delta_1$ is ground, the clauses resulting from its negation can be added to the set of blocked clauses, which will be reinserted when the second split part is considered. With this modification, the above splitting rule is as powerful as the DPLL splitting rule concerning proof complexity.

If we generalize the splitting rule such that the clause $L\colon \Gamma_1, \Gamma_2 \to \Delta_1, \Delta_2$ is a a consequence of N ($N \models L\colon \Gamma_1, \Gamma_2 \to \Delta_1, \Delta_2$) instead of being required to be contained in N, then DPLL-style splitting is an instance of this generalized rule. As this condition applies to any tautology, we may split $\emptyset\colon A \to A$ in the standard DPLL-like fashion for any ground atom A.

Definition 2 (Resolution) The inference

$$\mathcal{I}\frac{\Psi : N \quad L_1\colon \Gamma_1 \to \Delta_1, A \quad L_2\colon \Gamma_2, B \to \Delta_2}{\Psi : N \qquad L_1 \cup L_2\colon (\Gamma_1, \Gamma_2 \to \Delta_1, \Delta_2)\sigma}$$

where (i) σ is the most general unifier of A and B, (ii) no literal in Γ_1 is selected, (iii) $A\sigma$ is strictly maximal in $(\Gamma_1 \to \Delta_1, A)\sigma$, (iv) $B\sigma$ is selected or maximal in $(\Gamma_2, B \to \Delta_2)\sigma$ and no literal is selected in Γ_2 is called *resolution*.

Definition 3 (Factoring) The inference

$$\mathcal{I}\frac{\Psi : N \quad L\colon \Gamma \to \Delta, A, B}{\Psi : N \quad L\colon (\Gamma \to \Delta, A)\sigma}$$

where (i) σ is the most general unifier of A and B, (ii) A and B occur positively, (iii) A is maximal and no literal is selected in Γ is called *factoring*.

Reduction rules remove clauses that are redundant with respect to their premises. In the presence of splitting, these reducing clauses may themselves be removed during a subsequent backtracking step, in which case the reduced clauses may cease to be redundant and have to be reactivated. Therefore, reduced clauses can not be thrown away but must be kept on the split stack as long as there is a chance of them becoming non-redundant again. The role of the deleted sets D in splits is to store reduced clauses for later reinsertion. When a clause with label L_1 is used to reduce a clause $L_2 : C$, the reduced clause is stored in the deleted set at the split level of the reducing clause, that is, in the deleted set of the split at $\mathrm{index}_\Psi(\max(L_1))$. There are two special cases in which we are allowed to remove $L_2 : C$ for good: if $L_1 = \emptyset$, then the reducing clause does not depend on any split and thus will never be removed by backtracking; similarly, if both reducing and reduced clause have the same split level, i.e., $\max(L_1) = \max(L_2)$, then backtracking removes either both clauses, or none of them. We define a function $\mathrm{keep}(\Psi, L_1, L_2 : C)$ which backs up the reduced clause $L_2 : C$ in Ψ:

Definition 4 Let $\Psi : N$ be a labelled clause set with $\Psi = \langle \psi_n, \ldots, \psi_1 \rangle$, $L_1 \subseteq \mathrm{levels}(\Psi)$, and $L_2 : C$ a clause in N. Then we define

$$\mathrm{keep}(\Psi, L_1, L_2 : C) := \begin{cases} \Psi & \text{if } L_1 = \emptyset \text{ or } \max(L_1) = \max(L_2) \\ \langle \psi_n, \ldots, \psi'_{i_1}, \ldots, \psi_1 \rangle & \text{otherwise,} \end{cases}$$

where $\psi'_{i_1} = \psi_{i_1}[D := D_{i_1} \cup \{L_2 : C\}]$ with $i_1 = \mathrm{index}_\Psi(\max(L_1))$.

Definition 5 (Subsumption deletion) The reduction

$$\mathcal{R}\frac{\Psi : N \quad L_1 : \Gamma_1 \to \Delta_1 \quad L_2 : \Gamma_2 \to \Delta_2}{\Psi' : N \qquad\qquad L_1 : \Gamma_1 \to \Delta_1}$$

where $\Gamma_2 \to \Delta_2$ is subsumed by $\Gamma_1 \to \Delta_1$, and $\Psi' = \mathrm{keep}(\Psi, L_1, L_2 : \Gamma_2 \to \Delta_2)$ is called *subsumption deletion.*

The following rules for matching replacement resolution illustrate how simplifications can be viewed as combinations of inferences and reductions.

Definition 6 (Matching replacement resolution) The reductions

$$\mathcal{R}\frac{\Psi : N \quad L_1 : \Gamma_1 \to \Delta_1, A_1 \quad L_2 : \Gamma_2, A_2 \to \Delta_2}{\begin{array}{ll}\Psi'_L : N & L_1 : \Gamma_1 \to \Delta_1, A_1 \\ & L_1 \cup L_2 : \Gamma_2 \to \Delta_2\end{array}}$$

and

$$\mathcal{R}\frac{\Psi : N \quad L_1 : \Gamma_1, A_1 \to \Delta_1 \quad L_2 : \Gamma_2 \to \Delta_2, A_2}{\begin{array}{ll}\Psi'_R : N & L_1 : \Gamma_1, A_1 \to \Delta_1 \\ & L_1 \cup L_2 : \Gamma_2 \to \Delta_2\end{array}}$$

where $A_1\sigma = A_2$, $\Gamma_1\sigma \subseteq \Gamma_2$ and $\Delta_1\sigma \subseteq \Delta_2$ for some matcher σ, $\Psi'_L = \mathrm{keep}(\Psi, L_1, L_2 : \Gamma_2, A_2 \to \Delta_2)$ and $\Psi'_R = \mathrm{keep}(\Psi, L_1, L_2 : \Gamma_2 \to \Delta_2, A_2)$ are called *matching replacement resolution*.

Matching replacement resolution is a restricted variant of replacement resolution, itself a restricted form of resolution where the conclusion must subsume one of its parent clauses.

In general, any rule based on the standard notion of superposition redundancy can be lifted in this way. For the binary case this amounts to the following situation: if $L_1 \colon \Gamma_1 \to \Delta_1$ together with $L_2 \colon \Gamma_2 \to \Delta_2$ logically imply $\Gamma_2' \to \Delta_2'$ and $\Gamma_2' \to \Delta_2'$ is smaller than $\Gamma_2 \to \Delta_2$, then we replace $L_2 \colon \Gamma_2 \to \Delta_2$ by $L_1 \cup L_2 \colon \Gamma_2' \to \Delta_2'$ but take care of $L_2 \colon \Gamma_2 \to \Delta_2$ according to the keep function for the split stack.

2.2.1 Backtracking

We now formalize backtracking. In particular, we focus our attention on the deletion of splits from the split stack to ensure that all the bookkeeping is done correctly. When removing a split ψ_k from the stack, we have to take care to undo all reductions that involved a clause depending on (the level of) ψ_k. In particular, if a clause C depending on ψ_k was used to reduce some other clause D, then D must be reinserted, from the deleted set at C's level, back into the current clause set. The reason is that C will be removed, and D may then no longer be redundant. Although C depends on ψ_k, C may also depend on other splits and thus have a split level greater than l_k. Furthermore, C itself, after having reduced D, may have been reduced by some clause C', hence C will not necessarily be in the current clause set, but in the deleted set at the level of C'. Therefore, we have to look both into the current clause set as well as into all deleted sets on the stack, gather all clauses depending on ψ_k, determine their split levels, and reinsert the deleted clauses from those levels.

Let $\Psi : N$ be an arbitrary labelled clause set with Ψ of length n. For $1 \leq k \leq n$ let $R(\Psi : N,\ k)$ be the set

$$\left\{ \operatorname{index}_\Psi(\max(L)) \;\middle|\; L \colon C \in N \cup \bigcup_{\substack{i=1 \\ i \neq k}}^{n} D_i \ \wedge\ \operatorname{level}_\Psi(k) \in L \right\}$$

For every $j \in R(\Psi : N,\ k)$, there is a clause with level $\operatorname{level}_\Psi(j)$, either in N or some deleted set of Ψ, that also depends on ψ_k.

Definition 7 (Split deletion) For any labelled clause set $\Psi : N$ with $\Psi = \langle \psi_n, \ldots, \psi_1 \rangle$, and for $1 \leq k \leq n$, we define $\operatorname{delete}(\Psi : N, k) := \Psi' : N'$ with

$$\Psi' = \langle \psi_n', \ldots, \psi_{k+1}', \psi_{k-1}', \ldots, \psi_1' \rangle$$

and

$$N' = \left(N \cup D_k \cup \bigcup_{j \in R(\Psi : N,\ k)} D_j \right) \Bigg|_{\overline{\{\operatorname{level}_\Psi(k)\}}}$$

where

$$\psi_j' = \begin{cases} \psi_j[D := \emptyset] & \text{if } j \in R(\Psi : N,\ k) \\ \psi_j\big[D := \{L \colon \Gamma \to \Delta \in D_j \mid \operatorname{level}_\Psi(k) \notin L\}\big] & \text{otherwise,} \end{cases}$$

which removes split ψ_k, all clauses depending on ψ_k, and reinserts all clauses reduced by a clause depending on ψ_k.

The set $\bigcup_{j \in R(\Psi : N,\ k)} D_j$ contains all clauses that may have been reduced by a clause depending on ψ_k (except those in D_k, which are reinserted anyway). Reinserting all those clauses is an over-approximation, since not every clause in every D_j was necessarily reduced by a clause depending on ψ_k. In fact, it may well be that in some D_j, no clause was reduced by a clause depending on ψ_k. If we wanted to reinsert only clauses reduced by clauses depending on ψ_k, we would have to record which clause was used in each reduction step, not only the reducing clause's split level. It is not clear whether that additional effort would pay off in practice.

We now define a reduction[4] relation $\rightarrow$ on labelled clause sets to capture the structural transformations of the stack that take place during backtracking. Let $\Psi : N$ be a labelled clause set with Ψ is of length n. We denote by $\mathrm{maxr}(\Psi) := max(\{1 \leq i \leq n \mid \mathrm{right}(\mathrm{level}_\Psi(i))\} \cup \{0\})$ the last right split in Ψ. The reduction relation is defined by the following four rules, where write l_k for $\mathrm{level}_\Psi(k)$.

Definition 8 (Backjump) If $n > 0$, $L{:}\,\Box \in N$ and $\max(L) < l_n$, then

$$\Psi : N \rightarrow \mathrm{delete}(\Psi : N, n).$$

The Backjump rule removes the toplevel split if it did not contribute to the empty clause $L{:}\,\Box$. Applying Backjump exhaustively yields a split stack that is either empty (if $L = \emptyset$) or has a toplevel split with a split level $l \in L$.

Definition 9 (Branch-condense) If $n > 0$, $L{:}\,\Box \in N$, $\max(L) = l_n$, $\mathrm{left}(l_n)$ and $k_{max} := \max\ \{k \mid \mathrm{maxr}(\Psi) < k \leq n \text{ and } l_k \notin L\}$ exists, then

$$\Psi : N \rightarrow \mathrm{delete}(\Psi : N, k_{max}).$$

The rule Branch-condense removes an inner (i.e., non-toplevel) split if it did not contribute to the empty clause. However, only splits up to the last right split are considered. Dropping this restriction results in an unsound procedure, since the splits used to close a left branch (represented by the leaf marker) must be taken into consideration when analyzing the dependencies of the corresponding right branch. This analysis is performed by the following rule.

Definition 10 (Right-collapse) If $n > 0$, $L_2{:}\,\Box \in N$, $\max(L_2) = l_n$ and $\mathrm{right}(l_n)$, and $\varphi_n = \{L_1\}$, then

$$\Psi : N \rightarrow \Psi' : (N' \cup \{L{:}\,\Box\}),$$

where $\Psi' : N' = \mathrm{delete}(\Psi : N, n)$ and $L = L_1 \cup L_2 \setminus \{l_n - 1, l_n\}$.

The Right-collapse rule analyzes the dependencies involved in refuting left and right branches and computes a newly labelled empty clause by removing complementary split levels. The rule improves upon previous backtracking mechanisms by allowing consecutive sequences of Backjump steps (interleaved by applications of Right-collapse) to take place within a single Backtracking step, thus possibly pruning larger parts of the split tree.

[4]Not to be confused with reduction rules for clause sets. See [1] for a discussion of abstract reduction systems.

Definition 11 (Enter-right) If $n > 0$, $L\colon \Box \in N$, $\max(L) = l_n$, left(l_n) and $l_k \in L$ for all k with maxr$(\Psi) < k \leq n$, then $\Psi'' : N'' :=$ delete$(\Psi : N, n)$ and

$$\Psi : N \to \Psi' : N',$$

where $\Psi' = \langle (l_n + 1, \emptyset, \emptyset, \{L\}), \psi''_{n-1}, \ldots, \psi''_1 \rangle$ and $N' = N'' \cup B_n$.

Finally, Enter-right replaces a toplevel left split by a right split representing the second case of the corresponding splitting step.

At most one rule is applicable to any given labelled clause set containing one empty clause, since the preconditions are mutually exclusive. Furthermore, the length of the split stack together with the number of empty clauses in a labelled clause set induce a well-founded partial ordering on labelled clause sets, with respect to which each rule is decreasing. Hence any sequence $\Psi : N \to \Psi' : N' \to \ldots$ terminates and each labelled clause set $\Psi : N$ has a unique normal form with respect to $\to$, which we write $\Psi : N{\downarrow}$. We are now ready to give the definition of the backtracking rule:

Definition 12 (Backtracking) The reduction

$$\mathcal{R}\frac{\Psi : N \qquad L\colon \Box}{(\Psi : N')\!\downarrow}$$

where $N' = \{L'\colon C \in N \mid C \neq \Box\} \cup \{L\colon \Box\}$ is called *backtracking*.

Since we have not placed any restrictions on when to apply backtracking, there may be more than one empty clause in $\Psi : N$. Choosing one and removing all others before applying stack reductions ensures that the result of backtracking is uniquely defined. In a practical system, one would typically choose the most general empty clause for backtracking, i.e., the one whose label represents a minimal scope in the split tree. The following example shows the stack reduction rules in action.

Example 1 Consider the clause set

$$\begin{array}{lll}\{\to P(a), Q(b); & P(x) \to P(f(x)), R(c); & P(f(f(a))) \to; \\ \ \to Q(x), S(y); & Q(x), P(x) \to; & \\ \ \to R(a), R(b); & S(x), P(x) \to \} & \end{array}$$

where we omit empty labels. We perform the following operations:

- clause $\to P(a), Q(b)$ is split,
- resolution is applied to $P(a)$ and $P(x) \to P(f(x)), R(c)$,
- the resulting clause $\{1\}\colon\!\to P(f(a)), R(c)$ is split,
- clause $\to R(a), R(b)$ is split,
- resolution is applied to $\{1, 3\}\colon P(f(a))$ and $P(x) \to P(f(x)), R(c)$, resulting in $\{1, 3\}\colon\!\to P(f(f(a))), R(c)$,
- which is again split,
- finally the empty clause $\{1, 3, 7\}\colon \Box$ is derived by resolving $\{1, 3, 7\}\colon P(f(f(a)))$ and $P(f(f(a))) \to$.

The resulting split tree is shown in Fig. 2 (first tree). The corresponding split stack is

$$\langle\ (7, \{\{8\}\colon R(c)\}, \emptyset, \emptyset),\ (5, \{\{6\}\colon R(b)\}, \emptyset, \emptyset),\\ (3, \{\{4\}\colon R(c)\}, \emptyset, \emptyset),\ (1, \{\{2\}\colon Q(b)\}, \emptyset, \emptyset)\ \rangle.$$

We now apply backtracking.

- The third split did not contribute to the contradiction, so it is removed by Branch-condense in step 1,
- followed by Enter-right, which produces $(8, \emptyset, \emptyset, \{\{1, 3, 7\}\})$ as toplevel split.
- The clause $\to Q(x), S(y)$ is then split, resolution is applied to $\{1\}\colon P(a)$ and $Q(x), P(x) \to$ to derive $\{1\}\colon Q(x) \to$, which is resolved with $\{9\}\colon Q(x)$ to yield $\{1, 9\}\colon \square$. Enter-right is applied (step 3), producing toplevel split $(10, \emptyset, \emptyset, \{\{1, 9\}\})$, and the empty clause $\{1, 10\}\colon \square$ is derived using clauses $\{1, 10\}\colon S(y)$ and $S(x), P(x) \to$ and $\{1\}\colon P(a)$.
- In step 4, the clause labels $\{1, 9\}$ and $\{1, 10\}$ are collapsed into $\{1\}$.
- Finally, two Backjump steps followed by Enter-right yield the last tree, which corresponds to the split stack $\langle(2, \emptyset, \emptyset, \{\{1\}\})\rangle$.

Observe how the empty clause generated in step 4 allows us to skip branch 4 and jump directly to branch 2, which would not be possible without the Right-collapse rule.

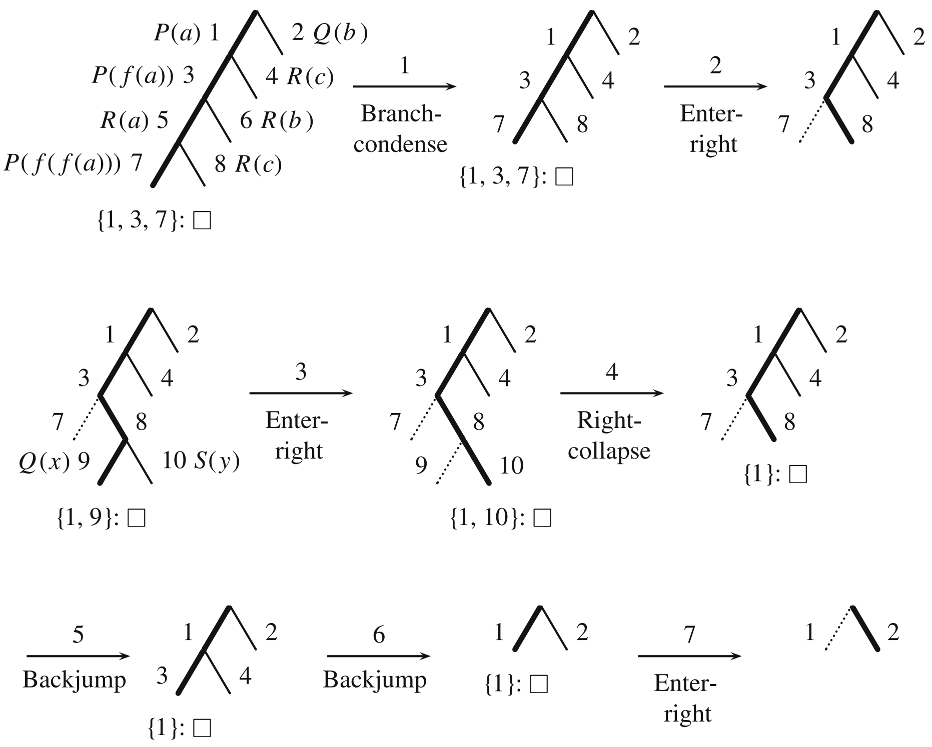

Fig. 2 Split tree development from Example 2.13

2.3 Correctness

In the following, we introduce the concept of satisfiability of labelled clause sets, and prove soundness of the calculus. In Section 2.3.5, we present a notion of fairness that takes splitting into account, and we prove refutational completeness of the calculus with respect to the fairness property.

A *derivation* in the labelled calculus is a sequence of labelled clause sets

$$\mathcal{D} = \Psi_0 : N_0 \ \rhd\ \Psi_1 : N_1 \ \rhd\ \Psi_2 : N_2 \ \rhd\ \dots$$

such that $\Psi_0 = \langle\rangle$, N_0 is the initial labelled clause set and for each $i \geq 1$, the labelled clause set $\Psi_i : N_i$ is the result of applying a rule of the calculus to $\Psi_{i-1} : N_{i-1}$. We call the step $\Psi_{i-1} : N_{i-1} \ \rhd\ \Psi_i : N_i$ the ith step of $\mathcal{D}$. We write $\Psi : N \rhd^* \Psi' : N'$ to indicate that $\Psi' : N'$ is obtained from $\Psi : N$ by zero or more steps. We use superscripts to make explicit that a split belongs to a particular split stack in a derivation: for example, we write ψ_j^i for the split of Ψ_i with index j, and D_j^i for the deleted set of ψ_j^i. Furthermore, we write D^i for $\bigcup_{j=1}^n D_j^i$, where n is the length of Ψ_i.

When a split ψ_j is created by splitting a clause $\Gamma_1, \Gamma_2 \to \Delta_1, \Delta_2$ into components $\Gamma_1 \to \Delta_1$ and $\Gamma_2 \to \Delta_2$, its level l_j is an odd number and it is easy to show that it is different from all other split levels in the stack. During backtracking, ψ_j is either removed (by Backjump or Branch-condense), or replaced by a split ψ_j' with level $l_j + 1$ (by Enter-right). Thus for any split stack Ψ containing ψ_j, the levels l_j and $l_j + 1$ uniquely correspond to clauses $\Gamma_1 \to \Delta_1$ and $\Gamma_2 \to \Delta_2$, respectively. We capture this correspondence in the function clause_Ψ:

$$\begin{aligned} \text{clause}_\Psi(l_j) &= \Gamma_1 \to \Delta_1 \\ \text{clause}_\Psi(l_j + 1) &= \Gamma_2 \to \Delta_2 \end{aligned}$$

In general, $\text{clause}_\Psi(l)$ returns the first or second component associated with the split at $\text{index}_\Psi(l)$, depending on whether $\text{left}(l)$ or $\text{right}(l)$ holds. Clearly, for any $\Psi_i : N_i$ occurring in a derivation, and where step $i + 1$ is not a backtracking step applying Backjump or Branch-condense to the split at $\text{index}_{\Psi_i}(l)$, it holds that $\text{clause}_{\Psi_i}(l) = \text{clause}_{\Psi_{i+1}}(l)$. We use $\text{clauses}_\Psi(L)$ as shorthand for $\{\text{clause}_\Psi(l) \mid l \in L\}$.

2.3.1 Satisfiability of labelled clause sets

In order to prove soundness and completeness results for our labelled calculus, we need to extend the notion of satisfiability from clause sets to labelled clause sets. Since we are exploring a tree whose branches represent alternatives of successive case distinctions, we associate a clause set with each unexplored branch on the stack. Formally, let $\Psi_i : N_i$ be a labelled clause set occurring in a derivation where Ψ_i is of length n. For each $1 \leq k \leq n$ with $\text{left}(\text{level}_{\Psi_i}(k))$, we define

$$N_i^k := \left(N_i \cup D^i\right)\big|_{L_k} \cup B_k$$

where $L_k = \{\text{level}_{\Psi_i}(1), \dots, \text{level}_{\Psi_i}(k-1)\}$. We call N_i the *active clause set* of $\Psi_i : N_i$, and the N_i^k the *inactive clause sets* of $\Psi_i : N_i$. We denote by $\mathcal{N}_i$ the set of all inactive sets of $\Psi_i : N_i$.

Definition 13 (Satisfiability of labelled clause sets) We say that $\Psi_i : N_i$ is *satisfiable*, if and only if N_i is satisfiable, or some $N_i^k \in \mathcal{N}_i$ is satisfiable.

In the following, we show the labelled calculus to be sound, that is, we show that the rules of the labelled calculus preserve satisfiability of labelled clause sets.

2.3.2 Label-validity

The *label-validity* property says that clause labels and leaf markers in a labelled clause set only refer to existing splits. The property implies that $\mathrm{index}_\Psi(l)$ is well-defined for any $l \in L$ whenever L is a clause label or $\{L\}$ is a leaf marker in Ψ.

Definition 14 (Label-validity) Let $\Psi_i : N_i$ be a labelled clause set.

1. We call the active set N_i *label-valid*, if $L \subseteq \mathrm{levels}(\Psi_i)$ for every $L : C \in N_i$,
2. we call some inactive set $N_i^k \in \mathcal{N}_i$ *label-valid*, if $L \subseteq \{\mathrm{level}_{\Psi_i}(1), \ldots, \mathrm{level}_{\Psi_i}(k)\}$ for every $L : C \in N_i^k$,
3. we call a leaf-marker $\varphi_j = \{L\}$ *label-valid*, if $L \subseteq \{\mathrm{level}_{\Psi_i}(1), \ldots, \mathrm{level}_{\Psi_i}(j-1), \mathrm{level}_{\Psi_i}(j) - 1\}$.

We call the labelled clause set $\Psi_i : N_i$ *label-valid*, if its active set, all its inactive sets and all its leaf markers are label-valid.

The following Lemma states that clause labels are correctly inherited throughout derivations:

Lemma 1 (Label monotonicity) *Let $\mathcal{D}$ be a derivation, $\Psi_i : N_i$ an arbitrary labelled clause set in $\mathcal{D}$, and $l \in \mathrm{levels}(\Psi_i)$, and let k be maximal such that $k \leq i$ and the splitting rule was applied at step k of $\mathcal{D}$ with l as the split level of the conclusion. Furthermore, let L be the label of the premise at step k. Then for any $L' : C \in N_i \cup D^i$ with $l \in L'$, we have $L \subseteq L'$.*

Proof It is easy to see that the property is maintained by all stack reduction rules, since no new clauses are added, except for rule Right-collapse, in which case the statement easily follows from the definition of the new empty clause's label. Now consider the calculus rules. The only interesting cases are inferences, but again, it is easy to see that the property is maintained by the definition of the label of the conclusion. □

Lemma 2 (Split deletion and label-validity) *Let $\Psi : N$ be an arbitrary labelled clause set, let $m \in \{1, \ldots, n\}$ be arbitrary and assume that for all $j > m$, the parent clause of split j does not depend on l_m, and that $\varphi_j = \emptyset$. Then* $\mathrm{delete}(\Psi : N, m)$ *is label-valid if $\Psi : N$ is label-valid.*

Proof Let $\Psi' : N' := \mathrm{delete}(\Psi : N, m)$.

1. Label-validity of N' is trivial.
2. Let $k \in \{1, \ldots, n-1\} \setminus \{m\}$ be arbitrary. Splitting is the only rule extending the blocked set B_k', and by definition of the splitting rule, all clauses in B_k' have the label $L \cup \{l_k + 1\}$, where L is the label of the parent clause of split k. Hence by assumption, no clause in B_k' depends on l_m. Therefore $N'^k = (N' \cup \bigcup_{j=1}^{n} D_j')\big|_{\mathrm{levels}(\Psi')} \cup B_k'$ is label-valid.

3. Let $k \in \{1, \ldots, n-1\} \setminus \{m\}$ again be arbitrary. If $k < m$, then φ'_k is label-valid by assumption. If $k \geq m$, then $\varphi'_k = \emptyset$ by assumption, and hence φ' is trivially label-valid. □

Lemma 3 (Stack reductions maintain label-validity) *Let $\Psi_i : N_i$ be an arbitrary labelled clause set in a derivation, and assume that $\Psi_i : N_i \rightarrow \Psi'_i : N'_i$. Then $\Psi'_i : N'_i$ is label-valid if $\Psi_i : N_i$ is label-valid.*

Proof Assume $\Psi_i : N_i$ is label-valid. The cases Backjump, Branch-condense and Enter-right follow directly by Lemma 2. Note that for the Branch-condense case, the assumptions of Lemma 2 are fulfilled: since $l_{k_{max}} = \text{level}_{\Psi_i}(k_{max})$ is the greatest left level that the empty clause doesn't depend on, it follows by Lemma 1 that the parent clauses of all splits with indices greater than k_{max} don't depend on $l_{k_{max}}$ either. Furthermore we have $\varphi_j = \emptyset$, since left(l_j) holds for all $j > k_{max}$, (the only rule modifying leaf markers, Enter-right, only produces right splits). For the Right-collapse case, note that $L \subseteq \text{levels}(\Psi')$ for the new empty clause $L : \Box$, since both L_1 and L_2 are subsets of levels(Ψ). □

We now show that label-validity is an invariant of any derivation.

Proposition 1 (Label-validity in derivations) *Any labelled clause set in any derivation is label-valid.*

Proof Let $\Psi_0 : N_0 \rhd \Psi_1 : N_1 \rhd \ldots$ be an arbitrary derivation. We show that any $\Psi_i : N_i$ is label-valid by induction over the derivation. Clearly, $\Psi_0 : N_0$ is label-valid, since Ψ_0 is empty. For the inductive case, we assume $\Psi_{i-1} : N_{i-1}$ is label-valid and distinguish which calculus rule was applied to obtain $\Psi_i : N_i$. For backtracking, the result follows from Lemma 3. In the case of splitting, we know by induction hypothesis that $L \subseteq \text{levels}(\Psi_{i-1})$, hence also $L' \subseteq \text{levels}(\Psi_i)$. Label-validity of the N_i^k and the leaf markers is obvious. For inferences, note that the union of two valid labels is again valid. The case for reductions is trivial. □

2.3.3 Path-validity

We now define a property of labelled clause sets, *path-validity*, which states that every clause in a labelled clause set follows logically from the initial clause set and the split components described by its label, and that the initial clause set together with the split components described by a leaf marker is unsatisfiable.

Definition 15 (Path-validity) Let $\Psi_i : N_i$ be a labelled clause set in a derivation. We call $\Psi_i : N_i$ *path-valid*, if

1. $N_0 \cup \text{clauses}_{\Psi_i}(L) \models C$ for every $L : C$ in N or some deleted set of Ψ_i, and
2. $N_0 \cup \text{clauses}_{\Psi_i}(L) \models \bot$ for every leaf marker $\{L\}$ in Ψ_i.

Lemma 4 (Stack reductions maintain path-validity) *Let $\Psi_i : N_i$ be an arbitrary labelled clause set in a derivation, and assume that $\Psi_i : N_i \rightarrow \Psi'_i : N'_i$. Then $\Psi'_i : N'_i$ is path-valid if all $\Psi_j : N_j$, $j \leq i$ are path-valid.*

Proof Assume all $\Psi_j : N_j$ are path-valid, for $j \leq i$. We distinguish which rule was applied to obtain $\Psi_i' : N_i'$. The cases BACKJUMP and BRANCH-CONDENSE follow immediately by assumption, since $(N_i' \cup \bigcup_{j=1}^{n-1} D_j') \subseteq (N_i \cup \bigcup_{j=1}^{n} D_j)$, and leaf markers are not extended.

RIGHT-COLLAPSE: We show that path-validity is maintained by the addition of the new empty clause $L{:}\,\Box$, that is, $N_0 \cup \text{clauses}_{\Psi_i'}(L_1 \cup L_2 \setminus \{l_n - 1, l_n\}) \models \bot$. Assume the kth step of the derivation was the splitting step that produced split n and let L' be the label of the parent clause. Let l_n denote $\text{level}_{\Psi_i}(n)$. Note that $\text{right}(l_n)$ holds, the corresponding left level is $l_n - 1$. By path-validity of $\Psi_{k-1} : N_{k-1}$, and because $\text{clauses}_{\Psi_i}(L') = \text{clauses}_{\Psi_{k-1}}(L')$, it holds that

$$N_0 \cup \text{clauses}_{\Psi_i}\left(L'\right) \models \text{clause}_{\Psi_i}\left(l_n - 1\right) \vee \text{clause}_{\Psi_i}\left(l_n\right).$$

By path-validity of $\Psi_i : N_i$, we have

$$N_0 \cup \text{clauses}_{\Psi_i}\left(L_1\right) \models \bot \;\text{ and }\; N_0 \cup \text{clauses}_{\Psi_i}\left(L_2\right) \models \bot.$$

Or, equivalently

$$N_0 \cup \text{clauses}_{\Psi_i}\left(L_1 \setminus \{l_n - 1\}\right) \models \neg\, \text{clause}_{\Psi}\left(l_n - 1\right)$$

and

$$N_0 \cup \text{clauses}_{\Psi_i}\left(L_2 \setminus \{l_n\}\right) \models \neg\, \text{clause}_{\Psi}\left(l_n\right).$$

By label monotonicity, $L' \subseteq L_1$ and $L' \subseteq L_2$, therefore

$$\begin{aligned} N_0 \cup \text{clauses}_{\Psi_i}\left(L_1 \cup L_2 \setminus \{l_n - 1, l_n\}\right) \models\ & \text{clause}_{\Psi_i}\left(l_n - 1\right) \vee \text{clause}_{\Psi_i}\left(l_n\right) \\ & \wedge \neg\, \text{clause}_{\Psi}\left(l_n - 1\right) \\ & \wedge \neg\, \text{clause}_{\Psi}\left(l_n\right) \end{aligned}$$

and thus $N_0 \cup \text{clauses}_{\Psi_i}(L_1 \cup L_2 \setminus \{l_n - 1, l_n\}) \models \bot$.

ENTER-RIGHT: By assumption, $N_0 \cup \text{clauses}_{\Psi_i}(L) \models \bot$ for the empty clause $L{:}\,\Box$, hence the new leaf marker $\{L\}$ in Ψ_i' is also path-valid. □

Proposition 2 (Path-validity in derivations) *Let $\mathcal{D} = \Psi_0 : N_0 \rhd \Psi_1 : N_1 \rhd \ldots$ be an arbitrary derivation. Then any $\Psi_i : N_i$ in $\mathcal{D}$ is path-valid.*

Proof By induction over $\mathcal{D}$. Clearly, $\Psi_0 : N_0$ is path-valid. Assume all $\Psi_j : N_j$, $j < i$ are path-valid and distinguish which rule was applied to obtain $\Psi_i : N_i$. The splitting case is trivial, and the backtracking case follows from Lemma 4. For inferences, assume the premises are $L_1 : C_1, \ldots, L_m : C_m$ and the conclusion is $L : C$. By assumption, $N_0 \cup \text{clauses}_{\Psi_i}(L_1 \cup \cdots \cup L_m) \models C_1 \wedge \cdots \wedge C_m$. Since the conclusion follows logically from the premises, $N_0 \cup \text{clauses}_{\Psi_i}(L) \models C$. For reductions, the statement follows from the fact that $N_i \cup D^i \subseteq N_{i-1} \cup D^{i-1}$. □

Corollary 1 (Restriction expansion) *Let $\Psi_i : N_i$ be an arbitrary labelled clause set in a derivation, $M \subseteq \text{levels}(\Psi_i)$ and $l \in \text{levels}(\Psi_i)$. If $(N_i \cup D^i)\big|_M \cup \{\text{clause}_{\Psi_i}(l)\}$ is satisfiable, then $(N_i \cup D^i)\big|_{M \cup \{l\}}$ is satisfiable.*

Proof By Proposition 2, any clause $L : C$ with $l \in L$ in $(N_i \cup D^i)\big|_{M\cup\{l\}}$ follows logically from $(N_i \cup D^i)\big|_M \cup \{\text{clause}_{\Psi_i}(l)\}$. □

Proposition 3 (Redundancy of deleted clauses) *Let $\mathcal{D}$ be a derivation, $\Psi_k : N_k$ an arbitrary labelled clause set in $\mathcal{D}$, and let $L\colon C$ be a clause contained in some deleted set of Ψ_k. Furthermore, let k' be maximal such that $L\colon C$ was reduced at step k' of $\mathcal{D}$. Then for all $k \le i \le k'$, $L\colon C$ is redundant with respect to N_i.*

Proof Assume the clause $L\colon C$ is contained in the deleted set of the split with index j in Ψ_k, i.e., $L\colon C \in D^k_j$. Hence the clause reducing $L\colon C$ in step k', call it $L_1\colon C_1$, has split level $l_j := \max(L_1) = \text{level}_{\Psi_{k'}}(j)$. By maximality of k', we know that for all $k \le i \le k'$, $L\colon C \in D^i_{j_i}$ with $j_i = \text{index}_{\Psi_i}(l_j)$. Therefore the split with level l_j is never deleted in steps $k'+1$ to k (otherwise $L\colon C$ would be reinserted). Hence for all $k \le i \le k'$, either $L_1\colon D_1 \in N_i$, or some other clause $L_2\colon D_2$ reduced $L_1\colon D_1$. In general, there may be several $L_m\colon D_m$ ($m \ge 2$) such that each $L_m\colon D_m$ subsumes $L_{m-1}\colon D_{m-1}$. But any such $L_m\colon D_m$ also subsumes $L\colon C$, hence $L\colon C$ is redundant in N_i. □

Corollary 2 (Active clause sets and deleted clauses) *Let $\Psi_i : N_i$ be an arbitrary labelled clause set in a derivation, where Ψ_i has length n. Then N_i is satisfiable if and only if $N_i \cup D^i$ is satisfiable.*

Proof Follows directly from Proposition 3. □

Lemma 5 (Split deletion and inactive clause sets) *Let $\Psi : N$ be an arbitrary labelled clause set with Ψ of length n, and let $m \in \{1, \ldots, n\}$ be such that for all $j > m$, the parent clause of split j does not depend on* $\text{level}_\Psi(m)$*. Then for $\Psi' : N' =$* $\text{delete}(\Psi : N, m)$ *and for all $1 \le k < n$,*

$$N'^k = \begin{cases} N^k & \text{if } k < m \\ N^{k+1}\big|_{\overline{\{\text{level}_\Psi(m)\}}} & \text{if } k \ge m. \end{cases}$$

Proof We write l_i for $\text{level}_\Psi(i)$ and l'_i for $\text{level}_{\Psi'}(i)$. Assume $k < m$, and let $L := \{l'_1, \ldots, l'_{k-1}\} = \{l_1, \ldots, l_{k-1}\}$. Then

$$\begin{aligned}
N'^k &= \left(N' \cup \bigcup_{j=1}^{n-1} D'_j\right)\Bigg|_L \cup B'_k \\
&= \left(\left(N \cup D_m \cup \bigcup_{l_j \in R} D_j\right)\Bigg|_{\text{levels}(\Psi')} \cup \bigcup_{j=1}^{n-1} D'_j\right)\Bigg|_L \cup B_k && (1)\\
&= \left(\left(N \cup D_m \cup \bigcup_{l_j \in R} D_j \cup \bigcup_{j=1}^{m-1} D_j \cup \bigcup_{j=m+1}^{n} D_j\right)\Bigg|_{\text{levels}(\Psi')}\right)\Bigg|_L \cup B_k && (2)\\
&= \left(N \cup \bigcup_{j=1}^{n} D_j\right)\Bigg|_L \cup B_k && (3)\\
&= N^k.
\end{aligned}$$

Equation 1 is obtained by replacing N' by its definition, and observing that $B'_k = B_k$. For Eq. 2, we use the fact that

$$\bigcup_{j=1}^{n-1} D'_j = \bigcup_{j=1}^{m-1} D_j\big|_{\text{levels}(\Psi')} \cup \bigcup_{j=m+1}^{n} D_j\big|_{\text{levels}(\Psi')}$$

because of level shifting. Finally, Eq. 3 follows from the fact that $L \subseteq \text{levels}(\Psi')$.

Now assume $k \geq m$, and let $L := \{l'_1, \ldots, l'_{k-1}\}$. Note that

$$L = \begin{cases} \{l_1, \ldots, l_{m-1}\} & \text{if } k = m \\ \{l_1, \ldots, l_{m-1}, l_{m+1}, \ldots, l_k\} & \text{if } k > m \end{cases}$$

or, in other words, $L = \{l_1, \ldots, l_k\} \setminus \{l_m\}$. Hence

$$\begin{aligned} N'^k &= \left(N' \cup \bigcup_{j=1}^{n-1} D'_j \right)\Bigg|_L \cup B'_k \\ &= \left(\left(N \cup D_m \cup \bigcup_{l_j \in R} D_j \cup \bigcup_{j=1}^{m-1} D_j \cup \bigcup_{j=m+1}^{n} D_j \right)\Bigg|_{\text{levels}(\Psi')} \right)\Bigg|_L \cup B_{k+1} \qquad (4) \\ &= \left(N \cup \bigcup_{j=1}^{n} D_j \right)\Bigg|_{\{l_1,\ldots,l_k\}\setminus\{l_m\}} \cup B_{k+1} \qquad (5) \\ &= N^{k+1}\big|_{\overline{\{l_m\}}}. \end{aligned}$$

We again use $L \subseteq \text{levels}(\Psi')$ for Eq. 5, and the assumption that the splits below m do not depend on l_m. □

2.3.4 Soundness

We now show that satisfiability of labelled clause set is preserved in derivations, as discussed in Section 2.3.1. We again begin by showing that satisfiability is preserved by each of the stack reduction rules.

Lemma 6 (Backjump maintains satisfiability) *Let $\Psi_i : N_i$ be an arbitrary labelled clause set in a derivation, and assume $\Psi_i : N_i \rightarrow \Psi'_i : N'_i$ with rule* Backjump. *Then $\Psi_i : N_i$ is satisfiable if and only if $\Psi'_i : N'_i$ is satisfiable.*

Proof Clearly, N_i is unsatisfiable, since $L{:}\,\Box \in N_i$. Furthermore, if $N_i^n \in \mathcal{N}_i$ exists (this is the case if left(l_n)) then $L{:}\,\Box \in N_i^n$, since $l_n \notin L$, and thus N_i^n is unsatisfiable. Hence $\Psi_i : N_i$ is satisfiable if and only if some $N_i^k \in \mathcal{N}_i$ is satisfiable, for $k < n$. On the other hand, $L{:}\,\Box \in N'_i$ since $l_n \notin L$, thus N'_i is also unsatisfiable. Therefore, $\Psi'_i : N'_i$ is satisfiable if and only if some $N_i'^k \in \mathcal{N}'_i$ is satisfiable. The statement then follows directly with Lemma 5. □

Lemma 7 (Branch-condense maintains satisfiability) *Let $\Psi_i : N_i$ be an arbitrary labelled clause set in a derivation, and assume $\Psi_i : N_i \to \Psi_i' : N_i'$ with rule* Branch-condense. *Then $\Psi_i : N_i$ is satisfiable if and only if $\Psi_i' : N_i'$ is satisfiable.*

Proof We write l_k for $\text{level}_{\Psi_i}(k)$. Again, N_i is unsatisfiable, since $L\colon \Box \in N_i$. Because $l_{k_{max}} \notin L$, we also know $L\colon \Box \in N_i'$, and thus N_i' is unsatisfiable. By Lemma 5, we know that $N_i^k = N_i'^k$ for $k < k_{max}$. Hence it suffices to show that there exists a satisfiable $N_i^k \in \mathcal{N}_i$ with $k \in \{k_{max}, \dots, n\}$ if and only if there exists a satisfiable $N_i'^{k'} \in \mathcal{N}_i'$ with $k' \in \{k_{max}, \dots, n-1\}$. Also note that since $\text{left}(l_k)$ for all $k \in \{k_{max}, \dots, n\}$, we have $\mathcal{N}_i = \{N_i^{k_{max}}, N_i^{k_{max}+1}, \dots, N_i^n\}$ (see Fig. 3).

For the forward direction, let us first assume that $N_i^{k_{max}}$ is satisfiable. We show that there exists a satisfiable $N_i'^k \in \mathcal{N}_i'$ with $k \in \{k_{max}, \dots, n-1\}$, by induction on k. For each k, we show that either

1. $N_i'^k$ is satisfiable, or
2. $(N_i \cup D^i)\big|_{\{l_1,\dots,l_{k+1}\}\setminus\{l_{k_{max}}\}}$ is satisfiable.

Since we know $(N_i \cup D^i)\big|_{\{l_1,\dots,l_n\}\setminus\{l_{k_{max}}\}}$ is unsatisfiable, it follows that there must be a satisfiable $N_i'^k \in \mathcal{N}_i'$.

We have assumed $N_i^{k_{max}} = (N_i \cup D^i)\big|_{\{l_1,\dots,l_{k_{max}-1}\}} \cup B_{k_{max}}$ to be satisfiable, hence also $N_i \cup D^i\big|_{\{l_1,\dots,l_{k_{max}-1}\}}$ is satisfiable – this provides the induction base.

For the inductive step, let $k \geq k_{max}$ and assume that

$$\left(N_i \cup D^i\right)\big|_{\{l_1,\dots,l_k\}\setminus\{l_{k_{max}}\}}$$

is satisfiable. Let L' be the label of the parent clause of split of the $k+1$st split of Ψ_i' (i.e., the kth split of Ψ_i). Since $l_{k+1} \in L$, we know by Lemma 1 that $L' \subseteq L$. Since $l_{k_{max}} \notin L$, we also have $l_{k_{max}} \notin L'$. So by path-validity (Proposition 2), we know that

$$\left(N_i \cup D^i\right)\big|_{\{l_1,\dots,l_k\}\setminus\{l_{k_{max}}\}} \models \text{clause}_{\Psi_i}\left(l_{k+1}\right) \vee \text{clause}_{\Psi_i}\left(l_{k+1}+1\right)$$

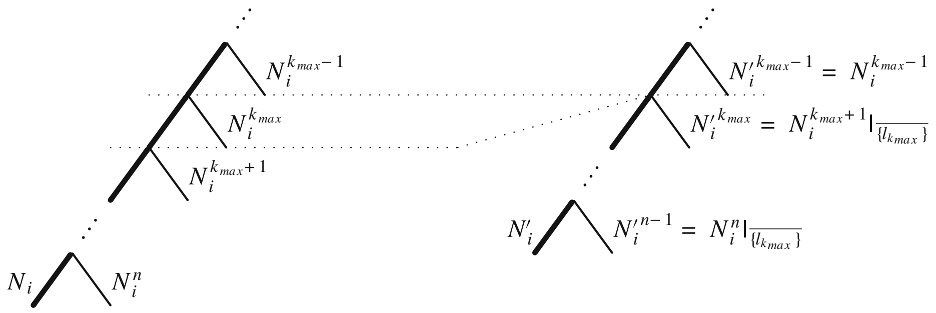
$N_i^{k_{max}-1}$
$N_i^{k_{max}}$
$N_i^{k_{max}+1}$
N_i
N_i^n
$N_i'^{k_{max}-1} = N_i^{k_{max}-1}$
$N_i'^{k_{max}} = N_i^{k_{max}+1}|_{\overline{\{l_{k_{max}}\}}}$
N_i'
$N_i'^{n-1} = N_i^n|_{\overline{\{l_{k_{max}}\}}}$

Fig. 3 Illustration of the proof of Lemma 7

(remember that left(l_k)). Hence either

1. $(N_i \cup D^i)\big|_{\{l_1,\dots,l_k\}\setminus\{l_{k_{max}}\}} \cup \{\text{clause}_{\Psi_i}(l_{k+1})\}$ is satisfiable, but then by Corollary 1, also
$$\big(N_i \cup D^i\big)\big|_{\{l_1,\dots,l_k,l_{k+1}\}\setminus\{l_{k_{max}}\}}$$
is satisfiable.
2. Or $(N_i \cup D^i)\big|_{\{l_1,\dots,l_k\}\setminus\{l_{k_{max}}\}} \cup \{\text{clause}_{\Psi_i}(l_{k+1}+1)\} = N_i^{k+1}\big|_{\overline{\{l_{k_{max}}\}}}$ is satisfiable. By Lemma 5, we have $N_i^{k+1}\big|_{\overline{\{l_{k_{max}}\}}} = N_i'^k$, hence $N_i'^k$ is satisfiable.

Still for the forward direction, assume now that some N_i^k is satisfiable, for $k \in \{k_{max}+1,\dots,n\}$. Then it immediately follows with Lemma 5 that $N_i'^{k-1} = N_i^k\big|_{\overline{\{l_{k_{max}}\}}}$ is satisfiable.

For the backward direction, assume $N_i'^k$ is satisfiable, for some $k \in \{k_{max},\dots, n-1\}$. Again by Lemma 5,
$$N_i'^k = N_i^{k+1}\big|_{\overline{\{l_{k_{max}}\}}} = \big(N_i \cup D^i\big)\big|_{\{l_1,\dots,l_k\}\setminus\{l_{k_{max}}\}} \cup B_{k+1}$$
is satisfiable. By path-validity (Proposition 2), we know that
$$\big(N_i \cup D^i\big)\big|_{\{l_1,\dots,l_{k_{max}-1}\}} \models \text{clause}_{\Psi_i}\big(k_{max}\big) \vee \text{clause}_{\Psi_i}\big(k_{max}+1\big).$$
Hence we again distinguish two cases:

1. Either $(N_i \cup D^i)\big|_{\{l_1,\dots,l_k\}\setminus\{l_{k_{max}}\}} \cup \{\text{clause}_{\Psi_i}(k_{max})\} \cup B_{k+1}$ is satisfiable. But then also
$$\big(N_i \cup D^i\big)\big|_{\{l_1,\dots,l_k\}} \cup B_{k+1} = N_i^{k+1}$$
is satisfiable.
2. Or $(N_i \cup D^i)\big|_{\{l_1,\dots,l_k\}\setminus\{l_{k_{max}}\}} \cup \{\text{clause}_{\Psi_i}(k_{max}+1)\} \cup B_{k+1}$ is satisfiable, hence also
$$\big(N_i \cup D^i\big)\big|_{\{l_1,\dots,l_{k_{max}-1}\}} \cup B_{k_{max}}$$
is satisfiable, since $B_{k_{max}} = \{\text{clause}_{\Psi_i}(k_{max}+1)\}$. □

Lemma 8 (Right-collapse maintains satisfiability) *Let $\Psi_i : N_i$ be an arbitrary labelled clause set in a derivation, and assume $\Psi_i : N_i \to \Psi_i' : N_i'$ with rule* Right-collapse. *Then $\Psi_i : N_i$ is satisfiable if and only if $\Psi_i' : N_i'$ is satisfiable.*

Proof First note that $L_2{:}\,\square \in N_i$ and $L{:}\,\square \in N_i'$, hence both N_i and N_i' are unsatisfiable. Thus it suffices to show that there exists a satisfiable $N_i'^{k'} \in \mathcal{N}_i'$ if and only if there is a satisfiable $N_i^k \in \mathcal{N}_i$. By Lemma 5 and definition of Right-collapse, we obtain that for all $N_i'^k \in \mathcal{N}_i'$,
$$N_i'^k = \begin{cases} N_i^k \cup \{L{:}\,\square\} & \text{if } k > \max(L) \\ N_i^k & \text{otherwise.} \end{cases}$$
For the first case, use the fact that for $k > \max(L)$, we have $L \subseteq \{1,\dots,k\}$, and path-validity (Proposition 2). The second case is immediate. □

Lemma 9 (Enter-right maintains satisfiability) *Let $\Psi_i : N_i$ be an arbitrary labelled clause set in a derivation, and assume $\Psi_i : N_i \to \Psi_i' : N_i'$ with rule* Enter-right. *Then $\Psi_i : N_i$ is satisfiable if and only if $\Psi_i' : N_i'$ is satisfiable.*

Proof Again because of the empty clause, N_i is unsatisfiable. Furthermore, by Lemma 5, $N_i'^k = N_i^k$ for $k < n$. Thus it suffices to show that N_i^n is satisfiable if and only if N_i' is satisfiable. The forward direction is immediate since $N_i' \subseteq N_i^n$. The backward direction follows from Proposition 3. □

Theorem 1 (Soundness) *Let N be an arbitrary clause set. Let $N_0 := \{\emptyset : C \mid C \in N\}$ be the associated set of labelled clauses, let $\Psi_0 := \langle\rangle$, and let $\Psi_0 : N_0 \rhd^* \Psi_m : N_m$ be an arbitrary derivation starting with $\Psi_0 : N_0$. Then $\Psi_m : N_m$ is satisfiable if and only if $\Psi_0 : N_0$ is satisfiable.*

Proof By induction over the derivation. Let $i \geq 0$ and assume $\Psi_i : N_i \rhd \Psi_{i+1} : N_{i+1}$. We distinguish which rule was applied to obtain $\Psi_{i+1} : N_{i+1}$. For inferences and reductions, the proof is trivial. For backtracking, the proof proceeds by induction over the stack reductions, using Lemmata 6 to 9. For the splitting case, let us first observe that Ψ_{i+1} has length $n + 1$, and that for all $1 \leq k \leq n$,

$$\begin{aligned} N_{i+1}^k &= \left(N_{i+1} \cup \bigcup_{j=1}^{n+1} D_j^{i+1}\right)\Bigg|_{\{l_1,\ldots,l_{k-1}\}} \cup B_k^{i+1} \\ &= \left(N_i \cup \bigcup_{j=1}^{n} D_j^{i}\right)\Bigg|_{\{l_1,\ldots,l_{k-1}\}} \cup B_k^{i} \\ &= N_i^k. \end{aligned}$$

Thus it suffices to show that N_i is satisfiable if and only if N_{i+1} or N_{i+1}^{n+1} is satisfiable. By definition of the splitting rule, we have $N_{i+1} = N_i \cup \{L' : \Gamma_1 \to \Delta_1\}$ and

$$\begin{aligned} N_{i+1}^{n+1} &= \left(N_{i+1} \cup \bigcup_{j=1}^{n+1} D_j^{i+1}\right)\Bigg|_{\{l_1,\ldots,l_n\}} \cup B_{n+1}^{i+1} \\ &= \left(N_i \cup \bigcup_{j=1}^{n} D_j^{i}\right)\Bigg|_{\{l_1,\ldots,l_n\}} \cup \left\{L' : \Gamma_2 \to \Delta_2\right\} \end{aligned}$$

since $l_n + 1 \in L'$ and thus $N_{i+1}\big|_{\{l_1,\ldots,l_n\}} = N_i$, and the sets of deleted clauses are not changed by splitting. By Corollary 2, satisfiability of $N_i \cup \bigcup_{j=1}^{n} D_j^i$ is equivalent to satisfiability of N_i, hence N_{i+1}^{n+1} is satisfiable if and only if $N_i \cup \{L' : \Gamma_2 \to \Delta_2\}$. Therefore, it suffices to show that N_i is satisfiable if and only if $N_i \cup \{L' : \Gamma_1 \to \Delta_1\}$ or $N_i \cup \{L' : \Gamma_2 \to \Delta_2\}$ is satisfiable, and this is immediate since $\Gamma_1 \to \Delta_1$ and $\Gamma_2 \to \Delta_2$ are variable-disjoint. □

2.3.5 Completeness

When backtracking is modelled explicitly, as we have done here, classical model construction techniques (as in [2, 15]) cannot directly be used to show completeness of the calculus. In particular, defining a fair derivation to be a derivation in which any non-redundant inference from persistent clauses is eventually computed, no longer guarantees that any fair derivation from an inconsistent clause set eventually yields a contradiction. The reason is that in our calculus, the changes to the clause set are not monotonic (in the sense that in each step, only derived clauses are added or redundant clauses are removed).

Example 2 Consider the unsatisfiable clause set

$$\begin{aligned} N_0 = \{\ & S(a); \qquad \neg S(a); \qquad P(a); \\ & P(x) \rightarrow Q(y), P(f(x)); \\ & Q(y) \rightarrow S(x) \qquad\qquad \} \end{aligned}$$

where all clauses have the empty label. We construct an infinite derivation $\mathcal{D} = \Psi_0 : N_0 \rhd \Psi_1 : N_1 \rhd \ldots$ as follows: we apply resolution to derive a clause $\rightarrow Q(y)$, $P(f^{n+1}(a))$ (initially, $n = 0$) which we then split. In the left branch, we use $Q(y)$ to infer $S(x)$. We then apply subsumption deletion to $S(x)$ and $S(a)$, removing $S(a)$ from the clause set and storing it in the deleted set of the current split. We apply resolution to infer the empty clause from $S(x)$ and $\neg S(a)$ and backtrack, entering the right branch, reinserting clause $S(a)$. We then repeat the procedure with n increased by one (see Fig. 4). The classical definition of persistent clauses as $N_\infty = \bigcup_i \bigcap_{j \geq i} N_j$ yields a set that does not contain $S(a)$ (thus N_∞ is satisfiable), because for each subsumption deletion step k, we have $S(a) \notin N_k$, hence $S(a)$ is not "persistent" when viewed over the whole derivation. Every inference from clauses in N_∞ is computed, thus $\mathcal{D}$ is fair in the classical sense, but never derives a contradiction.

In the above example, the non-persistent clause $S(a)$ is redundant in all left branches. However, every left branch is refuted at some point, causing $S(a)$ to be reinserted upon backtracking. Thus, the fact that the clause is redundant only in branches that are eventually closed is irrelevant in an infinite derivation. In the example, the split tree has an infinite path consisting of right branches, and along this path, the clause $S(a)$ is not redundant and should therefore eventually be considered for inferences. Our goal is therefore to define a notion of fairness that ignores closed branches and

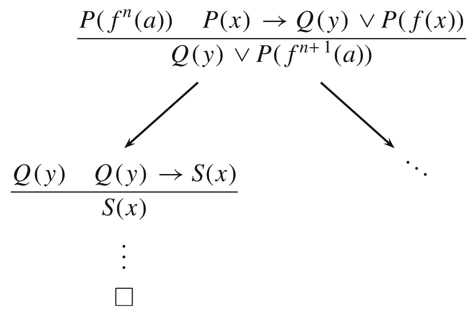
$P(f^n(a)) \quad P(x) \rightarrow Q(y) \vee P(f(x))$
$Q(y) \vee P(f^{n+1}(a))$
$Q(y) \quad Q(y) \rightarrow S(x)$
$S(x)$
$\vdots$
$\square$
$\ddots$

Fig. 4 Split tree of the derivation of Example 2

only talks about clauses on the infinite path of the split tree. In the following, we use $\mathrm{suf}_{\mathcal{D}}(i)$ to denote the suffix $\Psi_i : N_i \rhd \Psi_{i+1} : N_{i+1} \rhd \ldots$ of an infinite derivation $\mathcal{D} = \Psi_0 : N_0 \rhd \Psi_1 : N_1 \rhd \ldots$. Note that $\mathrm{suf}_{\mathcal{D}}(0) = \mathcal{D}$.

Definition 16 (Persistent split) Given an infinite derivation

$$\mathcal{D} = \Psi_0 : N_0 \rhd \Psi_1 : N_1 \rhd \ldots,$$

and $i \geq 1$, where $\Psi_i = \langle \psi_{n_i}, \ldots, \psi_1 \rangle$, we call ψ_k $(1 \leq k \leq n_i)$ *persistent in* $\mathrm{suf}_{\mathcal{D}}(i)$, if $l_k \in \mathrm{levels}(\Psi_j)$ for all $j \geq i$.

Definition 17 (Weakly persistent clause) Given an infinite derivation

$$\mathcal{D} = \Psi_0 : N_0 \rhd \Psi_1 : N_1 \rhd \ldots,$$

and $i \geq 0$, where $\Psi_i = \langle \psi_{n_i}, \ldots, \psi_1 \rangle$, we call a clause $L : C \in (N_i \cup D^i)$ *weakly persistent in* $\mathrm{suf}_{\mathcal{D}}(i)$, if for all $l_j \in L$, ψ_j is persistent in $\mathrm{suf}_{\mathcal{D}}(i)$.

Weakly persistent clauses are not necessarily contained in every clause set of the derivation from some point on. However, if a clause is weakly persistent, then from some point on, every clause set contains either the clause itself, or some clause that subsumes it. Also note that any clause that is weakly persistent in $\mathcal{D}$ is contained in N_0, since no split is persistent in $\mathcal{D}$.

Lemma 10 *For any infinite derivation $\mathcal{D}$ and for $i \geq 1$, if a clause $L : C \notin N_0$ is weakly persistent in* $\mathrm{suf}_{\mathcal{D}}(i)$*, then it is also weakly persistent in* $\mathrm{suf}_{\mathcal{D}}(k)$ *where $k \leq i$ is maximal such that step k of $\mathcal{D}$ has $L : C$ as its conclusion.*

Proof Let k be defined as above and assume $L : C$ is not weakly persistent in $\mathrm{suf}_{\mathcal{D}}(k)$. Then there exists $k \leq k' < i$ such that some split with level in L was deleted in step k' of $\mathcal{D}$. Thus by Definition 7, $L : C \notin (N_{k'} \cup D^{k'})$. Hence there must exist $k'' > k$ such that step k'' has $L : C$ as its conclusion, a contradiction. □

Definition 18 (Persistent step) Given an infinite derivation

$$\mathcal{D} = \Psi_0 : N_0 \rhd \Psi_1 : N_1 \rhd \ldots,$$

we say that step $i \geq 1$ of $\mathcal{D}$ is *persistent*, if

1. step i is a splitting or backtracking step, $\Psi_i = \langle \psi_{n_i}, \ldots, \psi_1 \rangle$, and ψ_{n_i} is persistent in $\mathrm{suf}_{\mathcal{D}}(i)$, or
2. step i is an inference or reduction step and all premises of the applied inference or reduction rule are weakly persistent in $\mathrm{suf}_{\mathcal{D}}(i-1)$.

Definition 19 Given an infinite derivation

$$\mathcal{D} = \Psi_0 : N_0 \rhd \Psi_1 : N_1 \rhd \ldots,$$

let $i \geq 0$ be a backtracking step, let l be the split level of the toplevel split of Ψ_i, and let $k < i$ be maximal such that step k of $\mathcal{D}$ is a splitting step producing a split of level $l-1$. Then we define $\mathrm{spt}_{\mathcal{D}}(i) := k$, the *associated splitting step* of backtracking step i.

Lemma 11 *For any infinite derivation $\mathcal{D} = \Psi_0 : N_0 \rhd \Psi_1 : N_1 \rhd \ldots$, if step $i \geq 1$ is a persistent splitting step with parent clause $L\colon C$ (a persistent backtracking step with $L\colon C$ the parent clause of the associated splitting step), then $L\colon C$ is weakly persistent in* $\mathrm{suf}_{\mathcal{D}}(i-1)$.

Proof Let l be the level of the split produced at step i. Assume $L\colon C$ is not weakly persistent. Let k be minimal such that $k > i$ and $l' \notin \mathrm{levels}(\Psi_k)$ for some $l' \in L$. Then step k of $\mathcal{D}$ is a backtracking step. Since the split produced at step i is persistent, the split with level l' must have been removed by application of Branch-condense. Hence there is an empty clause $L'\colon \Box \in N_{k-1}$ with $l \in L'$ but $l' \notin L'$, a contradiction to Lemma 1. □

Lemma 12 *For any infinite derivation $\mathcal{D}$ and for $i \geq 1$, if the conclusion of step i is weakly persistent in* $\mathrm{suf}_{\mathcal{D}}(i)$, *then all premises of step i are weakly persistent in* $\mathrm{suf}_{\mathcal{D}}(i-1)$.

Proof Follows from Definition 17 and Lemma 11. □

We will define the limit of an infinite derivation $\mathcal{D}$ to be the derivation obtained by starting from the original labelled clause set and applying exactly the persistent steps of $\mathcal{D}$. Note that all left splits are created by splitting, whereas all right splits are created by backtracking. The limit will contain no more applications of backtracking. Instead, whenever a persistent right branch is created in the original derivation, the limit will enter that branch directly, using the rule *splitting right*:

Definition 20 (Splitting right) The inference

$$\mathcal{I}\,\frac{\Psi : N \qquad L\colon \Gamma_1, \Gamma_2 \to \Delta_1, \Delta_2}{\Psi' : N \qquad\quad L'\colon \Gamma_2 \to \Delta_2}$$

where $\Psi = \langle \psi_n, \ldots, \psi_1 \rangle$, $L' = L \cup \{l_n+1\}$, $\Psi' = \langle \psi_{n+1}, \psi_n, \ldots, \psi_1 \rangle$ with $\psi_{n+1} = (l_n + 1, \emptyset, \emptyset, \emptyset)$, $\mathit{vars}(\Gamma_1 \to \Delta_1) \cap \mathit{vars}(\Gamma_2 \to \Delta_2) = \emptyset$, and $\Delta_1 \neq \emptyset$ and $\Delta_2 \neq \emptyset$ is called *splitting right.*

Definition 21 (Limit of a derivation) Given an infinite derivation

$$\mathcal{D} = \Psi_0 : N_0 \rhd \Psi_1 : N_1 \rhd \ldots,$$

let $i \geq 0$ be a backtracking step, let l be the split level of the toplevel split of Ψ_i, and let $k < i$ be maximal such that step k of $\mathcal{D}$ is a splitting step producing a split of level $l - 1$. We define the monotonic function $f_{\mathcal{D}} : \mathbb{N} \to \mathbb{N}$ as follows:

$$f_{\mathcal{D}}(0) := 0$$
$$f_{\mathcal{D}}(i+1) := \min\{j > f_{\mathcal{D}}(i) \mid \text{step } j \text{ of } \mathcal{D} \text{ is persistent}\}.$$

The *limit* of $\mathcal{D}$ is defined as

$$\lim(\mathcal{D}) := \Psi'_0 : N'_0 \rhd \Psi'_1 : N'_1 \rhd \Psi'_2 : N'_2 \rhd \dots$$

where $\Psi'_0 : N'_0 = \Psi_0 : N_0$ and $\Psi'_{i+1} : N'_{i+1}$ is obtained from $\Psi'_i : N'_i$ as follows:

1. if step $f_{\mathcal{D}}(i+1)$ of $\mathcal{D}$ is a backtracking step, consider the associated splitting step $\mathrm{spt}_{\mathcal{D}}(f_{\mathcal{D}}(i+1))$ with parent clause $L\colon \Gamma_1, \Gamma_2 \to \Delta_1, \Delta_2$. Then step $i+1$ of $\lim(\mathcal{D})$ is the splitting right step

$$\mathcal{I}\,\frac{\Psi'_i : N'_i \qquad L\colon \Gamma_1, \Gamma_2 \to \Delta_1, \Delta_2}{\Psi'_{i+1} : N'_{i+1} \qquad L'\colon \Gamma_2 \to \Delta_2}$$

2. otherwise, $\Psi'_{i+1} : N'_{i+1}$ is obtained by applying the same rule as in step $f_{\mathcal{D}}(i+1)$ of $\mathcal{D}$ to $\Psi'_i : N'_i$.

Lemma 13 *For any infinite derivation $\mathcal{D}$,* $\lim(\mathcal{D})$ *is well-defined.*

Proof We show by induction on $i \geq 1$ that every premise $L\colon C$ of step i of $\lim(\mathcal{D}) = \Psi'_0 : N'_0 \rhd \Psi'_1 : N'_1 \rhd \dots$ is contained in N'_{i-1}. By definition of $\lim(\mathcal{D})$, step $f_{\mathcal{D}}(i)$ of $\mathcal{D}$ is persistent.

1. If step $f_{\mathcal{D}}(i)$ of $\mathcal{D}$ is an inference or reduction step, then by Definition 18, $L\colon C$ is weakly persistent in $\mathrm{suf}_{\mathcal{D}}(f_{\mathcal{D}}(i) - 1)$.
2. If step $f_{\mathcal{D}}(i)$ of $\mathcal{D}$ is a splitting or backtracking step, then by Lemma 11, $L\colon C$ is weakly persistent in $\mathrm{suf}_{\mathcal{D}}(f_{\mathcal{D}}(i) - 1)$

Assume $L\colon C \notin N_0$. Lemma 10 tells us that $L\colon C$ is weakly persistent in $\mathrm{suf}_{\mathcal{D}}(k)$, where $k < f_{\mathcal{D}}(i)$ is maximal with $L\colon C$ the conclusion of step k of $\mathcal{D}$. Hence by Lemma 12, all premises of step k of $\mathcal{D}$ are weakly persistent, therefore step k is a persistent step. For the base case $i = 1$, this is a contradiction, since $f_{\mathcal{D}}(i)$ is the first persistent step of $\mathcal{D}$, and we can thus conclude that $L\colon C \in N_0 = N'_0$. For $i > 1$, there is $j < i$ such that $f_{\mathcal{D}}(j) = k$ and $L\colon C$ is the conclusion of step j of $\lim(\mathcal{D})$. Since $L\colon C \in N_{f_{\mathcal{D}}(j)-1}$, we can conclude that $L\colon C$ has not been persistently subsumed and hence $L\colon C \in N'_{i-1}$. □

Note that in general, neither $N'_i \subseteq N_{f_{\mathcal{D}}(i)}$ nor $N'_i \supseteq N_{f_{\mathcal{D}}(i)}$, because $N_{f_{\mathcal{D}}(i)}$ may contain clauses that are not (weakly) persistent, and persistent clauses may have been subsumed by non-persistent ones. Therefore we can not simply take the limit to be a subsequence of the initial derivation.

Lemma 14 *For every infinite derivation $\mathcal{D} = \Psi_0 : N_0 \rhd \Psi_1 : N_1 \rhd \dots$, and every $i \geq 1$, if step i of* $\lim(\mathcal{D})$ *is an inference step, then its conclusion is contained in $N_{f_{\mathcal{D}}(i)}$.*

Proof Follows directly from Definition 21. □

For $\lim(\mathcal{D}) = \Psi'_0 : N'_0 \rhd \Psi'_1 : N'_1 \rhd \dots$ we define $N_\infty^{\lim(\mathcal{D})} := \bigcup_i \bigcap_{j \geq i} N'_j$.

Definition 22 (Fairness) A derivation

$$\mathcal{D} = \Psi_0 : N_0 \ \rhd \ \Psi_1 : N_1 \ \rhd \ \ldots$$

is called *fair* if

1. either $\mathcal{D}$ is finite and in the final clause set N_k all resolution and factoring inferences are redundant in N_k, or $\mathcal{D}$ is infinite and every resolution or factoring inference from $N_\infty^{\lim(\mathcal{D})}$ is redundant in some $N'_i \in \lim(\mathcal{D})$; and
2. for every $i \geq 0$ with $L\colon \Box \in N_i$ with $L \neq \emptyset$, there exists $j > i$ such that step j of $\mathcal{D}$ is a backtracking step with premise $L\colon \Box$.

Lemma 15 *For any fair infinite derivation $\mathcal{D}$ and any $i \geq 0$, no $L\colon \Box$ with $L \neq \emptyset$ is weakly persistent in* $\mathrm{suf}_{\mathcal{D}}(i)$.

Proof Assume $L \neq \emptyset$ and consider the backtracking step with $L\colon \Box$ as premise (which must exist by condition 2 of Definition 22). It follows from the definitions of the stack reduction rules that the final rule applied is Enter-right, which deletes the split with the greatest split level in L. □

Our completeness proof will closely follow the one given in [2] (Theorem 4.9), with the exception that we limit ourselves to showing that unsatisfiability of N_0 implies $\emptyset\colon \Box \in N_\infty^{\lim(\mathcal{D})}$. We use the *standard redundancy criterion* that is based on a clause ordering $\succ$ [2] and we write $\mathcal{R}_{\mathcal{F}}^{\succ}(N)$ for the set of redundant clauses with respect to N and $\mathcal{R}_{\mathcal{I}}^{\succ}(N)$ for the set of redundant inferences with respect to N. The standard redundancy criterion satisfies the following four conditions:

(R1) if $N \subseteq N'$ then $\mathcal{R}_{\mathcal{F}}^{\succ}(N) \subseteq \mathcal{R}_{\mathcal{F}}^{\succ}(N')$ and $\mathcal{R}_{\mathcal{I}}^{\succ}(N) \subseteq \mathcal{R}_{\mathcal{I}}^{\succ}(N')$;
(R2) if $N' \subseteq \mathcal{R}_{\mathcal{F}}^{\succ}(N)$ then $\mathcal{R}_{\mathcal{F}}^{\succ}(N) \subseteq \mathcal{R}_{\mathcal{F}}^{\succ}(N \setminus N')$ and $\mathcal{R}_{\mathcal{I}}^{\succ}(N) \subseteq \mathcal{R}_{\mathcal{I}}^{\succ}(N \setminus N')$;
(R3) if N is unsatisfiable, then $N \setminus \mathcal{R}_{\mathcal{F}}^{\succ}(N)$ is also unsatisfiable; and
(R4) a resolution or factoring inference γ is in $\mathcal{R}_{\mathcal{I}}^{\succ}(N)$ whenever its conclusion is in $N \cup \mathcal{R}_{\mathcal{F}}^{\succ}(N)$.

Lemma 16 *Let* $\lim(\mathcal{D}) = \Psi'_0 : N'_0 \ \rhd \ \Psi'_1 : N'_1 \ \rhd \ \ldots$ *be the limit of a derivation $\mathcal{D}$. Then $\mathcal{R}_{\mathcal{F}}^{\succ}(\bigcup_j N'_j) \subseteq \mathcal{R}_{\mathcal{F}}^{\succ}(N_\infty^{\lim(\mathcal{D})})$ and $\mathcal{R}_{\mathcal{I}}^{\succ}(\bigcup_j N'_j) \subseteq \mathcal{R}_{\mathcal{I}}^{\succ}(N_\infty^{\lim(\mathcal{D})})$. Moreover, the set $N_\infty^{\lim(\mathcal{D})}$ is unsatisfiable if N'_0 is unsatisfiable.*

Proof Observe that for any $i \geq 0$, one of the following holds:

(i) $N'_{i+1} = N'_i \cup \{C\}$, where C is the conclusion of a resolution or factoring step from clauses in N'_i; or
(ii) $N'_{i+1} = N'_i \cup \{C\}$, where C is a non-trivial subclause of a clause in N'_i; or
(iii) $N'_{i+1} = N'_i \setminus \{D\}$, where $D \in \mathcal{R}_{\mathcal{F}}^{\succ}(N'_i)$.

Hence by definition of $N_\infty^{\lim(\mathcal{D})}$, any clause in $(\bigcup_j N'_j) \setminus N_\infty^{\lim(\mathcal{D})}$ is in some $\mathcal{R}_{\mathcal{F}}^{\succ}(N'_i)$. Therefore $(\bigcup_j N'_j) \setminus N_\infty^{\lim(\mathcal{D})} \subseteq \bigcup_j \mathcal{R}_{\mathcal{F}}^{\succ}(N'_j)$. Moreover, by condition (R1), $\bigcup_j \mathcal{R}_{\mathcal{F}}^{\succ}(N'_j) \subseteq \mathcal{R}_{\mathcal{F}}^{\succ}(\bigcup_j N'_j)$. As a consequence, we have $(\bigcup_j N'_j) \setminus \mathcal{R}_{\mathcal{F}}^{\succ}(\bigcup_j N'_j) \subseteq N_\infty^{\lim(\mathcal{D})}$. Applying condition (R1) again, we may infer that $\mathcal{R}((\bigcup_j N'_j) \setminus \mathcal{R}_{\mathcal{F}}^{\succ}(\bigcup_j N'_j)) \subseteq$

$\mathcal{R}(N_\infty^{\lim(\mathcal{D})})$ (where $\mathcal{R}$ may be either $\mathcal{R}_{\mathcal{F}}^{\succ}$ or $\mathcal{R}_{\mathcal{I}}^{\succ}$). Using condition (R2), we obtain $\mathcal{R}(\bigcup_j N'_j) \subseteq \mathcal{R}(N_\infty^{\lim(\mathcal{D})})$.

For the second part, assume N'_0 is unsatisfiable. Hence also $\bigcup_j N'_j$ is unsatisfiable, and by condition (R3), $(\bigcup_j N'_j) \setminus \mathcal{R}_{\mathcal{F}}^{\succ}(\bigcup_j N'_j)$ is unsatisfiable. Since $(\bigcup_j N'_j) \setminus \mathcal{R}_{\mathcal{F}}^{\succ}(\bigcup_j N'_j) \subseteq N_\infty^{\lim(\mathcal{D})}$, we can infer that $N_\infty^{\lim(\mathcal{D})}$ is unsatisfiable. □

We say that a set of labelled clauses N is *saturated up to redundancy*, if every resolution or factoring inference γ from $N \setminus \mathcal{R}_{\mathcal{F}}^{\succ}(N)$ is contained in $\mathcal{R}_{\mathcal{I}}^{\succ}(N)$.

Theorem 2 (Completeness) *Let $\mathcal{D} = \Psi_0 : N_0 \rhd \Psi_1 : N_1 \rhd \ldots$ be a fair derivation. If $\emptyset : \Box \notin N_\infty^{\lim(\mathcal{D})}$, then N_0 is satisfiable.*

Proof Since $\mathcal{D}$ is fair, every inference from $N_\infty^{\lim(\mathcal{D})}$ is redundant in some N'_i and therefore, by Lemma 16, also redundant in $N_\infty^{\lim(\mathcal{D})}$. Hence $N_\infty^{\lim(\mathcal{D})}$ is saturated up to redundancy. It follows ([2], Theorem 4.9) that $N_\infty^{\lim(\mathcal{D})}$ is satisfiable if and only if it does not contain a contradiction. By Lemma 15, the only possible contradiction in $N_\infty^{\lim(\mathcal{D})}$ is $\emptyset : \Box$. Thus by Lemma 16, if $\emptyset : \Box \notin N_\infty^{\lim(\mathcal{D})}$, then $N_0 = N'_0$ is satisfiable. □

3 Experiments and a comparison to DPLL

3.1 Experiments

Our enhanced backtracking process has been integrated into the Spass [22] theorem prover on top of Spass version 3.0 for the overall superposition calculus including equality. The basis for the implementation was Spass version 3.0. As we mentioned before, the calculus presented in this article represents only a small set of inference and reduction rules, sufficient for studying the specifics of the labelled splitting calculus. Compared to Spass version 3.0 the data structures used to represent the split stack were modified to allow storage of the dependencies of closed left branches in order to support the refined backtracking rule. Minor modifications to the implementation of reduction rules were made to ensure that reduced clauses are always recorded for later reinsertion. These modifications were necessary since the original branch condensation in Spass is performed only up to the last backtracking level, hence a redundant clause is recorded only if it has been subsumed by a clause with greater split level. Finally, a new backtracking procedure was written, implementing the stack reduction rules.

The implementation was tested on the TPTP problem library, version 3.2.0 [18], which consists of 8984 first-order problems. On 2513 of those problems, Spass (in automatic mode) uses the splitting rule during proof search. For the experiments, an Opteron Linux cluster was used, and Spass was given a time limit of 5 minutes per problem.

Overall, the number of splits performed per problem decreased by about 10% on average when using improved backtracking. In addition, Spass with improved backtracking terminated with a solution for 24 problems (22 proofs, 2 completions) that could not be solved by the version without improved backtracking within the given time limit. The new version loses 4 problems because of the potentially different search space exploration caused by the new backtracking rule. These problems are recovered by an increased time limit.

3.2 Comparing labelled splitting and DPLL

Our paper can be considered as an extension of the abstract DPLL calculus of Nieuwenhuis et al. [14] for first-order logic. Due to the need to create new clauses via inferences and reductions, the labelled splitting calculus is more involved.

The DPLL procedure [6], which lies at the core of most of today's state-of-the-art boolean satisfiability solvers, has received a great deal of attention in the past years. In particular, a lot of research has gone into improving the backtracking process. Thus it is natural to ask how, in the propositional domain, our backtracking scheme compares to that of modern SAT solvers. Let us first clarify how our handling of splitting and backtracking differs from the standard DPLL approach.

In basic DPLL, when exhaustive unit propagation has neither led to a conflict nor yielded a complete truth assignment, a choice is made by assuming some yet undefined literal to be true (typically a literal occurring in some clause). If this choice leads to a conflict, the truth value of that literal is flipped, hence the two branches of the case analysis are A and $\neg A$ for some propositional variable A. This can be simulated directly by the generalized version of the splitting rule, discussed after Definition 1, where we would split the clause $A \rightarrow A$. Independently, the refinement for ground clause splits also introduced after Definition 1 enables a DPLL style splitting if we split just one positive literal away. For example, we split $\Gamma \rightarrow \Delta, A$ into $\rightarrow A$ and $\Gamma \rightarrow \Delta, A \rightarrow$. The reason why this refinement is restricted to ground split parts is that the negation of universally quantified clauses leads to the introduction of new Skolem constants. For example, if we split into $\forall x\, A(x)$ and $\neg\forall x\, A(x)$, then for the second branch we introduce a fresh Skolem constant, say c, resulting in the literal $A(c)$. The literal $\neg A(c)$ is not useful for reductions, because c is fresh and therefore the clause set resulting from the second branch is not smaller in the sense we discussed in the introduction. The introduction of fresh Skolem constants can be avoided by remembering all ground terms instantiating the variables from the left branch in the course of a refutation. For example, if the left split part was $A(x)$ and a refutation of the branch involved substituting x by both a and $f(b)$ in different subtrees, then $\neg A(a) \vee \neg A(f(b))$ could be used instead of $\neg A(c)$ for the fresh Skolem constant c. Although this approach avoids the introduction of new symbols, tracking all instantiations adds overhead and the fact that the resulting lemmata are again in general non-units diminishes their usefulness for reductions. When dealing with purely propositional problems however, this lemma-generation can be used to simulate DPLL-style case analysis: for any clause $\Gamma \rightarrow \Delta, A$ choose A as the first split part – the lemma $\neg A$ is then available in addition on the second branch.

Furthermore, there are two major differences between our backtracking mechanism and the one used in SAT solvers. The first one is that the Branch-condense rule has no equivalent in DPLL-style systems. The reason is that in such systems, where the clause set is essentially fixed and progress is made by extending a partial assignment of truth values to literals, the cost of keeping a decision literal that did not contribute to a conflict in the partial assignment is negligible. On the other hand, when proof search involves generating new clauses via inferences, keeping unused splits on the stack comes at the price of a larger search space, since there are potentially more clauses that can participate in proof search.

Example 3 Consider a propositional problem containing the clauses $\neg B \vee \neg C \vee D$ and $\neg B \vee \neg C \vee \neg D$, and some other clauses where the variable A occurs. Assume our partial assignment is $A^d B^d C^d$, and we get D by unit propagation where we use d as a mark for decision literals. Taking $\neg B \vee \neg C$ as the backjump clause, the Backjump rule of [14] would revert the last decision C^d, yielding assignment $A^d B^d \neg C$. Note that the literal A^d did not contribute to this conflict, but because it is above the first decision literal that did contribute (B^d), it is not removed. On the other hand, in our framework, we would have derived an empty clause whose label does not refer to the first split (of which A^d is the first component). Hence Branch-condense could be applied, removing the first split and all clauses depending on it.

Secondly, modern SAT solvers usually rely on clause learning, meaning that conflicts are analyzed to yield a new clause (called the backjump or conflict clause) which is added to the clause set. The conflict clause captures the underlying reason for the conflict, so as to prevent the solver from following paths that would reproduce it. The conflict clause is also used during backtracking (see Backjump rule of [14]), since it contains information on what decision levels did (not) contribute to the conflict. If one takes the conflict clause to consist of the negations of all decision literals that contributed to the conflict, then its role during backtracking is comparable to that of the empty clause's label and the leaf markers in our framework. More precisely, when we derive an empty clause with label L, then the split components described by L correspond to the "decisions" that caused the conflict. Leaf markers can then be thought of as conflict clauses learned after closing the corresponding left branches, and the rule Right-collapse is equivalent to performing a resolution step between two conflict clauses. Several Right-collapse steps may be performed in a row, in the same way as propositional conflict analysis may take several backward steps through the conflict graph [13]. The crucial difference between learned clauses and clause labels is that a learned clause may be reused arbitrarily often, while the empty clauses' labels and leaf markers are consumed during backtracking.

We compared Spass implementing our new backtracking rule (Definition 12) with MiniSat v1.14 [8] on SAT problems, by manipulating both systems such that they essentially use the same propositional variable order for branching. Out of the current SATLIB (http://www.satlib.org) library we selected about 200 problems that Spasscould solve in a 5 min time limit. On more than 95% of all problems Minisat needs less splits than Spass. Out of these problems, Spass performs three times more splits than MiniSat, on the average. This result suggests that conflict-driven clause learning can offer advantages over the Spass split backtracking mechanism. So there is potential for exploring this mechanism also for the first-order case. However, again, this requires at least a detailed analysis of the closed branches and a careful decision when the generated lemmas help in closing branches, because split clauses may have been used in several instantiations.

Although our backtracking rule does not include learning of conflict clauses, there are cases where it is superior to the conflict driven clause learning of modern SAT solvers, i.e., a proof requires less splits. This is documented by the 5% of examples where Spass needed less splits than MiniSat and such examples can also be explicitly constructed.

The following example illustrates a clause set, where the labelled splitting style of backtracking is in favor of the DPLL style of backtracking.

Example 4

$$\begin{array}{lll}\{\ \neg A_1\vee\neg A_2\vee\neg A_3\vee A_5, & \neg A_1\vee\neg A_2\vee\neg A_3\vee\neg A_5, & \neg A_1\vee\neg A_4\vee A_6,\\ \ \ \neg A_1\vee\neg A_4\vee\neg A_6, & \neg A_1\vee\neg A_2\vee A_4\vee A_7, & \neg A_1\vee\neg A_2\vee A_4\vee\neg A_7,\\ \ \ A_1\vee C_1, & A_2\vee C_2, & A_3\vee C_3,\\ \ \ A_4\vee C_4, & \ldots\} & \end{array}$$

The C_i stand for further disjuncts containing positive literals. To keep the example small, it contains potential for simplification, e.g., the first two clauses can be reduced to $\neg A_1\vee\neg A_2\vee\neg A_3$ by a matching replacement resolution reduction (Definition 6). Marking DPLL decision literals again with a d, the partial truth assignment $A_1^d\,A_2^d\,A_3^d$ (corresponding to splittings of the clauses $A_i\vee C_i$, $1\leq i\leq 3$) is extended to $A_1^d\,A_2^d\,A_3^d\,A_5$ by unit propagation, leading to a conflict. In terms of the labelled resolution calculus, this corresponds to an empty clause with label $\{1,3,5\}$ (remember that left branches have odd levels) with first split clause $A_1\vee C_1$, second split clause $A_2\vee C_2$, and third split clause $A_3\vee C_3$. Both DPLL and labelled splitting then enter the right branch of the last split. In DPLL terms, we get the assignment $A_1^d\,A_2^d\,\neg A_3$, which we extend to $A_1^d\,A_2^d\,\neg A_3\,A_4^d$ via a further decision. This corresponds to the new split level 7 by splitting $A_4\vee C_4$ via the labelled splitting calculus. Clause $\neg A_1\vee\neg A_4\vee A_6$ propagates A_6, after which clause $\neg A_1\vee\neg A_4\vee\neg A_6$ becomes false. Resolving these two yields the conflict clause $\neg A_1\vee\neg A_4$, and DPLL backjumps to decision level 1, i.e. to assignment $A_1^d\,\neg A_4$. If this branch cannot be closed without repeating the decision on A_2 that was just undone, this overall search space needs to be explored again. On the other hand, in the labelled calculus, the conflict clause corresponds to an empty clause with label $\{1,7\}$, where we have split $A_4\vee C_4$. So Enter-right is the only applicable stack reduction rule. The right branch of the corresponding split can be closed by the clauses $\neg A_1\vee\neg A_2\vee A_4\vee A_7$ and $\neg A_1\vee\neg A_2\vee A_4\vee\neg A_7$ without further splitting, since the split of clause $A_2\vee C_2$ is still available.

4 Conclusion

We have extended the abstract DPLL calculus by Nieuwenhuis et al. [14] to the full first-order case, called labelled splitting. The calculus is sound and complete. We established a completeness result by introducing a new notion of fairness taking into account infinite paths in the split tree.

The calculus is implemented for the full superposition calculus [20] in Spass. It shows a 10% average gain in the number of splits on all TPTP problems where splitting is involved and Spass could decide 24 more problems compared to the previous version.

The fairness notion of Definition 22 does not directly provide an effective strategy for a fair inference selection. In Spass we mainly use two different strategies. For the propositional case we employ exhaustive splitting, because splitting combined with the reduction matching replacement resolution [20] constitutes already a complete calculus and the number of splits is a priori finitely bound by the number of different propositional variables. For the first-order case, clauses are selected for inferences with respect to their “weight” composed out of the number of contained symbols and their depth in the derivation. If such a clause can be split and the first split part has a “sufficient” reduction potential for other clauses, splitting is preferred over other inferences.

A comparison of the labelled splitting backtracking mechanism with DPLL style backtracking based on conflict-driven clause learning reveals room for further improvement. However, this is not a straightforward effort, because the negation of a first-order clause is an existentially quantified conjunction of literals that via Skolemization introduces new constants to the proof search. It is well known that the addition of new constants causes an increased complexity of the unsatisfiability problem and if potentially done infinitely often, can even cause completeness issues. So it seems to us that in the case of a conflict, an analysis of the proof and the used first-order terms in the proof is the most promising approach to enhance the presented labelled splitting backtracking mechanism with conflict-driven clause learning. This will be subject of future research.

References

1. Baader, F., Nipkow, T.: Term Rewriting and All That. Cambridge University Press, Cambridge (1998)
2. Bachmair, L., Ganzinger, H.: Resolution theorem proving. In: Handbook of Automated Reasoning, pp. 19–99 (2001)
3. Bachmair, L., Ganzinger, H., Waldmann, U.: Superposition with simplification as a decision procedure for the monadic class with equality. In: Gottlob, G., Leitsch, A., Mundici, D. (eds.) Computational Logic and Proof Theory, Third Kurt Gödel Colloquium. LNCS, vol. 713, pp. 83–96. Springer, New York (1993)
4. Basin, D., D’Agostino, M., Gabbay, D.M., Matthews, S., Viganó, L., (eds.) Labelled Deduction. Kluwer, Dordrecht (2000)
5. Baumgartner, P., Tinelli, C.: The model evolution calculus as a first-order DPLL method. Artif. Intell. **172**(4–5), 591–632 (2008)
6. Davis, M., Putnam, H.: A computing procedure for quantification theory. J. ACM **7**(3), 201–215 (1960)
7. de Nivelle, H.: Splitting through new proposition symbols. In: Logic for Programming, Artificial Intelligence, and Reasoning, 8th International Conference, LPAR 2001. LNAI, vol. 2250, pp. 172–185. Springer, New York (2001)
8. Eén, N., Sörensson, N.: An extensible SAT solver. Theory and Applications of Satisfiability Testing, pp. 502–518 (2004)
9. Fermüller, C.G., Leitsch, A., Hustadt, U., Tamet, T.: Resolution decision procedures. In: Robinson, A., Voronkov, A. (eds.) Handbook of Automated Reasoning, vol. II, chapter 25, pp. 1791–1849. Elsevier, Amsterdam (2001)
10. Fietzke, A., Weidenbach, C.: Labelled splitting. In: 4th International Joint Conference on Automated Reasoning (IJCAR). LNAI, vol. 5195, pp. 459–474. Springer, New York (2008)
11. Hillenbrand, T., Weidenbach, C.: Superposition for finite domains. Research report MPI-I-2007-RG1-002, Max-Planck Institute for Informatics, Saarbrücken (2007)
12. Lev-Ami, T., Weidenbach, C., Reps, T., Sagiv, M.: Labelled clauses. In: 21st International Conference on Automated Deduction (CADE-21). Lecture Notes in Computer Science, vol. 4603, pp. 311–327. Springer, New York (2007)
13. Nieuwenhuis, R., Oliveras, A.: Decision procedures for SAT, SAT Modulo Theories and Beyond. The Barcelogic Tools. (Invited Paper). In: Sutcliffe, G., Voronkov, A. (eds.) 12th International Conference on Logic for Programming, Artificial Intelligence and Reasoning, LPAR’05. Lecture Notes in Computer Science, vol. 3835, pp. 23–46. Springer, New York (2005)
14. Nieuwenhuis, R., Oliveras, A., Tinelli, C.: Solving SAT and SAT modulo theories: from an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL. J. ACM **53**(6), 937–977 (2006)

15. Nieuwenhuis, R., Rubio, A.: Paramodulation-based theorem proving. In: Robinson, A., Voronkov, A. (eds.) Handbook of Automated Reasoning, vol. I, chapter 7, pp. 371–443. Elsevier, Amsterdam (2001)
16. Nonnengart, A. Weidenbach, C.: Computing small clause normal forms. In: Robinson, A., Voronkov, A. (eds.) Handbook of Automated Reasoning, vol. 1, chapter 6, pp. 335–367. Elsevier, Amsterdam (2001)
17. Riazanov, A., Voronkov, A.: Splitting without backtracking. In: IJCAI, pp. 611–617 (2001)
18. Sutcliffe, G., Suttner, C.: The TPTP problem library: CNF release v1.2.1. J Autom Reason **21**(2), 177–203 (1998)
19. Tseitin, G.: On the complexity of derivations in propositional calculus. In: Siekmann, J., Wrightson, G. (eds.) Automation of Reasoning: Classical Papers on Computational Logic, vol. 2, pp. 466–483. Springer (1983). First published in: Studies in Constructive Mathematics and Mathematical Logic, (Slisenko, A.O., ed.) (1968)
20. Weidenbach, C.: Combining superposition, sorts and splitting. In: Robinson, A., Voronkov, A. (eds.), Handbook of Automated Reasoning, vol. 2, chapter 27, pp. 1965–2012. Elsevier, Amsterdam (2001)
21. Weidenbach, C., Gaede, B., Rock, G.: Spass and flotter, version 0.42. In: McRobbie, M., Slaney, J. (eds.) 13th International Conference on Automated Deduction, CADE-13. LNAI, vol. 1104, pp. 141–145. Springer, New York (1996)
22. Weidenbach, C., Schmidt, R., Hillenbrand, T., Rusev, R., Topic, D.: System description: SPASS version 3.0. In: Pfenning, F. (ed.) CADE-21: 21st International Conference on Automated Deduction. LNAI, vol. 4603, pp. 514–520. Springer, New York (2007)