

Saarland University
Faculty of Natural Sciences and Technology I
Department of Computer Science
Master's Program in Computer Science

Master Thesis

Labelled Splitting

Submitted by
Arnaud Fietzke
on October 26, 2007

Supervisor:
PD Dr. Christoph Weidenbach

Reviewers:
Prof. Dr. Bernd Finkbeiner
PD Dr. Christoph Weidenbach

Statement

Hereby I confirm that this thesis is my own work and that I have documented all sources used.

Saarbrücken, October 26, 2007

Arnaud Fietzke

Declaration of Consent

Herewith I agree that my thesis will be made available through the library of the Computer Science Department.

Saarbrücken, October 26, 2007

Arnaud Fietzke

Abstract

Saturation-based theorem provers are typically based on a calculus consisting of inference and reduction rules that operate on sets of clauses. While inference rules produce new clauses, reduction rules allow the removal or simplification of redundant clauses and are an essential ingredient for efficient implementations.

The power of reduction rules can be further amplified by the use of the splitting rule, which is based on explicit case analysis on variable-disjoint components of a clause.

In this thesis, I give a formalization of splitting and backtracking for first-order logic using a labelling scheme that annotates clauses and clause sets with additional information, and I present soundness and completeness results for the corresponding calculus.

The backtracking process as formalized here generalizes optimizations that are currently being used, and I present the results of integrating the improved backtracking into SPASS.

Acknowledgements

I would like to thank my supervisor PD Dr. Christoph Weidenbach for providing me with an inspiring thesis subject, and for guiding my research through his constructive criticism and insightful remarks.

I also want to thank Prof. Dr. Bernd Finkbeiner who kindly agreed to review this thesis.

I am grateful to my family for supporting and encouraging me throughout my whole studies.

Finally, I wish to thank my girlfriend for her patience and continuous emotional support without which this work would not have been possible.

Contents

| | |
|---|------------|
| Abstract | i |
| Acknowledgements | iii |
| 1 Introduction | 1 |
| 1.1 Related Work | 5 |
| 2 Preliminaries | 7 |
| 3 Labelled Framework | 8 |
| 3.1 Basic Definitions | 8 |
| 3.2 Stack Reduction Rules | 9 |
| 3.2.1 Deleting Splits | 9 |
| 3.2.2 Reduction Rules for Labelled Clause Sets | 11 |
| 3.3 Labelled Calculus | 12 |
| 3.3.1 Rule Semantics | 12 |
| 3.3.2 Inference and Reduction Rules | 12 |
| 4 Correctness | 15 |
| 4.1 Derivations in the Labelled Calculus | 15 |
| 4.1.1 Proving Derivation Invariants | 15 |
| 4.2 Satisfiability of Labelled Clause Sets | 16 |
| 4.3 Properties of Derivations | 17 |
| 4.3.1 Label-validity | 17 |
| 4.3.2 Path-validity | 19 |
| 4.3.3 Some Derivation Invariants | 21 |
| 4.3.4 Preservation of Satisfiability in Derivations | 23 |
| 4.4 Soundness and Completeness | 29 |
| 5 Experimental Results | 30 |
| 5.1 Implementation | 30 |
| 5.2 Results on TPTP Problems | 30 |
| 6 Future Work | 31 |
| 6.1 Splitting with Lemmata | 31 |
| 6.2 Clause Labels Only | 32 |

List of Lemmata, Propositions and Theorems

| | | |
|-------------|--|----|
| Lemma 4.1 | Stack reduction terminates | 16 |
| Lemma 4.2 | Existence of leaf markers | 17 |
| Lemma 4.3 | Minimality of parent labels | 17 |
| Lemma 4.4 | Split deletion and label-validity | 18 |
| Lemma 4.5 | Stack reductions maintain label-validity | 18 |
| Prop. 4.1 | Label-validity in derivations | 19 |
| Lemma 4.6 | Stack reductions maintain path-validity | 19 |
| Prop. 4.2 | Path-validity in derivations | 20 |
| Cor. 4.1 | Restriction expansion | 21 |
| Prop. 4.3 | Existence of a subsumption deletion step | 21 |
| Prop. 4.4 | Existence of subsuming clauses | 21 |
| Cor. 4.2 | Active clause sets and deleted clauses | 22 |
| Lemma 4.7 | Split deletion and inactive clause sets | 22 |
| Lemma 4.8 | BACKJUMP maintains satisfiability | 24 |
| Lemma 4.9 | BRANCH-CONDENSE maintains satisfiability | 24 |
| Lemma 4.10 | RIGHT-COLLAPSE maintains satisfiability | 27 |
| Lemma 4.11 | ENTER-RIGHT maintains satisfiability | 27 |
| Prop. 4.5 | Calculus rules maintain satisfiability | 28 |
| Theorem 4.1 | Soundness | 29 |
| Theorem 4.2 | Completeness for finite splitting | 29 |

1 Introduction

Saturation-based theorem provers are typically based on a calculus consisting of inference and reduction rules that operate on sets of clauses. While inference rules produce new clauses, reduction rules allow the removal or simplification of redundant clauses. Calculi based on first-order resolution are refutationally complete in the sense that an unsatisfiability proof can be found in finite time for any unsatisfiable set of clauses [2, 10]. For automatic theorem proving, the reduction rules play a major role, since they allow to simplify the clause set as much as possible, before again applying inference rules to derive new clauses. Together with ordering restrictions, strong reduction rules make automatic saturation-based theorem proving often feasible in practice.

An entirely different approach to mechanized theorem proving is represented by tableau-based methods (see [7] for an overview). Instead of saturating an initial clause set, these methods successively decompose a formula according to the semantics of the operators (e.g., conjunction, disjunction and quantifiers, for first-order logic). If the formulae are restricted to clauses, then a central decomposition rule of tableau calculi is the β -rule, which turns the task of refuting a disjunction into the tasks of refuting each disjunct.

Combining the β -rule with a resolution-based approach gives rise to the *splitting* rule. In essence, the splitting rule reduces the task of refuting a clause set $S \cup \{C_1 \vee C_2\}$ to refuting the two sets $S \cup \{C_1\}$ and $S \cup \{C_2\}$, whenever C_1 and C_2 have no variables in common. Clearly then, $S \cup \{C_1 \vee C_2\}$ is satisfiable if and only if either $S \cup \{C_1\}$ or $S \cup \{C_2\}$ (or both) are satisfiable.

The rationale behind the use of the splitting rule is that smaller clauses are more likely to be usable for reducing other clauses, thus making it potentially easier to refute the sets $S \cup \{C_1\}$ and $S \cup \{C_2\}$ than refuting $S \cup \{C_1 \vee C_2\}$ directly. On the propositional level, this case analysis together with unit propagation is the basis of the widely-used decision procedure DPLL [5]. Furthermore, the fact that splitting reduces the size of clauses is key to showing decidability results for certain subclasses of first-order logic [3, 8]. In particular, it is beneficial to split into clauses that have strictly fewer positive literals than the parent clause, because deciding satisfiability of Horn clause sets is often easier than deciding satisfiability of sets of arbitrary clauses. For example, for a set of propositional Horn clauses, satisfiability can be decided in linear time [6]. In this basic form, the splitting rule can be written as

$$\mathcal{S} \frac{\Gamma_1, \Gamma_2 \rightarrow \Delta_1, \Delta_2}{\Gamma_1 \rightarrow \Delta_1 \mid \Gamma_2 \rightarrow \Delta_2}$$

where clauses are written as implications and the Γ_i and Δ_i contain the positively

and negatively occurring atoms, respectively. The rule is applicable if $\Gamma_1 \rightarrow \Delta_1$ and $\Gamma_2 \rightarrow \Delta_2$ are variable-disjoint and both Δ_i are non-empty.

Applying the rule recursively builds up a binary tree that can be explored in a depth-first fashion, for example. After a refutation has been found on one of the branches, the next open branch is entered. This operation is called *backtracking*.

Let us now look at an example illustrating the important aspects in the interplay of inferences, reductions, splitting and backtracking. For the sake of example, we will prefer to do a lot of splits before starting to look for contradictions. An actual prover would of course try to simplify the clause sets as early as possible, e.g., propagating the unit clauses obtained by splitting.

Example 1.1. Consider the clause set

$$\begin{aligned}
 S = \{ & \rightarrow P_1(x), Q_1(y), & (C_1) \\
 & \rightarrow P_2(a), Q_2(b), & (C_2) \\
 & P_2(x) \rightarrow Q_3(x), R_3(x), & (C_3) \\
 P_2(a), Q_3(a) \rightarrow & , & (C_4) \\
 & \rightarrow P_4(a), Q_4(a), & (C_5) \\
 & \rightarrow P_4(x), Q_4(b), & (C_6) \\
 P_1(a), P_4(b) \rightarrow & , & (C_7) \\
 P_1(b), Q_4(b) \rightarrow & \} & (C_8)
 \end{aligned}$$

We first apply splitting to C_1 to obtain two new clause sets, $S_1 = S \setminus \{C_1\} \cup \{\rightarrow P_1(x)\}$ and $S_2 = S \setminus \{C_1\} \cup \{\rightarrow Q_1(y)\}$. Applying splitting again to $C_2 \in S_1$, yields $S_3 = S_1 \setminus \{C_2\} \cup \{\rightarrow P_2(a)\}$ and $S_4 = S_1 \setminus \{C_2\} \cup \{\rightarrow Q_2(b)\}$, and next set in the depth-first traversal is S_3 . Figure 1.1(a) shows the split tree, with the already explored branches shown in bold. We can perform a resolution step between C_3 and the new clause $\rightarrow P_2(a)$, yielding a new clause $\rightarrow Q_3(a), R_3(a)$, which we again split, obtaining clause sets $S_5 = S_4 \cup \{\rightarrow Q_3(a)\}$ and $S_6 = S_4 \cup \{\rightarrow R_3(a)\}$. The corresponding tree is shown in Figure 1.1(b). The current clause set S_5 now contains the clauses $\rightarrow P_2(a)$, $\rightarrow Q_3(a)$ and $C_4 = P_2(a), Q_3(a) \rightarrow$, hence the empty clause can be derived by unit propagation and we backtrack. We now have to refute S_6 . Splitting the clauses C_5 and C_6 produces new clause sets S_7, S_8, S_9 and S_{10} . Figure 1.1(c) shows the corresponding tree, the refuted branch corresponding to S_5 is dashed. The clause $\rightarrow P_4(x)$ makes the clause $\rightarrow P_4(a)$ redundant, so we might want to remove $\rightarrow P_4(a)$ from the current clause set S_9 . We now discover that S_9 is unsatisfiable by deriving the empty clause using clause C_7 , so we again backtrack and S_{10} becomes the current clause set.

A naive way to implement a depth-first traversal such as the one performed in Example 1.1 would be to keep all unexplored clause sets on a stack, adding a new set with each splitting step and refuting each set until the stack is empty. This would be highly inefficient, since clauses would be duplicated and inferences or reductions

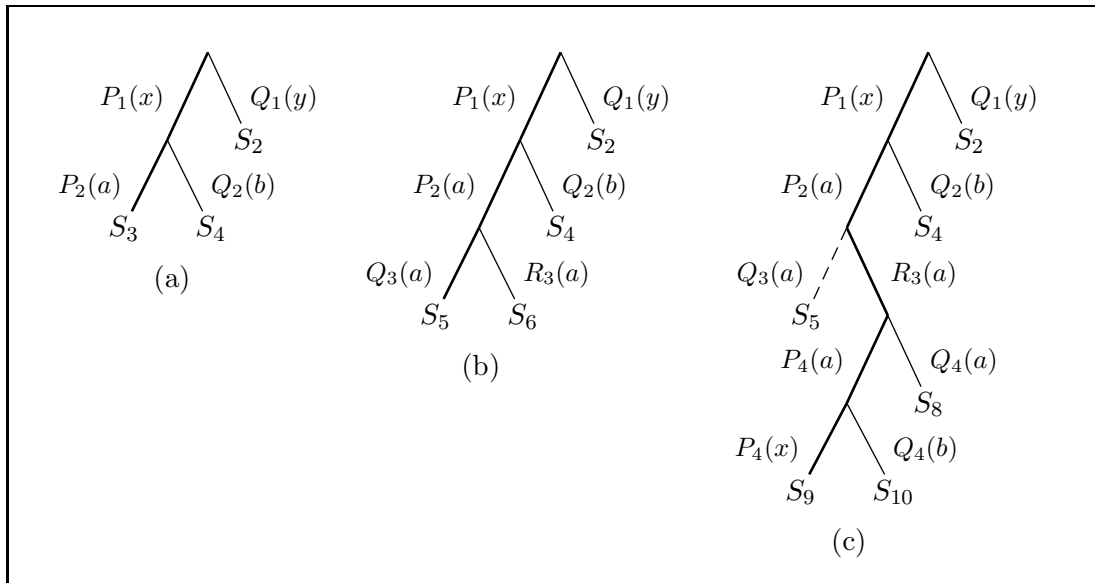


Figure 1.1: Three stages in the exploration of the split tree from Example 1.1.

involving clauses that are duplicated across multiple clause sets would also have to be performed multiple times. In practice, only a single clause set is maintained at any time, and a stack holds the information required to reconstruct the various clause sets when necessary. This reduces memory requirements and allows to share the results of inferences within subtrees. But it also requires significant bookkeeping to ensure that the clause set stays consistent across reductions and backtracking steps. Consider again Example 1.1: we removed the redundant clause $\rightarrow P_4(a)$ from the set S_9 , but when we backtrack to set S_{10} , the clause $\rightarrow P_4(a)$ is no longer redundant, since $\rightarrow P_4(x) \notin S_{10}$. Thus in practice, we must not delete the redundant clause $\rightarrow P_4(a)$, but we have to record it and reinsert when we backtrack to a point where the reducing clause is no longer valid.

In order to determine whether a clause is valid with respect to the current split tree, each clause is annotated with its *split history*, i.e., the splits it depends on, represented as a set of integers. In Example 1.1, clause $\rightarrow Q_3(a)$ in S_5 would depend on splits two and three: the parent clause $\rightarrow Q_3(a), R_3(a)$ was obtained from C_3 and clause $\rightarrow P_2(a)$, which was itself produced by the second split; clause $\rightarrow Q_3(a)$ was then produced by the third split. The greatest split a clause depends on is called the clause's *split level*. When the empty clause is derived, its split level indicates where backtracking should start to consider open branches and all deeper branches can be discarded. For example, if we derive the empty clause at split level zero, then we can immediately stop and don't have to consider any further possibly open branches. When a clause C is reduced by a clause D with a greater split level, C is removed from the current clause set and stored at D 's split level on the split stack and reinserted

if backtracking considers that level. A further optimization of backtracking that is implemented is SPASS [14] is *branch condensation*: If there are some split levels smaller than the empty clause's split level, and those split levels are greater than the last backtracking level, the corresponding branches can be discarded too. In Figure 1.1(c), the last backtracking level is level three, since this is where we last entered a right branch. In the refutation of S_9 , the empty clause was derived from clauses $\rightarrow P_1(x)$ and $\rightarrow P_4(x)$ and C_7 and thus depends only on splits one and five. Hence we would undo the fourth split, before entering the right branch of split five. The corresponding split tree is depicted in Figure 1.2. Note that removing a split "in the middle" of the tree also means we have to make sure that all clauses that depended on that split are removed. In Figure 1.2, the clause sets S'_9 and S'_{10} are the results of removing the clause $\rightarrow P_4(a)$ from S_9 and S_{10} , respectively.

Let us look at the example again, continuing with the situation from Figure 1.2, to see what happens next. The clause set S'_{10} , which we now want to refute, contains the clauses $\rightarrow P_1(b)$ (from split one) and $\rightarrow Q_4(b)$ (from the fifth split, which has been shifted to level four by branch condensation). Since S'_{10} also contains clause $C_8 = P_1(b), Q_4(b) \rightarrow$ from the initial set, it is unsatisfiable. Remember that in the refutation of the last left branch (now S'_9), we also used only splits one and five. In other words, we have shown that the initial clause set S is unsatisfiable under the assumptions $P_1(x)$, $P_4(x)$, and under the assumptions $P_1(x)$, $Q_4(b)$. But since the clause $C_6 = \rightarrow P_4(x), Q_4(b)$ was in S , at least one of its conjuncts must be true in any model of S . We can therefore conclude that S is unsatisfiable under the assumption $\rightarrow P_1(x)$ alone. For backtracking, this means that we don't need to enter the right branch at level two, since $\rightarrow P_1(x) \in S_4$, so we can directly jump to the right branch at level one. This pruning of branches above the last backtracking level, which we call *generalized branch condensation*, gives rise to a new and enhanced backtracking mechanism, which will be described in detail in this thesis. For generalized branch condensation, additional mechanisms are required to ensure that the proof procedure remains sound. Consider again Figure 1.2: had we used clause $Q_2(b)$ of split three in refuting the sets S'_9 and S'_{10} , the simple reasoning we did before would not work any more, because the clause $\rightarrow P_2(a)$ of split two may have been used in refuting the set S_5 . This means that in order to condense branches above the last backtracking level, we need to remember which dependencies were involved in refuting the left subtree at that level. That information, together with the dependencies of the right subtree, allows us to decide which splits we can remove.

In this thesis, I present a first-order calculus that captures depth-first splitting and backtracking with generalized branch condensation for the single-clause-set approach we have discussed. In particular, the backtracking process, which previously took place outside of the calculus, now becomes an integral part of it, and a formal proof of soundness and completeness properties is presented in this thesis for the first time. The formalization relies on the use of labels, for both clauses and clause sets. Simply put, clauses will be labelled with the split levels they depend on, while the single clause set will carry the split stack as its label. The calculus discussed in this thesis

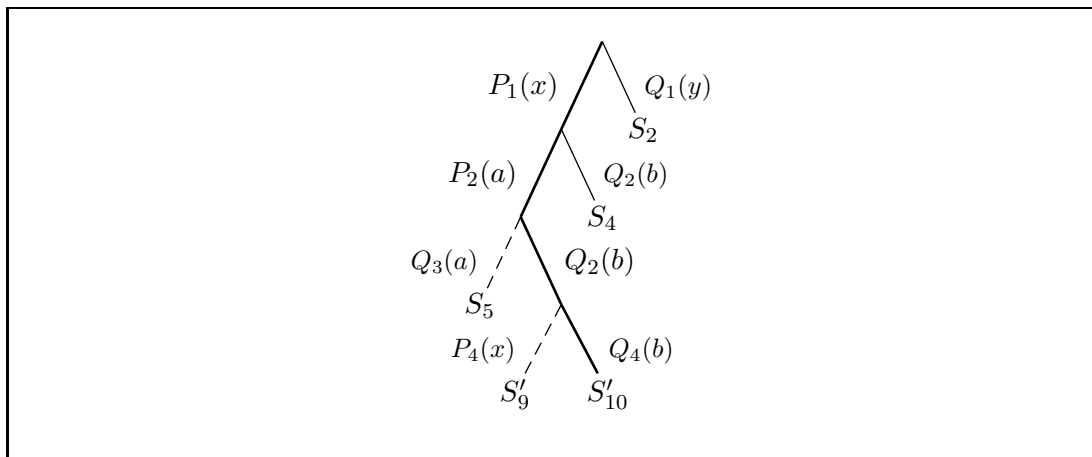


Figure 1.2: The split tree obtained from Figure 1.1(c) after backtracking with branch condensation.

deals with first-order logic without equality. The reason is that we want to focus our attention on the issues related to splitting, and we therefore keep the other parts of the calculus as simple as possible. Rules for reasoning about equality can be integrated to this calculus in a straightforward way, since they don't interfere with the splitting-related aspects (see [13]).

1.1 Related Work

Tableau-based methods have been around for a long time, and exist in many flavors – [7] gives an overview of the most important concepts. For propositional satisfiability, most state-of-the-art solvers are based on the DPLL procedure [5] which relies on the splitting principle. In the field of first-order theorem proving, the splitting rule has been successfully integrated into SPASS [13, 14]. An alternative approach to case analysis in saturation-based theorem proving, relying on the introduction of special propositional symbols instead of a split stack, is presented in [11]. The main difference to our framework is that when simulating splitting with new propositional symbols, the generated clauses can not be directly used for reductions: for example, splitting the clause $\rightarrow P(x), Q(y)$ in this way produces the clauses $\rightarrow P(x), p$ and $p \rightarrow Q(x)$, where p is a new propositional symbol. On the other hand, our approach is motivated by the increased reductive potential of smaller clauses obtained by splitting and therefore we cannot trade off the complex bookkeeping involved in splitting into actual clause components. 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 [9], which provided a starting point for the work presented here. We will discuss the approach from [9] in more detail in

Chapter 6.

The remainder of this thesis is organized as follows: Chapter 2 provides a brief review of the basic notions of first-order logic which we will rely on in later chapters. In Chapter 3, we introduce the concepts of labelled clauses and labelled clause sets, and we define the calculus that operates on them. In Chapter 4, we establish some properties of derivations in the labelled calculus, and work out soundness and completeness results. Chapter 5 presents the results of the integration of the modified backtracking procedure into SPASS. Finally, Chapter 6 points out some further theoretical and practical aspects of labelled splitting.

2 Preliminaries

We employ the usual notions and notations of first-order logic in a way consistent with [13]. A first-order language is constructed over a signature $\Sigma = (\mathcal{F}, \mathcal{R})$, where \mathcal{F} and \mathcal{R} are non-empty, disjoint, in general infinite sets of function and predicate symbols, respectively. Every function or predicate symbol has some fixed arity. We further assume an infinite set \mathcal{X} of variable symbols, disjoint from the symbols in Σ . Then the set of all *terms* $\mathcal{T}(\mathcal{F}, \mathcal{X})$ is recursively defined by: (i) every function symbol $c \in \mathcal{F}$ with arity zero (a *constant*) is a term, (ii) every variable $x \in \mathcal{X}$ is a term and (iii) whenever t_1, \dots, t_n are terms and $f \in \mathcal{F}$ is a function symbol with arity n , then $f(t_1, \dots, t_n)$ is a term. A term not containing a variable is a *ground term*. If t_1, \dots, t_n are terms and $R \in \mathcal{R}$ is a predicate symbol with arity n , then $R(t_1, \dots, t_n)$ is an *atom*. An atom or the negation of an atom is called a *literal*. Disjunctions of literals are clauses where all variables are implicitly universally quantified. Clauses are often denoted by their respective multisets of literals. A clause consisting of exactly one literal is called a *unit*.

The set of *free variables* of an atom (term) ϕ , denoted by $vars(\phi)$ is defined as follows: $vars(P(t_1, \dots, t_n)) = \bigcup_i vars(t_i)$ and $vars(f(t_1, \dots, t_n)) = \bigcup_i vars(t_i)$, $vars(x) = \{x\}$. The function naturally extends to literals, clauses and (multi)sets of terms (literals, clauses).

A *substitution* σ is a mapping from the set of variables to the set of terms such that $x\sigma \neq x$ for only finitely many $x \in \mathcal{X}$. Given two terms (atoms) s, t , a substitution σ is called a *unifier* for s and t if $s\sigma = t\sigma$. It is called a *most general unifier* (or mgu) if for any other unifier τ of s, t there exists a substitution λ with $\sigma\lambda = \tau$. A substitution is called a *matcher* from s to t if $s\sigma = t$. The notion of a mgu is extended to atoms and literals in the obvious way.

As an alternative to the already mentioned notation of clauses, we also write clauses in the form $\Gamma \rightarrow \Delta$ where Γ and Δ are multisets containing arbitrary atoms. Logically, the atoms in Γ denote negative literals while the atoms in Δ denote positive literals in the clause. The empty clause \square denotes \perp (falsity). We often abbreviate multiset union with sequencing, e.g., we write $\Gamma \rightarrow \Delta, R(t_1, \dots, t_n)$ for $\Gamma \rightarrow \Delta \cup \{R(t_1, \dots, t_n)\}$.

A clause $\Gamma_1 \rightarrow \Delta_1$ *subsumes* a clause $\Gamma_2 \rightarrow \Delta_2$ if $\Gamma_1\sigma \subseteq \Gamma_2$ and $\Delta_1\sigma \subseteq \Delta_2$ for some matcher σ . The relation "is subsumed by" between clauses is a quasi-ordering on clauses.

3 Labelled Framework

In this chapter, we introduce the concepts of labelled clauses and labelled clause sets, where clauses are labelled with their split history and clause sets are labelled with the split stack. We then introduce rules which carry out backtracking by transforming labelled clause sets into a normal form. Finally, we present a calculus with inference, reduction, splitting and backtracking rules operating on labelled clause sets.

3.1 Basic Definitions

A clause $\Gamma \rightarrow \Delta$ is labelled with a finite set $L \subseteq \mathbb{N}$ of *split levels*:

$$L: \Gamma \rightarrow \Delta.$$

We define the *split level of* $L: \Gamma \rightarrow \Delta$ to be the greatest element in L . We say that a clause $L: \Gamma \rightarrow \Delta$ *depends* on l if $l \in L$. We extend the usual notions about clause sets to sets of labelled clauses in the natural way: 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 .

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 of the form

$$\Psi ::= \langle \psi_n, \dots, \psi_1 \rangle$$

for $n \geq 0$. We call n the *length* of Ψ . The ψ_i are tuples

$$\psi_i = (l_i, B_i, D_i, b_i, \varphi_i)$$

called *splits*, where

- $l_i \in \mathbb{N}$ is the splits' level
- B_i is the set of *blocked* clauses
- D_i is the set of *deleted* clauses
- $b_i ::= \textit{left} \mid \textit{right}$
- $\varphi_i ::= \emptyset \mid \{L\}$, with $L \subseteq \mathbb{N}$, the *leaf marker*.

We denote by $\max_r(\Psi) := \max\{1 \leq i \leq n \mid b_i = \text{right}\}$ the last right split in Ψ .

To improve readability, we will use the notation $\psi[x := v]$ where x is one of l, B, D, b, 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 any given split stack Ψ , we define the set $\text{levels}(\Psi)$ to be the set of split levels occurring in Ψ , i.e.

$$\text{levels}(\Psi) := \{l_1, \dots, l_n\}.$$

For any given clause set N and set of split levels $K \subseteq \mathbb{N}$ we define

$$N|_K := \{L: \Gamma \rightarrow \Delta \in N \mid L \subseteq K\}$$

and

$$N|\overline{K} := \{L: \Gamma \rightarrow \Delta \in N \mid L \cap K = \emptyset\}.$$

We will later show that each split is uniquely identified by its split level, and that each split level occurring in a clause label or a blocked or deleted set corresponds to an existing split. In the subsequent definitions, this will justify the use of the notation l_k to refer to the split level of the k th split in the split stack under consideration.

3.2 Stack Reduction Rules

In this section, we define rules that formalize backtracking with generalized branch condensation. In particular, we focus our attention on the deletion of splits from the split stack to ensure that all the bookkeeping is done correctly.

3.2.1 Deleting Splits

When removing a split k from the stack, we have to take care to undo all reductions that involved a clause depending on that split.

In particular, if a clause C_1 depending on split k was used to reduce some other clause C_2 , then C_2 must be reinserted into the current clause set. The reason is that C_1 will be removed from the current clause set, and C_2 may then no longer be redundant. If C_2 was reduced by C_1 , then C_2 will be in the set of deleted clauses at the split level of C_1 . Note that although we know that C_1 depends on split k , C_1 may also depend on other splits and thus have a split level higher than l_k .

Our goal is to reinsert the deleted clauses at C_1 's split level. But C_1 itself, after having reduced C_2 , may have been reduced by some clause C_3 , hence C_1 will not necessarily be in the current clause set, but in the deleted set at the level of C_3 . Since we don't know whether C_3 depends on split k or not, C_1 may be sitting in the deleted set of any split (except k).

Let us formalize the above discussion. Let $\Psi: N$ be an arbitrary labelled clause set with Ψ of length n , and $1 \leq k \leq n$ be arbitrary but fixed. Furthermore, let

$$D := \bigcup_{\substack{i=1 \\ i \neq k}}^n D_i$$

and

$$R := \{\max(L) \mid L: \Gamma \rightarrow \Delta \in N \cup D \text{ and } l_k \in L\}.$$

Note how the set R is defined to hold the split levels of all clauses that depend on split k , both in N and any deleted set D_i . It follows that the set

$$\bigcup_{l_j \in R} D_j$$

contains all clauses that may have been reduced by a clause in N depending on split k .

Definition 3.1 (Delete_split). *Now we can define the function*

$$\text{delete_split}(\Psi: N, k) := \Psi': N'$$

where

- $\Psi' = \langle \psi'_n, \dots, \psi'_{k+1}, \psi'_{k-1}, \dots, \psi'_1 \rangle$
- $N' = (N \cup D_k \cup \bigcup_{l_j \in R} D_j) |_{\text{levels}(\Psi')}$

with

$$\psi'_j = \begin{cases} \psi_j[D := \emptyset] & \text{if } l_j \in R \\ \psi_j[D := \{L: \Gamma \rightarrow \Delta \in D_j \mid l_k \notin L\}] & \text{otherwise} \end{cases}$$

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

Note that reinserting all clauses in D_j with $l_j \in R$ is an overapproximation, since not every clause in D_j was necessarily reduced by a clause depending on split k . In fact, it may well be that no clause in D_j was reduced by a clause depending on k . If we wanted to reinsert only clauses reduced by clauses depending on 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.

3.2.2 Reduction Rules for Labelled Clause Sets

We now define a reduction¹ relation $\Psi: N \rightarrow \Psi': N'$ on labelled clause sets to capture the structural transformations of the stack taking place during backtracking. The reduction relation is defined by the following four rules, where we assume that $\Psi: N$ is a labelled clause store and Ψ is of length n .

Definition 3.2 (BACKJUMP). *If $n > 0$, $L: \square \in N$ and $\max(L) < l_n$, then*

$$\Psi: N \rightarrow \text{delete_split}(\Psi: N, n).$$

This rule is called BACKJUMP.

Definition 3.3 (BRANCH-CONDENSE). *If $n > 0$, $L: \square \in N$, $\max(L) = l_n$, $b_n = \text{left}$ and $k_{\max} := \max \{k \mid \max_r(\Psi) < k \leq n \text{ and } l_k \notin L\}$ exists, then*

$$\Psi: N \rightarrow \text{delete_split}(\Psi: N, k_{\max}).$$

This rule is called BRANCH-CONDENSE.

Definition 3.4 (RIGHT-COLLAPSE). *If $n > 0$, $L_2: \square \in N$, $\max(L_2) = l_n$ and $b_n = \text{right}$, and $\varphi_n =: \{L_1\}$, then*

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

where $\Psi': N' = \text{delete_split}(\Psi: N, n)$ and

$$L = \begin{cases} L_1 \cap L_2 & \text{if } l_n \notin L_1 \cap L_2 \text{ and either } L_1 \subseteq L_2 \text{ or } L_2 \subseteq L_1 \\ L_1 \cup L_2 \setminus \{l_n\} & \text{otherwise.} \end{cases}$$

This rule is called RIGHT-COLLAPSE.

Definition 3.5 (ENTER-RIGHT). *If $n > 0$, $L: \square \in N$, $\max(L) = l_n$, $b_n = \text{left}$ and $l_k \in L$ for all k with $\max_r(\Psi) < k \leq n$, then $\Psi'': N'' := \text{delete_split}(\Psi: N, n)$ and*

$$\Psi: N \rightarrow \Psi': N',$$

where

$$\Psi' = \langle (n, \emptyset, \emptyset, \text{right}, \{L\}), \psi''_{n-1}, \dots, \psi''_1 \rangle$$

and $N' = N'' \cup B_n$. This rule is called ENTER-RIGHT.

Note that at most one rule is applicable to any given labelled clause set, since the preconditions are mutually exclusive. Furthermore, Lemma 4.1 shows that any sequence $\Psi: N \rightarrow \Psi': N' \rightarrow \dots$ terminates, for any labelled clause set $\Psi: N$ arising in the context of our calculus. Hence each labelled clause set $\Psi: N$ has a unique normal form, which we write $\Psi: N \downarrow$.

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

3.3 Labelled Calculus

This section presents the actual labelled calculus as a set of inference and reduction rules for labelled clause sets.

3.3.1 Rule Semantics

We distinguish inference rules

$$\mathcal{I} \frac{\Psi: N \quad L_1: \Gamma_1 \rightarrow \Delta_1 \quad \dots \quad L_n: \Gamma_n \rightarrow \Delta_n}{\Psi': N' \quad K: \Pi \rightarrow \Lambda}$$

and reduction rules

$$\mathcal{R} \frac{\Psi: N \quad L_1: \Gamma_1 \rightarrow \Delta_1 \quad \dots \quad L_n: \Gamma_n \rightarrow \Delta_n}{\Psi': N' \quad K_1: \Pi_1 \rightarrow \Lambda_1}$$

$$\vdots$$

$$K_k: \Pi_k \rightarrow \Lambda_k$$

The clauses $L_i: \Gamma_i \rightarrow \Delta_i$ are called the *premises* and the clauses $K_{(i)}: \Pi_{(i)} \rightarrow \Lambda_{(i)}$ 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

$$\Psi': (N' \cup \{K: \Pi \rightarrow \Lambda\}).$$

In the case of a reduction, the resulting labelled clause set is

$$\Psi': (N' \setminus \{L_i: \Gamma_i \rightarrow \Delta_i \mid 1 \leq i \leq n\} \cup \{K_j: \Pi_j \rightarrow \Lambda_j \mid 1 \leq j \leq k\}).$$

3.3.2 Inference and Reduction Rules

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. The emphasis lies on the modelling of the splitting process, hence more advanced rules like those discussed in [14] have been omitted, since their presentation here would not add to the understanding of splitting-specific issues. Such rules can be integrated to the present setting in a straightforward way.

Definition 3.6 (Splitting). *The inference*

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

where

1. $\Psi = \langle \psi_n, \dots, \psi_1 \rangle$

2. $L' = L \cup \{l_n + 1\}$
3. $\Psi' = \langle \psi_{n+1}, \psi_n, \dots, \psi_1 \rangle$ with $\psi_{n+1} = (l_n + 1, \{L': \Gamma_2 \rightarrow \Delta_2\}, \emptyset, left, \emptyset)$
4. $vars(\Gamma_1 \rightarrow \Delta_1) \cap vars(\Gamma_2 \rightarrow \Delta_2) = \emptyset$
5. $\Delta_1 \neq \emptyset$ and $\Delta_2 \neq \emptyset$

is called *splitting*.

A new split representing the left branch $\Gamma_1 \rightarrow \Delta_1$ is created on the stack. The remainder is added to the new split's blocked clauses, which will be restored upon backtracking. Furthermore, note that splitting is an inference and the parent clause $\Gamma_1, \Delta_1 \rightarrow \Gamma_2, \Delta_2$ is not removed from the clause set. A concrete proof strategy may require that subsumption deletion be applied to the parent clause immediately after each splitting step (and after each backtracking step, when the corresponding right branch was entered), thus turning splitting into a reduction. In SPASS, splitting is a reduction in this sense.

Definition 3.7 (Backtracking). *The reduction*

$$\mathcal{R} \frac{\Psi : N \quad L : \square}{\Psi : N \downarrow}$$

is called *backtracking*.

Definition 3.8 (Resolution). *The inference*

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

where

1. $\sigma = mgu(A, B)$

is called *resolution*.

Definition 3.9 (Subsumption Deletion). *The reduction*

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

where

1. $\Gamma_2 \rightarrow \Delta_2$ is subsumed by $\Gamma_1 \rightarrow \Delta_1$

2. $l_{m_1} := \max(L_1)$
3. $l_{m_2} := \max(L_2)$
4. $\Psi = \langle \psi_n, \dots, \psi_{m_1}, \dots, \psi_1 \rangle$
5. $\Psi' = \begin{cases} \Psi & \text{if } m_1 = m_2 \\ \langle \psi_n, \dots, \psi_{m_1}[D := D_{m_1} \cup \{L_2: \Gamma_2 \rightarrow \Delta_2\}], \dots, \psi_1 \rangle & \text{otherwise} \end{cases}$

is called *subsumption deletion*.

The subsumption deletion rule is presented here as one prominent example of a reduction rule, of which many more exist [13]. They all have in common that a clause is simplified (or removed), either because of some property inherent to the clause itself, or because of the presence of another clause (as with subsumption deletion). In the first case, no particular precautions are needed with respect to splitting. In the second case however, we must account for the possibility that the reducing (here: subsuming) clause will eventually be removed from the clause set, e.g., because a split that the clause depends on is removed by branch condensation. This is why we store the subsumed clause at the subsuming clause's level on the split stack. On the other hand, if both the subsumer and the subsumee have the same split level, then one can show (and we will, in Proposition 4.4) that there always remains a subsuming clause in the current clause set as long as the corresponding split is not deleted, hence we can remove and forget the subsumed clause.

Definition 3.10 (Factoring). *The inferences*

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

and

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

where

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

are called *factoring right* and *factoring left*, respectively.

4 Correctness

In this chapter, we introduce the concept of satisfiability of labelled clause sets, we establish some important properties of derivations in the labelled calculus, and we show that all the transformations of the stack presented in Chapter 3 maintain labelled clause set satisfiability, which implies soundness of the labelled calculus. Finally, we give a completeness result.

4.1 Derivations in the Labelled Calculus

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

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

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 $\Psi_{i-1} : N_{i-1} \triangleright \Psi_i : N_i$ the i th step of \mathcal{D} . Sometimes we need to make it explicit that a split belongs to a particular split stack in a derivation. For example, we will write ψ_j^i for the j th split of split stack Ψ_i , and D_j^i for the deleted set of ψ_j^i .

Clearly, each ψ_j is the result of a unique splitting step of a clause $\Gamma_1^j, \Gamma_2^j \rightarrow \Delta_1^j, \Delta_2^j$ into components $\Gamma_1^j \rightarrow \Delta_1^j$ and $\Gamma_2^j \rightarrow \Delta_2^j$, since the splitting rule is the only rule extending the stack. For a given stack Ψ , we denote those components by $s_\Psi^1(j)$ and $s_\Psi^2(j)$, respectively, and the "active" component by

$$s_\Psi(j) := \begin{cases} s_\Psi^1(j) & \text{if } b_j = \textit{left} \\ s_\Psi^2(j) & \text{if } b_j = \textit{right}. \end{cases}$$

In the following, we omit the subscript whenever Ψ is clear from the context.

4.1.1 Proving Derivation Invariants

If backtracking was applied, then it follows from the definition of the backtracking rule that

$$\Psi_i : N_i = (\Psi_{i-1} : N_{i-1}) \downarrow$$

Figure 4.1 illustrates the relation between backtracking and stack reduction rules. To prove that a property P of labelled clause sets is an invariant of any derivation, we will proceed in two steps:

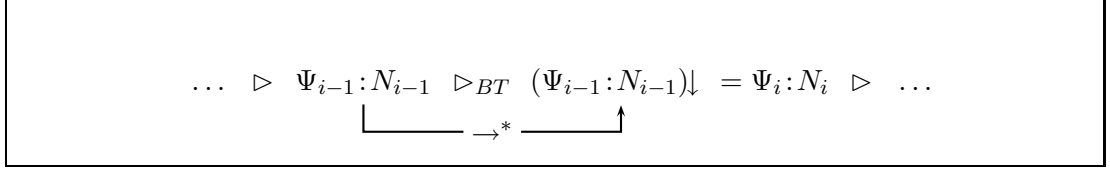


Figure 4.1: A backtracking step BT and the associated sequence of reductions.

1. we show that P is maintained by each of the reduction rules defined in section 3.2.2, it then follows by induction that the normal form $(\Psi_i:N_i)\downarrow$ satisfies P , whenever $\Psi_i:N_i$ does;
2. we use induction over the derivation to show that P holds at each $\Psi_i:N_i$, using the result of 1. for the backtracking case.

Of course, for 1., we need the reduction relation on labelled clause sets to be terminating:

Lemma 4.1 (Stack reduction terminates). *For any labelled clause set $\Psi:N$ in a derivation, any sequence of reductions $\Psi:N \rightarrow \Psi':N' \rightarrow \dots$ terminates.*

Proof. Any reduction is applicable only if Ψ is non-empty, and there is an empty clause in N . All reductions except ENTER-RIGHT reduce the stack size by one. But after ENTER-RIGHT, the clause $L:\square$ is not in N' , by definition of $delete_split(\Psi:N, n)$. Since both Ψ and N are of finite size, it follows that eventually either $\Psi' = \langle \rangle$ or N' contains no more empty clause. ■

4.2 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. We define the following set of clause sets:

$$\mathcal{N}_i := \left\{ (N_i \cup \bigcup_{j=1}^n D_j) \upharpoonright_L \cup B_k \mid k \in \{1, \dots, n\}, b_k = left, L = \{l_1, \dots, l_{k-1}\} \right\}$$

We will use the notation N_i^k to denote a particular $(N_i \cup \bigcup_{j=1}^n D_j) \upharpoonright_L \cup B_k \in \mathcal{N}_i$. We call N_i the *active clause set* of $\Psi_i:N_i$, and \mathcal{N}_i the set of *inactive clause sets* of $\Psi_i:N_i$.

Definition 4.1 (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.

Our goal is to show that the rules of the labelled calculus preserve satisfiability of labelled clause sets, i.e., for each step $\Psi_{i-1} : N_{i-1} \triangleright \Psi_i : N_i$ in a derivation, $\Psi_{i-1} : N_{i-1}$ is satisfiable if and only if $\Psi_i : N_i$ is satisfiable.

4.3 Properties of Derivations

We will now prove invariants of derivations which we will need later to show the correctness of the labelled calculus. Let us first note that leaf markers are non-empty only when a left branch has been closed:

Lemma 4.2 (Existence of leaf markers). *Let $\Psi_i : N_i$ be an arbitrary labelled clause set in a derivation, and let $j \in \{1, \dots, n\}$ be arbitrary. Then $\varphi_j = \emptyset$ whenever $b_j = \text{left}$, and $\varphi_j = \{L\}$ for some L , whenever $b_j = \text{right}$.*

Proof. It is easy to see that the stack reduction rules maintain this property, since leaf markers are set to $\{L\}$ by ENTER-RIGHT and not modified by any other rules. For the calculus rules, the only relevant case is the splitting rule, which only produces splits where $\varphi = \emptyset$ and $b = \text{left}$. ■

Next we show that clause labels are correctly inherited throughout derivations:

Lemma 4.3 (Minimality of parent labels). *Let \mathcal{D} be a derivation, $\Psi_i : N_i$ an arbitrary labelled clause set in \mathcal{D} , and $l \in \text{levels}(\Psi_i)$, 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 $L' : C \in N_i \cup \bigcup_{j=1}^n D_j^i$ 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 case is resolution, but again, it is easy to see that the property is maintained by the definition of the label of the conclusion. ■

4.3.1 Label-validity

We now establish the concept of *label validity*, which will justify our use of the notation l_j when referring to an element of a clause label or leaf marker (in the sense that it will guarantee the existence of split ψ_j with split level l_j).

Definition 4.2 (Label-validity). Let $\Psi_i : N_i$ be a labelled clause set.

1. We say that the active set N_i is label-valid, if $L \subseteq \text{levels}(\Psi)$ for every $L : \Gamma \rightarrow \Delta \in N_i$;
2. we say that some inactive set $N_i^k \in \mathcal{N}_i$ is label-valid, if $L \subseteq \{l_1, \dots, l_k\}$ for every $L : \Gamma \rightarrow \Delta \in N_i^k$;
3. finally, we say that a leaf-marker $\varphi_j = \{L\}$ is label-valid, if $L \subseteq \{l_1, \dots, l_j\}$.

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.

Lemma 4.4 (Split deletion and label-validity). Let $\Psi : N$ be an arbitrary labelled clause set, let $m \in \{1, \dots, n\}$ 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 $\Psi' : N' := \text{delete_split}(\Psi : N, m)$ is label-valid if $\Psi : N$ is label-valid.

Proof.

1. Label-validity of $N' = (N \cup D_k \cup \bigcup_{l_j \in R} D_j)|_{\text{levels}(\Psi')}$ is immediate.
2. Let $k \in \{1, \dots, n-1\} \setminus \{m\}$ 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 $L \cup \{l_k\}$, 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 $N'^k = (N' \cup \bigcup_{j=1}^n D'_j)|_{\text{levels}(\Psi')} \cup B'_k$ is label-valid.
3. Let $k \in \{1, \dots, n-1\} \setminus \{m\}$ 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 4.5 (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. We distinguish which rule was applied to obtain $\Psi'_i : N'_i$.

BACKJUMP: Follows directly with Lemma 4.4.

BRANCH-CONDENSE: Follows again with Lemma 4.4. Note that the assumptions of Lemma 4.4 are fulfilled: Since $l_{k_{max}}$ is the greatest split level (below the last backtracking level) that the empty clause doesn't depend on, it follows by Lemma 4.3 that the parent clauses of all splits below k_{max} don't depend on $l_{k_{max}}$ either. Furthermore, since for all $j > k_{max}$, it holds that $b_j = \text{left}$, we know by Lemma 4.2 that $\varphi_j = \emptyset$.

RIGHT-COLLAPSE: Like in the **BACKJUMP**-case, Lemma 4.4 guarantees label-validity of all clauses, except the new empty clause $L : \square$. Hence we need to show that $L \subseteq \text{levels}(\Psi')$. But this is obvious, since by assumption, both L_1 and L_2 are subsets of $\text{levels}(\Psi)$, and by the definition of L , we know that $l_n \notin L$.

ENTER-RIGHT: We again use Lemma 4.4, and note that the new leaf marker is label-valid, since the empty clause $L : \square$ was label-valid by assumption. ■

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

Proposition 4.1 (Label-validity in derivations). *Any labelled clause set in any derivation is label-valid.*

Proof. Let $\Psi_0 : N_0 \triangleright \Psi_1 : N_1 \triangleright \dots$ 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 with Lemma 4.5. In the case of splitting, we know by induction hypothesis that $L \subseteq \text{levels}(\Psi_{i-1})$, hence also $L' \subseteq \text{levels}(\Psi_i)$ and therefore N_i is label-valid since N_{i-1} is label-valid by induction hypothesis. Label-validity of the N_i^k and the leaf markers is obvious. For resolution, note that the union of two valid labels is again valid. Finally, the subsumption deletion and factoring cases are trivial. ■

4.3.2 Path-validity

We now define a property of labelled clause sets, *path-validity*, which states that the clauses in the active and deleted sets follow logically from the initial clause set and all active split clauses, and that the initial clause set together with the active split clauses described by a leaf marker is unsatisfiable.

Definition 4.3 (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. $N_0 \cup \bigcup_{l_j \in L} s(j) \models C$ for every $L : C \in N_i \cup \bigcup_{j=1}^n D_j$, and
2. $N_0 \cup \bigcup_{l_j \in L \setminus \{l_k\}} s(j) \cup \bigcup_{l_j \in L \cap \{l_k\}} s^1(j) \models \perp$ for every $\varphi_k = \{L\}$.

Lemma 4.6 (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 $\Psi_j : N_j$ are path-valid, for $j \in \{0, \dots, i\}$.*

Proof. Assume all $\Psi_j : N_j$ are path-valid, for $j \in \{0, \dots, i\}$. 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: Let

$$M_1 := N_0 \cup \bigcup_{l_j \in L_1 \setminus \{l_n\}} s(j) \cup \bigcup_{l_j \in L_1 \cap \{l_n\}} s^1(j),$$

$$M_2 := N_0 \cup \bigcup_{l_j \in L_2} s(j)$$

and

$$M := N_0 \cup \bigcup_{l_j \in L} s(j).$$

By assumption, $M_1 \models \perp$ and $M_2 \models \perp$. In the case where $L = L_1 \cap L_2$, we immediately get $M \models \perp$ since $M_1 \subseteq M$ or $M_2 \subseteq M$. For the case where $L = L_1 \cup L_2 \setminus \{l_n\}$, there are two possibilities:

1. If $l_n \in L_1 \cap L_2$, then $s^1(n) \in M_1$ and $s^2(n) \in M_2$. Assume the k th step of the derivation was the last splitting step, i.e. the step that produced split n . Assume the parent clause was $L': \Gamma_1, \Gamma_2 \rightarrow \Delta_1, \Delta_2$, hence $s^1(n) = \Gamma_1 \rightarrow \Delta_1$ and $s^2(n) = \Gamma_2 \rightarrow \Delta_2$. Since no further splitting step occurred after step k , we know that $L' \subseteq L_1$ and $L' \subseteq L_2$. By assumption, we also know that $N_0 \cup \bigcup_{l_j \in L'} s(j) \models s^1(n) \vee s^2(n)$. But since $M_1 \models \perp$ and $M_2 \models \perp$, it follows that $N_0 \cup \bigcup_{l_j \in L'} s(j) \models \perp$ and therefore $M \models \perp$, since $L' \subseteq L$.
2. If $L_1 \not\subseteq L_2$ and $L_2 \not\subseteq L_1$, then either $L_1 \subseteq L$ or $L_2 \subseteq L$, hence either $M_1 \subseteq M$ or $M_2 \subseteq M$, so $M \models \perp$ follows immediately.

ENTER-RIGHT: By assumption that $N_0 \cup \bigcup_{l_j \in L} D_j \models \perp$, for the empty clause L : \square . This immediately implies statement 2, since the only new leaf marker is $\varphi'_n = \{L\}$. Let us show that the blocked clauses $B'_n = \{s^2(n)\}$ are also path-valid: Again let $L': \Gamma_1, \Gamma_2 \rightarrow \Delta_1, \Delta_2$ be the parent clause of split n . By assumption, $N_0 \cup \bigcup_{l_j \in L'} s(j) \models s^1(n) \vee s^2(n)$. Since $l_n \in L$, we know that $N_0 \cup \bigcup_{l_j \in L'} s(j) \cup \{s^1(n)\} \models \perp$ and hence $N_0 \cup \bigcup_{l_j \in L'} s(j) \models s^2(n)$. \blacksquare

Proposition 4.2 (Path-validity in derivations). *Any labelled clause set in any derivation is path-valid.*

Proof. Let $\Psi_0 : N_0 \triangleright \Psi_1 : N_1 \triangleright \dots$ be an arbitrary derivation. We show that any $\Psi_i : N_i$ is path-valid by induction over the derivation. Clearly, $\Psi_0 : N_0$ is path-valid. For the inductive case, we assume all $\Psi_j : N_j$ are path-valid, for $j \in \{0, \dots, i-1\}$, and distinguish which calculus rule was applied to obtain $\Psi_i : N_i$.

- For splitting, we are adding the left split clause to the active set, so $s(n) = s^1(n) \in N_i$ and path-validity is trivially maintained.
- For backtracking, the statement follows by induction hypothesis and Lemma 4.6.

- Since resolution is sound, we know that the conclusion follows logically from the premises. The statement then follows from the fact that the conclusion's label is the union of both premises' labels.
- For subsumption deletion, the statement follows from the fact that

$$N_i \cup \bigcup_{j=1}^n D_j^i \subseteq N_{i-1} \cup \bigcup_{j=1}^n D_j^{i-1}$$

and induction hypothesis.

- The factoring case is trivial, since the conclusion follows logically from the premise. ■

4.3.3 Some Derivation Invariants

Corollary 4.1 (Restriction expansion). *Let $\Psi_i : N_i$ be an arbitrary labelled clause set in a derivation, and let $M \subseteq \text{levels}(\Psi_i)$ and $k \in \{1, \dots, n\}$. If $(N_i \cup \bigcup_{j=1}^n D_j)|_{M \cup \{s(k)\}}$ is satisfiable, then $(N_i \cup \bigcup_{j=1}^n D_j)|_{M \cup \{l_k\}}$ is satisfiable.*

Proof. Let $L : C \in (N_i \cup \bigcup_{j=1}^n D_j)|_{M \cup \{l_k\}} \setminus (N_i \cup \bigcup_{j=1}^n D_j)|_{M \cup \{s(k)\}}$. By Proposition 4.2, $L : C$ follows logically from $(N_i \cup \bigcup_{j=1}^n D_j)|_{M \cup \{s(k)\}}$. ■

The following proposition follows trivially from the definition of subsumption deletion:

Proposition 4.3 (Existence of a subsumption deletion step). *Let \mathcal{D} be a derivation and $\Psi_k : N_k$ an arbitrary labelled clause set in \mathcal{D} . For each $L : C$ in any D_j^k , there exists $k' \leq k$ such that subsumption deletion was applied at step k' of \mathcal{D} with $L : C$ the subsumed clause, and a subsuming clause with split level l_j .*

Proof. By induction over the derivation, and induction over sequences of stack reductions for the backtracking case: For Ψ_0 , all D_j are empty, and subsumption deletion is the only rule extending sets of deleted clauses, no stack reduction rule extends sets of deleted clauses. ■

Proposition 4.4 (Existence of subsuming clauses). *Let \mathcal{D} be a derivation, $\Psi_k : N_k$ an arbitrary labelled clause set in \mathcal{D} , and $L : C \in D_j^k$ for some j . Furthermore, let k' be maximal such that subsumption deletion was applied at step k' of \mathcal{D} with $L : C$ as subsumed clause, and a subsuming clause with split level l_j . Then for all $\Psi_i : N_i$ with $i \in \{k', k' + 1, \dots, k\}$:*

1. $L : C \in D_{j,i}^i$, where $l_{j,i}^i = l_j^k$, and
2. there exists a clause $L' : C'$ with split level l_j , such that $L' : C'$ subsumes $L : C$ and either $L' : C' \in N_i$ or $L' : C' \in D_l^i$ for some $l \in \text{levels}(\Psi_i)$. Furthermore, if $L' : C' \in D_l^i$, then it holds that $h > k'$, where h is maximal such that subsumption deletion was applied at step h of \mathcal{D} with $L' : C'$ as subsumed clause, and there exists a clause $L'' : C'' \in N_i$ that subsumes $L : C$.

Proof. First of all, note that k' exists, by Proposition 4.3. We first show 1. By definition of subsumption deletion, $L : C \in D_{j'}^{k'}$ where $l_{j'}^{k'} = l_j^k$. Assume for contradiction that there exists some $i \in \{k' + 1, \dots, k - 1\}$ such that $L : C \notin D_{j_i}^i$ for $l_{j_i}^i = l_j^k$. Since $L : C \in D_j^k$, and subsumption deletion is the only rule extending the deleted sets, one of the steps $k' + 1, \dots, k$ must have been a subsumption step with $L : C$ as subsumed clause and a subsuming clause with split level l_j . This is a contradiction to the maximality of k' .

Let us now show 2. We first show the property is maintained by all stack reduction rules. So let $i \in \{k', \dots, k - 1\}$ and assume there exists a clause $L' : C'$ with split level l_j , such that $L' : C'$ subsumes $L : C$ and either $L' : C' \in N_i$ or $L' : C' \in D_l^i$ for some $l \in \text{levels}(\Psi_i)$, and assume $\Psi_i : N_i \rightarrow \Psi'_i : N'_i$. Note that any stack reduction rule invokes $\text{delete_split}(\Psi_i : N_i, m)$ for some $m \in \text{levels}(\Psi_i)$ (e.g., $m = k_{\max}$ for BRANCH-CONDENSE). We know that $l_m \notin L'$, since otherwise we would have $D_{j'}^{k'} = \emptyset$ (where $l_{j'}^{k'} = l_j$) by definition of $\text{delete_split}(\Psi_i : N_i, m)$, a contradiction to 1. So $L' : C'$ does not depend on l_m , and hence if $L' : C' \in N_i$, then also $L' : C' \in N'_i$. On the other hand, if $L' : C' \in D_l^i$, then either $L' : C' \in D_l^i$, or $L' : C' \in N'_i$ if $l \in R \cup \{l_m\}$.

We now show that the property indeed holds for all $\Psi_i : N_i$ with $i \in \{k', k' + 1, \dots, k\}$, by induction on i . We have already shown that it holds for $i = k'$. For the inductive step, assume it holds for $i - 1$. The cases splitting, resolution and factoring are trivial, and the backtracking case follows from the above discussion. In the case of subsumption deletion, it is easy to see that the property is maintained if the subsumed clause does not itself subsume $L : C$. So let $h = i$ be maximal such that step h of \mathcal{D} is subsumption deletion with $L' : C'$ as subsumed clause, $L'' : C''$ as subsuming clause, let l_m be the split level of $L'' : C''$ and assume that $L' : C'$ subsumes $L : C$. Observe that if $l_m \neq \max(L')$, then $L' : C' \in D_m^h$, by definition of subsumption deletion, and so property 2. holds. On the other hand, if $l_m = \max(L')$, then $L' : C'$ is removed. But since the subsuming clause also subsumes $L : C$ and is in N_h , property 2 still holds. \blacksquare

Corollary 4.2 (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 $N_i \cup \bigcup_{j=1}^n D_j$ is satisfiable.*

Proof. For the forward direction, we know by Proposition 4.4 that for every clause $L : C$ in any D_j , there exists a subsuming clause in N_i . Hence $N_i \cup \bigcup_{j=1}^n D_j$ is satisfiable if N_i is satisfiable. The backward direction is trivial. \blacksquare

Lemma 4.7 (Split deletion and inactive clause sets). *Let $\Psi : N$ be an arbitrary labelled clause set with Ψ of length n , and let $m \in \{1, \dots, n\}$ be such that for all $j > m$, the parent clause of split j does not depend on l_m , and let $\Psi' : N' = \text{delete_split}(\Psi : N, m)$. Then for all $k \in \{1, \dots, n - 1\}$,*

$$N'^k = \begin{cases} N^k & \text{if } k < m \\ N^{k+1} \setminus \{l_m\} & \text{if } k \geq m. \end{cases}$$

Proof. Assume $k < m$, and let $L := \{l'_1, \dots, l'_{k-1}\} = \{l_1, \dots, l_{k-1}\}$. Then

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

Equation (1) is obtained by plugging in the definition of N' , and observing that $B'_k = B_k$. For equation (2), we use the fact that $\bigcup_{j=1}^{n-1} D'_j = \bigcup_{j=1}^{m-1} D_j|_{\text{levels}(\Psi')} \cup \bigcup_{j=m+1}^n D_j|_{\text{levels}(\Psi')}$ because of level shifting. Finally, equation (3) follows from the fact that $L \subseteq \text{levels}(\Psi')$.

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

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

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

$$\begin{aligned}
N'^k &= (N' \cup \bigcup_{j=1}^{n-1} D'_j)|_L \cup B'_k \\
&= \left((N \cup D_m \cup \bigcup_{l_j \in R} D_j \cup \bigcup_{j=1}^{m-1} D_j \cup \bigcup_{j=m+1}^n D_j)|_{\text{levels}(\Psi')} \right) \Big|_L \cup B_{k+1} \quad (1) \\
&= (N \cup \bigcup_{j=1}^n D_j)|_{\{l_1, \dots, l_k\} \setminus \{l_m\}} \cup B_{k+1} \quad (2) \\
&= N^{k+1}|_{\{l_m\}}.
\end{aligned}$$

We again use $L \subseteq \text{levels}(\Psi')$ for equation (2), and the assumption that the splits below m do not depend on l_m . \blacksquare

4.3.4 Preservation of Satisfiability in Derivations

We can now tackle the preservation of labelled clause set satisfiability, as discussed in Section 4.2. We again begin by showing that satisfiability is preserved by each of the stack reduction rules.

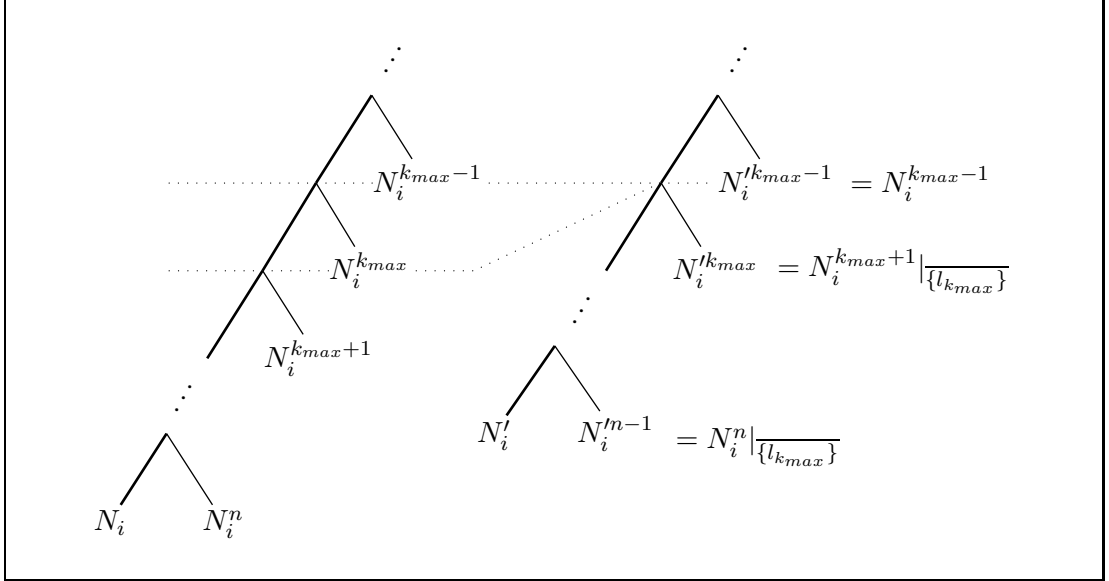


Figure 4.2: Illustration of the proof of Lemma 4.9.

Lemma 4.8 (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 $L : \square \in N_i$. Furthermore, if $N_i^n \in \mathcal{N}_i$ exists – this is the case if $l_n = left$ – then $L : \square \in N_i^n$, 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, $L : \square \in N'_i$ 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 4.7. \blacksquare

Lemma 4.9 (BRANCH-CONDENSE 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 BRANCH-CONDENSE. Then $\Psi_i : N_i$ is satisfiable if and only if $\Psi'_i : N'_i$ is satisfiable.*

Proof. Again, N_i is unsatisfiable, since $L : \square \in N_i$. Because $l_{k_{max}} \notin L$, we also know $L : \square \in N'_i$, and thus N'_i is unsatisfiable. By Lemma 4.7, 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 $b_k = left$ for all $k \in \{k_{max}, \dots, n\}$, we have $\mathcal{N}_i = \{N_i^{k_{max}}, N_i^{k_{max}+1}, \dots, N_i^n\}$.

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'^k is satisfiable, or
2. $(N_i \cup \bigcup_{j=1}^n D_j)|_{\{l_1, \dots, l_{k+1}\} \setminus \{l_{k_{max}}\}}$ is satisfiable.

Since we know $(N_i \cup \bigcup_{j=1}^n D_j)|_{\{l_1, \dots, l_n\} \setminus \{l_{k_{max}}\}}$ is unsatisfiable, it follows that there must be a satisfiable $N'_i{}^k \in \mathcal{N}'_i$.

We have assumed $N_i^{k_{max}} = (N_i \cup \bigcup_{j=1}^n D_j)|_{\{l_1, \dots, l_{k_{max}-1}\}} \cup B_{k_{max}}$ to be satisfiable, hence also $N_i \cup \bigcup_{j=1}^n D_j|_{\{l_1, \dots, l_{k_{max}-1}\}}$ is satisfiable – this provides the induction base. For the inductive step, let $k \geq k_{max}$ and assume that

$$(N_i \cup \bigcup_{j=1}^n D_j)|_{\{l_1, \dots, l_k\} \setminus \{l_{k_{max}}\}}$$

is satisfiable. Assume parent clause of the $k + 1$ st split of Ψ'_i (or the k th split of Ψ_i) was $L' : s^1(k + 1) \vee s^2(k + 1)$. Since $l_{k+1} \in L$, we know by Lemma 4.3 that $L' \subseteq L$. Since $l_{k_{max}} \notin L$, we also have $l_{k_{max}} \notin L'$. But then, by path-validity (Proposition 4.2), we know that

$$(N_i \cup \bigcup_{j=1}^n D_j)|_{\{l_1, \dots, l_k\} \setminus \{l_{k_{max}}\}} \models s^1_{\Psi_i}(k + 1) \vee s^2_{\Psi_i}(k + 1).$$

Hence either

1. $(N_i \cup \bigcup_{j=1}^n D_j)|_{\{l_1, \dots, l_k\} \setminus \{l_{k_{max}}\}} \cup \{s^1_{\Psi_i}(k + 1)\}$

is satisfiable, but then by Corollary 4.1, also

$$(N_i \cup \bigcup_{j=1}^n D_j)|_{\{l_1, \dots, l_k, l_{k+1}\} \setminus \{l_{k_{max}}\}}$$

is satisfiable.

2. Or

$$(N_i \cup \bigcup_{j=1}^n D_j)|_{\{l_1, \dots, l_k\} \setminus \{l_{k_{max}}\}} \cup \{s^2_{\Psi_i}(k + 1)\} = N_i^{k+1} \Big|_{\{l_{k_{max}}\}}$$

is satisfiable. By Lemma 4.7, we have $N_i^{k+1} \Big|_{\{l_{k_{max}}\}} = N_i'^k$, hence $N_i'^k$ is satisfiable.

Still for the forward direction, let us now assume that some N_i^k is satisfiable, for $k \in \{k_{max} + 1, \dots, n\}$. Then it immediately follows with Lemma 4.7 that $N_i'^{k-1} = N_i^k \Big|_{\{l_{k_{max}}\}}$ is satisfiable.

Let us now prove the backward direction. So assume $N_i^{l_k}$ is satisfiable, for some $k \in \{k_{max}, \dots, n-1\}$. Again by Lemma 4.7,

$$N_i^{l_k} = N_i^{k+1} |_{\overline{\{l_{k_{max}}\}}} = (N_i \cup \bigcup_{j=1}^n D_j) |_{\{l_1, \dots, l_k\} \setminus \{l_{k_{max}}\}} \cup B_{k+1}$$

is satisfiable. By path-validity (Proposition 4.2), we know that

$$(N_i \cup \bigcup_{j=1}^n D_j) |_{\{l_1, \dots, l_{k_{max}-1}\}} \models s_{\Psi_i}^1(k_{max}) \vee s_{\Psi_i}^2(k_{max}).$$

Hence we again distinguish two cases:

1. Either

$$(N_i \cup \bigcup_{j=1}^n D_j) |_{\{l_1, \dots, l_k\} \setminus \{l_{k_{max}}\}} \cup \{s_{\Psi_i}^1(k_{max})\} \cup B_{k+1}$$

is satisfiable. But then also

$$(N_i \cup \bigcup_{j=1}^n D_j) |_{\{l_1, \dots, l_k\}} \cup B_{k+1} = N_i^{k+1}$$

is satisfiable.

2. Or

$$(N_i \cup \bigcup_{j=1}^n D_j) |_{\{l_1, \dots, l_k\} \setminus \{l_{k_{max}}\}} \cup \{s_{\Psi_i}^2(k_{max})\} \cup B_{k+1}$$

hence also

$$(N_i \cup \bigcup_{j=1}^n D_j) |_{\{l_1, \dots, l_{k_{max}-1}\}} \cup B_{k_{max}}$$

is satisfiable, since $B_{k_{max}} = \{s_{\Psi_i}^2(k_{max})\}$. ■

Lemma 4.10 (RIGHT-COLLAPSE 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 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 4.7 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 4.2). The second case is immediate. \blacksquare

Lemma 4.11 (ENTER-RIGHT 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 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 4.7, $N_i^{k'} = N_i^k$ 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 $N'_i \subseteq N_i^n$. For the backward direction, assume that N'_i is satisfiable. We show that then N_i^n must also be satisfiable. So let $L: C \in N_i^n \setminus N'_i$ be arbitrary but fixed. Let us show that $L: C$ is redundant with respect to N'_i . We inductively define integers j_l and k_l , and clauses $L_l: C_l$ for $l \geq 1$ as follows: By definition of N_i^n and N'_i , we know that $L: C \in D_{j_1}$ such that $j_1 \notin R \cup \{l_n\}$, and $l_n \notin L$. Let k_1 be the step of the derivation where $L: C$ was subsumed (k_1 exists by Proposition 4.3). By Proposition 4.4, there exists a clause $L_1: C_1 \in N_i \cup \bigcup_{j=1}^n D_j$ such that $L_1: C_1$ subsumes $L: C$ and $\max(L_1) = j_1$.

- If $l_n \in L_1$, then we know $L: C \in N'_i$, by definition of *delete_split*.
- Otherwise, if $l_n \notin L_1$
 - If $L_1: C_1 \in N_i$, then also $L: C \in N'_i$, by definition of *delete_split*.
 - Otherwise, $L_1: C_1 \in D_{j_2}$. Let k_2 be the step of the derivation where $L_1: C_1$ was subsumed. By Proposition 4.4, we know that $k_2 > k_1$, and that there exists $L_2: C_2$ with $\max(L_1) = j_1$, such that $L_2: C_2$ subsumes $L_1: C_1$ and hence also subsumes $L: C$.

Clearly, some k_m will be the last subsumption step in the derivation where $L_{m-1}: C_{m-1} \in D_{j_m}$. The clause $L_m: C_m$ subsumes $L_{m-1}: C_{m-1}$ and hence also subsumes $L: C$, but $L_m: C_m$ was not itself subsumed. Therefore, $L_m: C_m \in N_i$.

- If $l_n \in L_m$, then $L_{m-1}: C_{m-1} \in N'_i$, by definition of *delete_split*.
- Otherwise $l_n \notin L_m$, but then $L_m: C_m \in N'_i$, by definition of *delete_split*.

Both $L_{m-1}: C_{m-1}$ and $L_m: C_m$ subsume $L: C$, hence in both cases, the clause $L: C$ is redundant with respect to N'_i . ■

Proposition 4.5 (Calculus rules maintain satisfiability). *Let $\Psi_i: N_i$ be an arbitrary labelled clause set in a derivation, and assume $\Psi_i: N_i \triangleright \Psi_{i+1}: N_{i+1}$. Then $\Psi_i: N_i$ is satisfiable if and only if $\Psi_{i+1}: N_{i+1}$ is satisfiable.*

Proof. Let n be the length of Ψ_i . We distinguish which rule was applied to obtain $\Psi_{i+1}: N_{i+1}$.

- The cases resolution and factoring are trivial, since the conclusion follows logically from the premises.
- For the splitting case, let us first observe that Ψ_{i+1} has length $n + 1$, and that for all $1 \leq k \leq n$,

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

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 $N_{i+1} = N_i \cup \{L': \Gamma_1 \rightarrow \Delta_1\}$ and

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

since $l_n + 1 \in L'$ and thus $N_{i+1}|_{\{l_1, \dots, l_n\}} = N_i$, and the sets of deleted clauses are not changed by splitting. By Corollary 4.2, 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 $N_i \cup \{L': \Gamma_2 \rightarrow \Delta_2\}$. Therefore, it suffices to show that N_i is satisfiable if and only if $N_i \cup \{L': \Gamma_1 \rightarrow \Delta_1\}$ or $N_i \cup \{L': \Gamma_2 \rightarrow \Delta_2\}$ is satisfiable, and this is immediate since $\Gamma_1 \rightarrow \Delta_1$ and $\Gamma_2 \rightarrow \Delta_2$ are variable-disjoint.

- For backtracking, we use induction over the stack reductions together with Lemmata 4.8 to 4.11.
- For subsumption deletion, observe that N_i and N_{i+1} are equisatisfiable, since the subsuming clause $L_1: \Gamma_1 \rightarrow \Delta_1$ is still in N_{i+1} . ■

4.4 Soundness and Completeness

In this section, we prove the soundness of the labelled calculus, and we show completeness for the case where only a finite number of splitting steps are allowed.

Theorem 4.1 (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 $\Psi_0 : N_0 \triangleright^* \Psi_m : N_m$ be an arbitrary derivation starting with $\Psi_0 : N_0$. Then $\Psi_m : N_m$ is satisfiable if $\Psi_0 : N_0$ is satisfiable.*

Proof. By induction over the derivation, using Proposition 4.5. ■

To prove general completeness, the usual notion of fairness for derivations needs to be extended to take the splitting operation into account. While this is not difficult to do, it is beyond the scope of this thesis. However, a completeness result can be obtained if we assume a finite number of splitting steps for any given derivation.

Theorem 4.2 (Completeness for finite splitting). *Let N be an arbitrary clause set. Let $N_0 := \{\emptyset : C \mid C \in N\}$ be the associated set of labelled clauses, and let $\Psi_0 := \langle \rangle$. If N is unsatisfiable, then any fair derivation with finite splitting and exhaustive application of the other calculus rules yields the empty clause \emptyset : \square .*

Proof. Assume N is unsatisfiable. Let $m \in \mathbb{N}$ be the maximal number of splitting steps allowed. Let $\Psi_0 : N_0 \triangleright^* \Psi_k : N_k$ be a derivation such that step k was the m th application of the splitting rule. By Proposition 4.5, $\Psi_k : N_k$ is unsatisfiable. We now show by induction over the number of inactive clause sets, that we eventually derive the empty clause \emptyset : \square . By fairness and completeness of general resolution, the empty clause will be derived by any fair application of the rules resolution, factoring and subsumption deletion to an unsatisfiable set of labelled clauses. Once the empty clause has been derived in the active clause set, the only rule eventually applicable is backtracking (since splitting is forbidden), which makes the number of inactive clause sets strictly smaller. Eventually, either an empty clause with label \emptyset is derived, or the last active clause set corresponds to a right branch at split level one. Then, by path-validity (Proposition 4.2), the only split level in the empty clause's label is one. The final ENTER-RIGHT step then produces the desired clause \emptyset : \square . ■

Note that we restrict ourselves to the finite splitting case because we have not introduced a formal notion of fairness in the presence of splitting and backtracking. Defining this fairness concept formally allows to prove general completeness without much additional effort, but is beyond the scope of this thesis.

5 Experimental Results

5.1 Implementation

The enhanced backtracking process formalized in this thesis has been integrated into the SPASS [14] theorem prover. The basis for the implementation was 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. 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 discussed in Section 3.2.

5.2 Results on TPTP Problems

The implementation was tested on the TPTP problem library [12], 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, 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. On the other hand, SPASS with improved backtracking ran out of time on 5 problems for which a solution (4 proofs, 1 completion) was found without improved backtracking. The reason why some proofs are no longer found is that after improved backtracking, a different clause set is obtained, and hence the clause selection function causes a different part of the search space to be explored. In that part of the search space, the proof that was previously found may no longer be reachable, and the procedure runs out of time before finding an alternative proof.

6 Future Work

Besides the need for an extended notion of fairness for splitting with backtracking, discussed in Section 4.3.4, there are two more aspects to labelled splitting worth mentioning. We first discuss an extension of the splitting rule which is motivated by the implementation of splitting in SPASS, and we then briefly look at an entirely different labelling scheme in which only clauses are labelled, but clause sets are not.

6.1 Splitting with Lemmata

In SPASS, the following optimization is used during splitting: If the first split part is ground, then its negation is added to the set of blocked clauses of the split. Using the notation of the labelled calculus, splitting with ground lemmata would be expressed like this:

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

where

1. $\Psi = \langle \psi_n, \dots, \psi_1 \rangle$
2. $L' = L \cup \{l_n + 1\}$
3. $\Psi' = \langle \psi_{n+1}, \psi_n, \dots, \psi_1 \rangle$ with $\psi_{n+1} = (l_n + 1, \{L' : \Gamma_2 \rightarrow \Delta_2\} \cup U, \emptyset, left, \emptyset)$
4. $vars(\Gamma_1 \rightarrow \Delta_1) \cap vars(\Gamma_2 \rightarrow \Delta_2) = \emptyset$
5. $\Delta_1 \neq \emptyset$ and $\Delta_2 \neq \emptyset$
6. $\Gamma_1 = A_1, \dots, A_m$ and $\Delta_1 = A'_1, \dots, A'_l$
7. $U = \begin{cases} \bigcup_j \{L' : \rightarrow A_j\} \cup \bigcup_k \{L' : A'_k \rightarrow\} & \text{if } vars(\Gamma_1 \rightarrow \Delta_1) = \emptyset \\ \emptyset & \text{otherwise} \end{cases}$

The set U which is added to the blocked set contains unit clauses that together correspond to the negation of the first split part, whenever the first split part is ground. These unit clauses can be viewed as lemmata that are obtained when the left part of the split has been refuted. Making these unit lemmata available to the right subtree can help a lot in reducing clauses [13]. The reason why we restrict lemma generation to the case where the left split part is ground is that negating a non-ground clause in general leads to the introduction of new Skolem constants

(remember that clauses are implicitly universally quantified). In practice, this tends to extend the search space for the second split part. Proving the correctness of this optimization is beyond the scope of this thesis. However, only minor modifications to the proofs presented in Chapter 4 are needed to show it.

6.2 Clause Labels Only

The use of clause labels to model splitting was first suggested in [9]. In the proposed framework, only clauses have labels, and there is no notion of a global split stack. The clause labels are sequences of (overlined) clauses, where ϵ is the empty sequence. To make sure that clauses depending on complementary parts of a split do not interact, two operators on clause labels, \circ and \bullet , combine the labels of the premises for inferences and reductions, respectively. A special label \ominus (read *blocked*) is used, and if the combination of two clause labels returns \ominus , the corresponding inference or reduction is disallowed. The operators \circ and \bullet are both defined to be the greatest lower bound of the prefix relation, i.e., if one clause label is a prefix of the other, the longer label is returned. The blocked label \ominus is returned whenever the two labels are not prefixes of each other. The inference and reduction rules are extended so that they assign the combined label to the clauses in the conclusion, if the combined label is different from \ominus . The splitting rule is then defined as the inference

$$\mathcal{I} \frac{m: \Gamma_1, \Gamma_2 \rightarrow \Delta_1, \Delta_2}{\frac{m.\overline{\Gamma_1}, \Gamma_2 \rightarrow \Delta_1, \Delta_2: \Gamma_1 \rightarrow \Delta_1}{m.\overline{\Gamma_1}, \Gamma_2 \rightarrow \Delta_1, \Delta_2: \Gamma_2 \rightarrow \Delta_2}}$$

with the usual conditions on variable-disjointness and non-empty Δ_i , and backtracking is defined as the inference

$$\mathcal{I} \frac{m.C: \square \quad m.\overline{C}: \square}{m: \square}$$

and a refutation of a set of clauses (initially labelled with ϵ) is a derivation of $\epsilon: \square$.

The requirement that labels be prefixes of each other can be understood as allowing inferences and reductions only between clauses that lie on the same path of the split tree. This is a stronger condition than just disallowing interaction between clauses in sibling subtrees. For example, two clauses that are the respective results of the splitting of two input clauses $\epsilon: C_1$ and $\epsilon: C_2$ would be labelled with C_1 (or $\overline{C_1}$) and C_2 (or $\overline{C_2}$), respectively. Hence they would not be allowed to interact.

As a possible way of relaxing this restrictive condition on clause labels, consider the following scheme: Clause labels are tuples (S, \mathcal{R}) , where

- S is a set of (overlined) clauses, describing which splits the clause depends on
- \mathcal{R} is a set of sets of (overlined) clauses, describing sets of splits which make the clause redundant.

The splitting rule can then be defined as the inference

$$\mathcal{I} \frac{(S, \mathcal{R}): \Gamma_1, \Gamma_2 \rightarrow \Delta_1, \Delta_2}{\begin{array}{c} (S \cup \{\Gamma_1, \Gamma_2 \rightarrow \Delta_1, \Delta_2\}, \mathcal{R}): \Gamma_1 \rightarrow \Delta_1 \\ (S \cup \{\overline{\Gamma_1}, \overline{\Gamma_2} \rightarrow \overline{\Delta_1}, \overline{\Delta_2}\}, \mathcal{R}): \Gamma_2 \rightarrow \Delta_2 \end{array}}$$

with the usual conditions on variable-disjointness and non-empty Δ_i , and backtracking is defined as the inference

$$\mathcal{I} \frac{(S_1 \cup \{C\}, \mathcal{R}_1): \square \quad (S_2 \cup \{\overline{C}\}, \mathcal{R}_2): \square}{(S_1 \cup S_2, \emptyset): \square}$$

We then define a predicate $block((S_1, \mathcal{R}_1), (S_2, \mathcal{R}_2))$ that is *true* if and only if

- S_1 and S_2 contain complementary clauses (e.g., $C \in S_1$ and $\overline{C} \in S_2$), or
- $R \subseteq S_2$, for some $R \in \mathcal{R}_1$, or
- $R' \subseteq S_1$, for some $R' \in \mathcal{R}_2$.

Finally, the operators \circ and \bullet are defined as

$$(S_1, \mathcal{R}_1) \circ (S_2, \mathcal{R}_2) := \begin{cases} \ominus & \text{if } block((S_1, \mathcal{R}_1), (S_2, \mathcal{R}_2)) \\ (S_1 \cup S_2, \mathcal{R}_1 \cup \mathcal{R}_2) & \text{otherwise} \end{cases}$$

and

$$(S_1, \mathcal{R}_1) \bullet (S_2, \mathcal{R}_2) := \begin{cases} \ominus & \text{if } block((S_1, \mathcal{R}_1), (S_2, \mathcal{R}_2)) \\ (S_2, \mathcal{R}_2 \cup \{S_1\}) & \text{otherwise.} \end{cases}$$

The idea is that we want to block exactly the interactions between clauses that depend on complementary parts of a split. As an example illustrating the motivation behind the definition of \bullet , consider the reduction rule *subsumption separation*:

$$\mathcal{R} \frac{(S_1, \mathcal{R}_1): \Gamma_1 \rightarrow \Delta_1 \quad (S_2, \mathcal{R}_2): \Gamma_2 \rightarrow \Delta_2}{\begin{array}{c} (S_1, \mathcal{R}_1): \Gamma_1 \rightarrow \Delta_1 \\ (S_1, \mathcal{R}_1) \bullet (S_2, \mathcal{R}_2): \Gamma_2 \rightarrow \Delta_2 \end{array}}$$

where $\Gamma_2 \rightarrow \Delta_2$ is subsumed by $\Gamma_1 \rightarrow \Delta_1$. The subsumed clause has the second component of its label extended to indicate that it is redundant with respect to all clauses that depend on the splits described by the first component of the subsuming clause's label.

The above definitions are just a rough sketch, and the discussion of formal properties of the resulting system is beyond the scope of this thesis. However, it seems that by using sets of splits for the labels, instead of sequences, the search space is no longer accurately described by a single tree. Instead, clauses may be "valid" in a number of different trees, and the results of inferences may be shared across branches, in a way similar to what is described in [11].

Bibliography

- [1] Franz Baader and Tobias Nipkow. *Term Rewriting and All That*. Cambridge University Press, United Kingdom, 1998.
- [2] Leo Bachmair and Harald Ganzinger. Resolution theorem proving. In Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 2, pages 19–99. Elsevier, 2001.
- [3] Leo Bachmair, Harald Ganzinger, and Uwe Waldmann. Superposition with simplification as a decision procedure for the monadic class with equality. In Georg Gottlob, Alexander Leitsch, and Daniele Mundici, editors, *Computational Logic and Proof Theory, Third Kurt Gödel Colloquium*, volume 713 of *LNCS*, pages 83–96. Springer, August 1993.
- [4] David Basin, Marcello D’Agostino, Dov Gabbay, Sean Matthews, and Luca Viganò, editors. *Labelled Deduction*. Kluwer Academic Publishers, 200.
- [5] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *J. ACM*, 7(3):201–215, 1960.
- [6] William F. Dowling and Jean H. Gallier. Linear-time algorithms for testing the satisfiability of propositional horn formulae. *Journal of Logic Programming*, 1(3):267–284, 1984.
- [7] Reiner Hähnle. Tableaux and related methods. In Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, volume 1, chapter 3, pages 100–178. Elsevier, 2001.
- [8] Thomas Hillenbrand and Christoph Weidenbach. Superposition for finite domains. Research Report MPI-I-2007-RG1-002, Max-Planck Institute for Informatics, Saarbrücken, Germany, April 2007.
- [9] Tal Lev-Ami, Christoph Weidenbach, Thomas Reps, and Mooly Sagiv. Labelled clauses. In *21st International Conference on Automated Deduction (CADE-21)*, Bremen, Germany, 2007. Springer. Accepted for Publication.
- [10] Robert Nieuwenhuis and Albert Rubio. Paramodulation-based theorem proving. In Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 7, pages 371–443. Elsevier, 2001.
- [11] Alexandre Riazanov and Andrei Voronkov. Splitting without backtracking. In *IJCAI*, pages 611–617, 2001.

- [12] Geoff Sutcliffe and Christian B. Suttner. The TPTP Problem Library: CNF Release v1.2.1. *Journal of Automated Reasoning*, 21(2):177–203, 1998.
- [13] Christoph Weidenbach. Combining superposition, sorts and splitting. In Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, volume 2, chapter 27, pages 1965–2012. Elsevier, 2001.
- [14] Christoph Weidenbach, Renate Schmidt, Thomas Hillenbrand, Rostislav Rusev, and Dalibor Topic. Spass version 3.0. In *21st International Conference on Automated Deduction (CADE-21)*, Bremen, Germany, 2007. Springer. Accepted for Publication.