Exploring and Exploiting Algebraic and Graphical Properties of Resolution

Pascal Fontaine, Stephan Merz, Bruno Woltzenlogel Paleo

Exploring and Exploiting Algebraic and Graphical Properties of Resolution

Pascal Fontaine, Stephan Merz and Bruno Woltzenlogel Paleo
[Pascal.Fontaine,Stephan.Merz,Bruno.WoltzenlogelPaleo]@loria.fr
LORIA, INRIA & Nancy University, Nancy, France

Abstract

Integrating an SMT solver in a certified environment such as an LF-style proof assistant requires the solver to output proofs. Unfortunately, those proofs may be quite large, and the overhead of rechecking the proof may account for a significant fraction of the proof time. In this paper we explore techniques for reducing the sizes of propositional proofs, which are at the core of SMT proofs. Our techniques are justified in an algebra of resolution and rely on a graph-theoretical representation of proofs that allows us to detect the potential for reordering and combining resolution inferences.

1 Introduction

SMT solvers have attracted much interest from various communities, and have been used in many contexts. In many cases it is important that the SMT solver not only provides a yes/no answer about the satisfiability of the input formula but also produces a model in case the formula is satisfiable or a refutation proof if it is not. In the following we concentrate on certifying proofs produced by the SMT solver within a proof assistant. Our goal is to make the deductive power of SMT solvers available to users of proof assistants while retaining high guarantees of correctness.

Building efficient dedicated proof checkers is possible [15, 16]. However, our preliminary work on integrating SMT solvers within proof assistants [10] showed that the overhead of reading and replaying the proof within the proof assistant accounts for a significant fraction of the overall proof time. Recent work (e.g., [4]) has improved on this overhead, but it remains considerable. Since proofs can be large objects, an obvious way to speed up proof reconstruction is to investigate techniques for transforming a given proof into a smaller one, without significantly increasing the complexity of proof generation or reconstruction. This paper reports on work in progress aiming at reducing proof sizes and lengths.

Straightforward (and efficient) techniques can already account for significant reductions of proof lengths. For example, Table 1 presents some experimental observations on the lengths of the proofs generated by veriT [5] for some randomly selected formulas from the SMT-lib. The first column shows the length of the original proof produced by veriT, without any post-processing. The second column shows the length of the proof after eliminating rules whose consequences are never used. As a second obvious step, we remove duplicate inferences, resulting in proofs whose lengths appear in the third column. The last column shows the overall reduction ratio obtained in this way. From our experience, these numbers are fairly representative of the benchmarks contained in the SMT-lib.

The goal of the work presented here is to examine proofs more deeply in order to transform them in less obvious ways and obtain even smaller proofs. As a starting point, we focus on propositional resolution proofs: propositional reasoning is at the core of SMT solvers, and accounts for much of the length of the proofs they produce. (For instance, resolution inferences typically account for 30–50% of the size of proofs generated by veriT.)

To achieve this goal we rely on techniques from proof theory. We start by establishing an isomorphism between resolution proofs and compact algebraic circuit structures (Section 3). We observe some algebraic properties of resolution (section 3.2) and in particular identify properties such as chain associativity and subsumption-free distributivity, which can be exploited to rearrange and combine resolution

	Number of deduction steps			Ratio		
File	1: Raw	2: Pruned	3: Merged	(2)/(1)	(3)/(2)	Overall
22s	3791	1954	1937	0.52	0.99	0.51
dlx1c	7319	5553	2767	0.76	0.50	0.38
fischer3-mutex-10	24841	16938	12060	0.68	0.71	0.49
qg6/iso_icl_sk002	10891	6881	5128	0.63	0.75	0.47
NEQ023_size4	11406	6438	5621	0.56	0.87	0.49
qlock-4-10-10.base.cvc	119386	72505	37998	0.61	0.52	0.32

Table 1: Proof length for some SMT-lib formulas. The proof length is measured as the number of inference steps, where a single step can be an application of transitivity of equality, congruence, (binary or multi-premise) resolution or a rule of linear arithmetic. Reflexivity and symmetry of equality are implicit.

inferences. We introduce *resolution hypergraphs* (section 4) as a representation of resolution proofs that is particularly convenient for detecting redundancies in proofs and for simplifying them. Our objective is to contribute to a general and systematic study of resolution from an algebraic perspective.

Relation to Existing Work

Cotton [8] describes a proof compression technique relying on splitting the unsatisfiability proof into a proof of x and a proof of $\bar{x}$ for some variable x; adequately choosing the variable x can account for some proof size reduction. Bar-Ilan et al. [3] notice that a unit learned clause may be used to simplify the part of the proof generated before this unit clause is learned. They also observe that two resolutions with the same resolved atom occurring in a path from the leaf to the root is symptomatic of unnecessary resolution steps. In those two particular cases they provide a method to rearrange the proof in order to shorten it.

Previous algebraic investigations of resolution were limited to the fragment of Horn clauses and with the distinct goal of developing resolution-based methods of cut-elimination for substructural logics [6]. The idea to map resolution proofs to isomorphic but more compact expressions (i.e. resolution traces) is inspired by the Curry-Howard isomorphism [9], which maps natural deduction proofs of theorems of the implicational fragment of intuitionistic logic to more compact simply typed lambda expressions. The invention of resolution hypergraphs is also motivated by recent proof-theoretic techniques, such as proof nets [11] and atomic flows [12]. Resolution hypergraphs are an alternative representation for resolution in the same spirit as proof nets and atomic flows are alternative representations for linear logic sequent calculi and deep inference calculi, respectively.

Resolution hypergraphs resemble *link graphs* defined by Amjad [1], but differ in a few important ways. Link graphs require proofs to be in tree-like form, whereas it is more natural (and compact) to represent a proof as a dag. Transformation to tree-like form requires duplicating branches, and therefore the link graph of a proof can be exponentially larger than its dag representation. Resolution hypergraphs, on the other hand, can directly represent dag proofs. There is no increase in size; the number of edges is exactly the number of resolution inferences and the number of nodes is exactly the number of leaf clauses in the dag proof. In resolution hypergraphs, every resolution inference corresponds to a single hyperedge, whereas it can correspond to several edges in link graphs, if implicit factoring occurs. This is not only less natural, but also worse from a complexity point of view, since the number of links is then quadratic in the number of literals that have been factored. Moreover, there is also a fundamental difference in the way these two graphical structures are used. Link graphs have been used as a device for heuristically choosing which resolution inferences to perform first, in the hope that the chosen order will make some inferences superfluous and hence result in shorter proofs. In contrast, we use the resolution hypergraph of a proof to analyze it as a whole; subgraphs of particular shapes (e.g. paths and stars) indicate subsets of inferences of the proof that can be replaced by single multi-premise resolution inferences, and hyperedges labeled

by the same atom can be merged under certain conditions.

Amjad [2] proposes another interesting approach to compressing propositional resolution proofs. He addresses the problem of detecting and merging repeated subproofs, and gives an algorithm inspired by data compression techniques for strings. However, by considering a proof essentially as a string, the proposed algorithm fails to exploit basic algebraic properties of resolution such as commutativity. Moreover, the output of the algorithm is not a resolution proof but rather a natural deduction proof. In Amjad's work (and in our main application) this is not an issue and can even be desirable—ultimately, we intend to replay proofs in a proof assistant based on natural deduction. However, we prefer to keep separate the two distinct problems of compressing proofs and of translating them from one calculus to another, aiming for a more modular approach.

2 The Resolution Calculus

A *literal* is an atomic formula or a negated atomic formula. $\mathscr{L}$ denotes the set of all literals, and $\mathscr{L}^+$ is the set of atoms (positive literals). A *clause* is a set of literals. $\bot$ denotes the *empty clause*. We write $\overline{l}$ to denote the complementary literal of literal l, and similarly write $\overline{\kappa}$ for the clause that consists of the complementary literals of those contained in clause κ. A clause containing an atom and its negation is called *tautological*. We let $\mathscr{C}$ denote the set of all clauses and $\mathscr{C}_\circ$ denote the set of all non-tautological clauses. By $\top$, we denote an idealized tautological clause containing all literals, and we write $\mathscr{C}_\circ^\top$ for $\mathscr{C}_\circ \cup \{\top\}$. A clause κ_1 *subsumes* a clause κ_2 if and only if $\kappa_1 \subseteq \kappa_2$ or $\kappa_2 = \top$.

In standard resolution calculus the resolution rule is a partial operation because it can only be applied to clauses that contain complementary literals. Definition 2.1 extends resolution so that it becomes a total binary operation on clauses.

Definition 2.1 (Resolution)**.** A *resolution inference* is an instance of the following rule. The clauses $\kappa_1, \kappa_2 \in \mathscr{C}_\circ^\top$ are called *premises* and κ is called *resolvent* of κ_1 and κ_2.

$$\frac{\kappa_1 \qquad \kappa_2}{\kappa}\,\mathbf{r}$$

$$\text{where} \quad \kappa \doteq \begin{cases} \mathrm{t}((\kappa_1 \setminus \{\ell\}) \cup (\kappa_2 \setminus \{\overline{\ell}\})) & \text{if } \kappa_1 \cap \overline{\kappa_2} = \{\ell\} \\ \mathrm{t}(\kappa_1 \cup \kappa_2) & \text{if } \kappa_1 \cap \overline{\kappa_2} = \emptyset \\ \top & \text{if } |\kappa_1 \cap \overline{\kappa_2}| > 1 \end{cases}$$

$$\text{and} \quad \mathrm{t}(\kappa) \doteq \begin{cases} \top & \text{if } \kappa \text{ is a tautological clause} \\ \kappa & \text{otherwise} \end{cases}$$

In the first case, the literal ℓ is called the *resolved literal*, and its underlying atom the *resolved atom*. If $\kappa_1 \cap \kappa_2 \neq \emptyset$, we say that *factoring* implicitly occurs, and the literals in $\kappa_1 \cap \kappa_2$ are the *factored literals*. A resolution inference is *subsumption-free* if its conclusion is not subsumed by any of its premises. □

A *resolution proof* ψ of an *end clause* κ from a set of clauses C is a directed acyclic graph (dag) of inferences: its source nodes are *axiom inferences* (without premises) whose conclusions are elements of C, the other nodes are resolution inferences, and ψ has a single sink node whose conclusion is κ. The dag contains an edge from an inference ρ_1 to an inference ρ_2 if and only if the conclusion of ρ_1 is a premise of ρ_2. A *refutation* of C is a resolution proof of $\bot$ from C. A proof ψ is *subsumption-free* if and only if for no clause κ in ψ there is a clause κ' in a path from κ to an axiom clause of ψ such that κ' subsumes κ. The set of all proofs is denoted $\mathscr{P}$.

Remark 2.2. $\eta[\varphi]$ denotes a *proof link* to a proof named φ. Proof links are just a convenient syntactic way of representing proof dags with a tree-like notation, as shown in the following example.

Example 2.3. Consider the set of clauses below:

$$\{\underbrace{\{a\}}_{\kappa_1}\ ;\ \underbrace{\{b\}}_{\kappa_2}\ ;\ \underbrace{\{\neg c,d\}}_{\kappa_3}\ ;\ \underbrace{\{\neg d,e\}}_{\kappa_4}\ ;\ \underbrace{\{\neg d,\neg f\}}_{\kappa_5}\ ;\ \underbrace{\{\neg g,c\}}_{\kappa_6}\ ;\ \underbrace{\{\neg b,\neg a,g\}}_{\kappa_7}\ ;\ \underbrace{\{\neg g,\neg e,f\}}_{\kappa_8}\}$$

It can be refuted by the following refutation ψ:[1]

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\neg g,\neg e,f \qquad \eta[\psi_{\kappa_7}]}{\neg e,f,\neg b,\neg a}\,\mathbf{r}_7 \qquad \eta[\psi_{\kappa_1}]}{\neg e,f,\neg b}\,\mathbf{r}_6 \qquad \eta[\psi_{\kappa_2}]}{\neg e,f}\,\mathbf{r}_5 \qquad \dfrac{\neg d,\neg f \qquad \eta[\psi']}{\neg f}\,\mathbf{r}_3}{\neg e}\,\mathbf{r}_2 \qquad \dfrac{\neg d,e \qquad \eta[\psi']}{e}\,\mathbf{r}_4}{\bot}\,\mathbf{r}_1$$

where ψ_{κ_1}, ψ_{κ_2} and ψ_{κ_7} are proofs containing, respectively, only the axiom clauses κ_1, κ_2 and κ_7; and ψ' is:

$$\dfrac{\dfrac{\eta[\psi_{\kappa_2}] \qquad \dfrac{\eta[\psi_{\kappa_1}] \qquad \dfrac{\neg g,c \qquad \eta[\psi_{\kappa_7}]}{c,\neg b,\neg a}\,\mathbf{r}_{11}}{c,\neg b}\,\mathbf{r}_{10}}{c}\,\mathbf{r}_9 \qquad \neg c,d}{d}\,\mathbf{r}_8$$

Note that every proof link points to a subproof whose conclusion clause has been used more than once as a premise. Using proof links we can represent the proof dag as a collection of 5 trees (ψ, ψ', ψ_{κ_1}, ψ_{κ_2} and ψ_{κ_7}). □

In traditional definitions of resolution with forward subsumption (e.g., [14]), the resolution rule is only defined in case $\kappa_1 \cap \overline{\kappa_2} = \{\ell\}$ and neither κ_1 nor κ_2 is a tautological clause. This is because the other two cases only give subsumed resolvents; moreover, tautologies are always deleted. Such a definition is advantageous from the point of view of automated deduction: it helps reducing the search space, also tautologies cannot contribute to refutations. However, from an algebraic point of view, the traditional definition has the disadvantage that resolution becomes a partial operation over $\mathscr{C}$. It is easy to see that every resolution proof satisfying the usual definitions of resolution with implicit forward subsumption is a subsumption-free resolution proof satisfying the definition given here. Moreover, the resolvent of two clauses according to Def. 2.1 is always well-defined and unique (Theorem A.1), and resolution enjoys several interesting algebraic properties, as shown in Section 3.2.

3 Resolution Traces

3.1 A syntax for resolution proofs

As discussed in Section 2 above, our definition allows us to regard the resolution rule as an algebraic operation $\odot$ over the set of clauses $\mathscr{C}_\circ^\top$. Definition 3.1 introduces *resolution traces*, and these are used to represent resolution proofs as algebraic circuits. Since the resolvents of clauses are uniquely defined, they can be suppressed from the trace, which thus becomes smaller than the proof it represents. By **T** (formally introduced in Def. A.2) we denote the function that maps proofs to traces. It is easy to see that **T** is an isomorphism (Thm. A.3).

[1] This refutation was output by the SMT-solver veriT and shown in [5]. Since here we are not interested in the internal structure of atomic formulas, though, we have abbreviated them by the propositional symbols a, b, c, d, e, f and g.

Definition 3.1 (Resolution Trace)**.** A *resolution trace* is a circuit in which the source nodes are clauses from $\mathscr{C}_\circ^\top$ and all other nodes are $\odot$ gates. For convenience, in order not to have to draw traces, a formula-like notation specified by the following grammar is used to denote traces:

$$\mathbf{t} \quad ::= \quad \kappa \quad | \quad (\mathbf{t} \odot \mathbf{t}) \quad | \quad \eta[\mathbf{n}]$$

where $\eta[\mathbf{n}]$ is a *link* to a subtrace having *name* $\mathbf{n}$. $\mathscr{T}$ denotes the set of all resolution traces. □

Notice that the links are necessary when denoting non-tree-like traces, because they contain at least one gate with outdegree greater than one. Because $\mathbf{T}$ is an isomorphism (Thm. A.3), we can define a subsumption-free trace as a trace whose proof is subsumption-free.

Example 3.2. Let ψ be the refutation shown in Example 2.3. Its trace $\mathbf{T}(\psi)$ is shown below:

$$((((\kappa_8 \odot \eta[t_{\psi_{\kappa_7}}]) \odot \eta[t_{\psi_{\kappa_1}}]) \odot \eta[t_{\psi_{\kappa_2}}]) \odot (\kappa_5 \odot \eta[t_{\psi'}])) \odot (\kappa_4 \odot \eta[t_{\psi'}])$$

$$t_{\psi'}: \ (\eta[t_{\psi_{\kappa_2}}] \odot (\eta[t_{\psi_{\kappa_1}}] \odot (\kappa_6 \odot \eta[t_{\psi_{\kappa_7}}]))) \odot \kappa_3 \qquad t_{\psi_{\kappa_1}}: \ \kappa_1 \qquad t_{\psi_{\kappa_2}}: \ \kappa_2 \qquad t_{\psi_{\kappa_7}}: \ \kappa_7$$

3.2 Exploring Algebraic Properties of Resolution Traces

Definition 3.1 gives the signature of an algebra of traces that has $\odot$ as its binary operation. The following definition interprets a trace as the clause it derives. This definition is the basis for discussing algebraic properties of (the operation denoted by) $\odot$.

Definition 3.3 (Evaluation of traces, evaluation equivalence)**.** The *evaluation* $\langle t \rangle$ of a trace t is the conclusion clause of $\mathbf{T}^{-1}(t)$. Two traces t_1 and t_2 are *evaluation equivalent* (denoted $t_1 \equiv t_2$) if and only if $\langle t_1 \rangle = \langle t_2 \rangle$. □

We now study some algebraic facts of resolution traces. We do so not only for their intrinsic theoretical interest, but also because these facts underly the transformations of proofs that we propose. Appendix A.1 gives proofs of elementary algebraic properties: $(\mathscr{C}_\circ^\top, \odot)$ is a commutative magma (i.e. a set with a closed, total, binary operation) in which $\bot$ is an identity element and $\top$ is a zero. (Note that it is precisely because $\top$ is a zero that tautologies can be deleted during resolution proof search.) The following subsections establish weak forms of distributivity and associativity.

3.2.1 Distributivity

It is easy to see (Example 3.4) that $\odot$ is not distributive: in general, it is not the case that

$$(\kappa \odot \kappa_1) \odot (\kappa \odot \kappa_2) \equiv \kappa \odot (\kappa_1 \odot \kappa_2).$$

However, Theorem 3.5 shows that the non-distributivity is confined to cases in which subsumed clauses are derived. Hence, as long as the proofs and traces are subsumption-free (and this is usually the case in practice, because provers implement forward subsumption and tautology deletion), $\odot$ is essentially distributive.

Example 3.4. Consider the clauses $\kappa \doteq \{a, b\}$, $\kappa_1 \doteq \{\neg a, \neg b\}$ and $\kappa_2 \doteq \{b\}$. It is easy to check that $\langle (\kappa \odot \kappa_1) \odot (\kappa \odot \kappa_2) \rangle = \top$ but $\langle \kappa \odot (\kappa_1 \odot \kappa_2) \rangle = \{b\}$.

Theorem 3.5 (Subsumption-Free Distributivity)**.** $\odot$ is distributive for subsumption-free traces: for all clauses $\kappa, \kappa_1, \kappa_2 \in \mathscr{C}_\circ^\top$, if $(\kappa \odot \kappa_1) \odot (\kappa \odot \kappa_2)$ and $\kappa \odot (\kappa_1 \odot \kappa_2)$ are subsumption-free traces, then

$$(\kappa \odot \kappa_1) \odot (\kappa \odot \kappa_2) \equiv \kappa \odot (\kappa_1 \odot \kappa_2).$$

Proof. See Appendix A.2. □

Weak distributivity in the sense of Thm. 3.5 is immediately useful for compressing traces and their corresponding proofs. Whenever a subsumption-free trace contains a subtrace $(\kappa \odot \kappa_1) \odot (\kappa \odot \kappa_2)$, this subtrace can be replaced by the shorter and smaller evaluation equivalent trace $\kappa \odot (\kappa_1 \odot \kappa_2)$. Proofs could be preprocessed so that subtraces of this form occur more frequently, by swapping resolution inferences whose order does not matter. In fact, we will show in Section 4 that this idea can be generalized.

3.2.2 Associativity

Just as $\odot$ is not distributive in general, it is not associative, as shown in Example 3.6 below. Identifying cases where $\odot$ is nevertheless associative helps us to reduce proof length: the individual binary resolution steps can be performed in any order and can therefore be replaced by a single multi-clause resolution.

Example 3.6. Consider clauses $\kappa_1 \doteq \{a,b\}$, $\kappa_2 \doteq \{\neg a\}$, $\kappa_3 \doteq \{\neg b\}$. Then $\langle(\kappa_1 \odot \kappa_2) \odot \kappa_3\rangle = \bot$ whereas $\langle \kappa_1 \odot (\kappa_2 \odot \kappa_3)\rangle = \top$.[2]

However, note that using the commutativity of $\odot$, we could permute κ_1 and κ_2 in the previous example, thus obtaining the trace $(\kappa_2 \odot \kappa_1) \odot \kappa_3$, which is evaluation equivalent to $\kappa_2 \odot (\kappa_1 \odot \kappa_3)$. So, for κ_2, κ_1 and κ_3 in this order, $\odot$ behaves associatively. This observation raises the question of when it is possible, by exploiting commutativity, to permute the clauses in a way that $\odot$ becomes associative. This question is partially answered below: Def. 3.7 gives a sufficient condition for $\odot$ to be associative, as shown in Thm. 3.9 below.

Definition 3.7 (Chain Associability)**.** The clauses $\kappa_1, \ldots, \kappa_n$ are *chain associable* with respect to a permutation function π if and only if

$$|\kappa_{\pi(i)} \cap \overline{\kappa_{\pi(j)}}| = \begin{cases} 1 & \text{if } j = i+1 \text{ or } j = i-1 \\ 0 & \text{otherwise} \end{cases}$$

Clauses $\kappa_1, \ldots, \kappa_n$ are chain-associable if and only if there are two clauses that have subsumption-free resolvents with only one other clause each, and each of the remaining $n-2$ clauses has subsumption-free resolvents with exactly two other clauses. This motivates the name "chain", since the clauses can be permuted in an order that resembles a chain where every clause is adjacent to the clauses with which it has (subsumption-free) resolvents.

Example 3.8. The clauses $\kappa_1 \doteq \{a,b\}$, $\kappa_2 \doteq \{a,\neg b\}$ and $\kappa_3 \doteq \{\neg a\}$ are not chain associable. Every clause has subsumption-free resolvents with both other clauses. Also, for any permutation π,

$$(\kappa_{\pi(1)} \odot \kappa_{\pi(2)}) \odot \kappa_{\pi(3)} \not\equiv \kappa_{\pi(1)} \odot (\kappa_{\pi(2)} \odot \kappa_{\pi(3)}),$$

because in one of these traces, an implicit factoring of the two occurrences of the literal a occurs, and hence the conclusion-clause is $\bot$, while in the other trace, factoring does not occur, and hence the literal a occurs in the conclusion clause. This shows that the possibility of implicit factoring may invalidate chain associability; conversely, chain associability ensures that implicit factoring does not occur.

Theorem 3.9 (Chain Associativity)**.** Let $\kappa_1, \ldots, \kappa_n$ be the clauses of a tree-like trace t. If they are chain associable with respect to a permutation π, then:

$$t \equiv \kappa_{\pi(1)} \odot \ldots \odot \kappa_{\pi(n)}$$

where parentheses have been omitted to emphasize that $\odot$ is associative for the sequence of clauses $\kappa_{\pi(1)}, \ldots, \kappa_{\pi(n)}$.

[2] In this example, $\kappa_1 \odot (\kappa_2 \odot \kappa_3)$ is not defined according to the standard (partial) definition of resolution, but it is easy to find examples of non-associativity for standard resolution.

Proof sketch. Since $\kappa_1, \ldots, \kappa_n$ are chain associable, no implicit factoring can occur in $\kappa_{\pi(1)} \odot \ldots \odot \kappa_{\pi(n)}$, independently of the order in which the $\odot$ operations are evaluated. In particular, no implicit factoring can have occurred in t, and hence $t \equiv \kappa_{\pi(1)} \odot \ldots \odot \kappa_{\pi(n)}$ for any parenthesis structure. □

Intuitively, Thm. 3.9 says that for chain associable clauses, no particular sequential order has to be respected. Definition 3.10 defines a more general form of resolution that resolves n premises simultaneously in parallel. Observe that the chain resolution rule defined here is different from the multi-premise resolution rule frequently employed in proofs output by SAT- or SMT-solvers, which enforces left associativity (i.e., premises are resolved sequentially in the given order). In the chain resolution rule defined here, premises can be resolved with their neighbours in any order.

Definition 3.10 (Chain Resolution)**.** A *chain resolution* inference is an instance of the following rule:

$$\frac{\kappa_1 \quad \ldots \quad \kappa_n}{\kappa}\ \tilde{\mathbf{r}}$$

where $\kappa_1, \ldots, \kappa_n$ are chain associable with respect to the identity permutation and $\kappa \doteq \langle \kappa_1 \odot \ldots \odot \kappa_n \rangle$ is called the *chain resolvent* of $\kappa_1, \ldots, \kappa_n$. By Theorem 3.9, κ is uniquely defined. Moreover, it can be easily computed as: $(\kappa_1 \cup \ldots \cup \kappa_n) \setminus \{\ell_1, \overline{\ell_1}, \ldots, \ell_{n-1}, \overline{\ell_{n-1}}\}$, where ℓ_i is the single element of $\kappa_i \cap \overline{\kappa_{i+1}}$. We denote by $\ominus$ the n-ary algebraic operation corresponding to chain resolution and from now on allow traces to contain $\ominus$ gates as well as $\odot$ gates. □

The property of chain associativity immediately yields a way for shortening traces: given a trace t, replace any tree-like subtrace t_s whose clauses $\kappa_1, \ldots, \kappa_n$ are chain associable with respect to a permutation π by the subtrace $\ominus(\kappa_{\pi(1)}, \ldots, \kappa_{\pi(n)})$. In this way $n-1$ gates (inferences) are replaced by a single gate (inference), thus resulting in a reduction in the length of the trace (proof). For an estimate of how much compression can be achieved in this way, note that for tree-like traces without implicit factoring, any subtrace with only three clauses and two resolution gates is such that the three clauses are chain associable. We therefore obtain at least 50% reduction in the length in this case. Again, proofs can be preprocessed by swapping resolution inferences whose order is irrelevant, in order to maximize the number of subtraces with associable clauses.

3.2.3 Star Resolution

The chain resolution rule combines several resolution steps based on associativity of $\odot$. However, resolution steps can sometimes be combined even in the absence of associativity. In particular, a certain clause may be resolvable (without introducing subsumption) with more than two clauses, ruling out chain resolution. However, we may have a star-like configuration, with the distinguished clause in the center.

Definition 3.11 (Star Resolvability)**.** The clauses $\kappa_1, \ldots, \kappa_n$ are *star resolvable* if and only if there is a single *core clause* κ_i such that, for every other clause κ_j, $|\kappa_i \cap \overline{\kappa_j}| = \{\ell_j\}$, where each resolved literal ℓ_j is distinct (i.e. $\ell_{j_1} \neq \ell_{j_2}$, for $j_1 \neq j_2$) and does not occur in any non-core clauses (i.e. $\ell_j \notin \kappa_m$, for $m \neq i$); and $|\kappa_l \cap \overline{\kappa_m}| = \emptyset$ for every pair of non-core clauses. □

Definition 3.12 (Star Resolution)**.** A *star resolution* inference is an instance of the following rule:

$$\frac{\kappa_1 \quad \ldots \quad \kappa_n}{\kappa}\ \mathbf{r}^{\star}$$

where $\kappa_1, \ldots, \kappa_n$ are star resolvable clauses with core clause κ_1 and $\kappa \doteq \langle((\ldots(\kappa_1 \odot \kappa_2) \odot \ldots) \odot \kappa_n)\rangle$ is called *star resolvent* of $\kappa_1, \ldots, \kappa_n$. We denote by $\circledast$ the n-ary algebraic operation corresponding to star resolution and also allow $\circledast$ gates to occur in traces. □

Star resolution can be used for shortening traces and their corresponding proofs in a way analogous to the use of chain resolution: given a trace t, replace any tree-like subtrace t_s whose clauses $\kappa_1, \ldots, \kappa_n$ are star resolvable with core clause κ_i by the subtrace $\circledast(\kappa_i; \kappa_1, \ldots, \kappa_{i-1}, \kappa_{i+1}, \ldots \kappa_n)$. In this way $n-1$ gates (inferences) are replaced by a single gate (inference). For an estimate of how much compression can be achieved, note that for tree-like traces without implicit factoring, any subtrace with only four clauses and three resolution gates is such that the four clauses are either chain associable or star resolvable. Depending on the case, we can replace the subtrace by a single chain or star resolution gate. Hence, at least 66% reduction in trace length is possible in this case.

4 Resolution Hypergraphs

We have mentioned in sections 3.2.1 and 3.2.2 that it can be useful to swap resolution inferences in order to obtain subtraces for which chain resolution, star resolution or distributivity can best be exploited. Essentially, resolution proofs and traces impose a sequential order on the inferences, and resolution inferences can often be rearranged. Instead of attempting to heuristically find an optimal rearrangement of resolutions, we propose in this section a representation of proofs and traces as a (hyper-)graph that abstracts away the irrelevant and inessential details of the order of inferences.

The basic idea is to map a proof to a graph whose nodes and edges correspond to the axiom clauses and the inferences of the proof, respectively. Edges connect the nodes containing ancestors of the resolved literal. Due to factoring, the resolved literal can actually have ancestors in more than two axiom clauses, and we therefore need hyperedges (and hence hypergraphs) instead of simple edges and graphs. The basic graph structure does not impose any order on the inferences represented by hyperedges. For cases in which the order of inferences does matter, we augment the hypergraph by a strict partial order on its set of edges. In this section, we assume that proofs are subsumption-free.

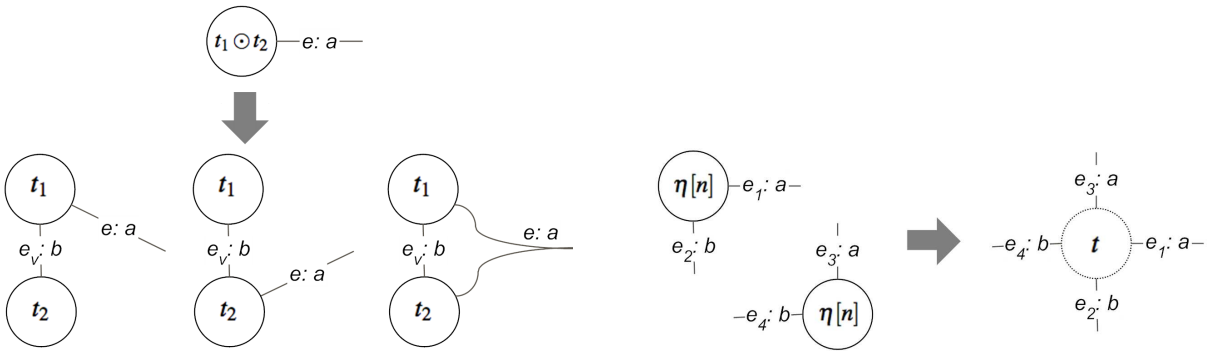
Definition 4.1 (Resolution Hypergraph)**.** A *resolution hypergraph* is a tuple $(V, M, \beta_v, E, \beta_e, \prec)$ where V is a set of *nodes*; $\beta_v : V \to \mathscr{T}$ is a *node labeling function* mapping nodes to traces; E is a multiset of *hyperedges*, which are sets of nodes; $\beta_e : E \to \mathscr{L}^+$ is an *edge labeling function* mapping edges to their resolved atoms; $\prec$ is a strict partial order over E; $M \subseteq V$ is the set of *merged nodes*. The set of all resolution hypergraphs is denoted $\mathscr{G}$. □

4.1 Constructing Hypergraphs from Proofs

Given a trace (or its underlying proof), we construct the corresponding hypergraph by graph rewriting. The initial hypergraph has a single node corresponding to the entire trace. We then repeatedly split or merge nodes until a normal form is reached. In this section we give only an informal and simplified graphical explanation of the graph rewriting rules; precise technical definitions can be found in Appendix B.

Definition 4.2 (Initial hypergraph of a trace)**.** The *initial hypergraph* of a trace t is the hypergraph $(\{v\}, \emptyset, \beta_v, \emptyset, \beta_e, \prec)$ with $\beta_v(v) \doteq t$. The initial hypergraph of a proof ψ is the initial hypergraph of $\mathbf{T}(\psi)$.

Definition 4.3 (Node splitting)**.** A subgraph with a node v labeled by a trace of the form $t_1 \odot t_2$, as shown in Fig. 1(a), can be rewritten to a graph having two new nodes for t_1 and t_2. These two nodes are connected by a new edge e_v with $\beta_e(e_v) = b$, where b is the atom resolved in the inference $t_1 \odot t_2$. Moreover, for any edge (with resolved atom a) originally connected to node v, there are three possibilities depending on whether a (or $\neg a$) belongs only to $\langle t_1 \rangle$, only to $\langle t_2 \rangle$ or to both. (In the latter case the edge e becomes a hyperedge connected to both new nodes.) These three possibilities are shown in the bottom



(a) Rewrite rule for node splitting (Def. 4.3). (b) Rewrite rule for node merging (Def. 4.4).

Figure 1: Illustrations of the rewrite rules for node splitting and merging.

part of Fig. 1(a). The partial order $\prec$ is extended such that $e_v \prec e'$ for every edge e' with $e \prec e'$. Moreover, if v was a merged node, then we add the constraints $e_v \prec e$ for all edges e connected to v. □

Rule 4.3 generalizes in the obvious way to chain resolution gates: a n-ary chain resolution gate can be regarded as $n-1$ binary resolution gates whose evaluation order does not matter. The expansion of an n-ary chain resolution gate results in a chain with n connected nodes.

Definition 4.4 (Node merging)**.** A subgraph having m nodes that contain trace links to a subtrace t with name n and outdegree m can be rewritten to a graph in which all m nodes are merged into a single node containing t and belonging to all edges to which the m nodes belonged. This rewriting rule is shown graphically in Fig. 1(b) for the case when $m = 2$ and the two nodes have two edges each. The new node is marked as a merged node. □

Theorem 4.5. The hypergraph rewriting system defined above is terminating and confluent.

Proof sketch. The number of node expansions is bounded by the number of resolution inferences in the proof. The number of node mergings is bounded by the number of gates with outdegree greater than one. Hence, the rewrite system is terminating. It is easy to see that, independently of the strategy used to apply the rewriting rules, the normal form will contain a node for each axiom clause of the trace and an edge for each resolution gate of the trace. Moreover, the strict partial order also depends only on the trace and not on the reduction strategy: $e \prec e'$ if and only if $\odot_e$ occurs in a subtrace t with outdegree greater than one and the root gate $\odot_t$ of t occurs above the gate $\odot_{e'}$ corresponding to e' (i.e. there is an upward path connecting $\odot_{e'}$ with $\odot_t$). This proves the confluence of the rewrite system. □

Termination and confluence of the rewriting system implies that we can define a function $\mathbf{G}$ from subsumption-free traces or proofs to the unique resolution hypergraph that results from the initial hypergraph by rewriting. It is easy to see that $\mathbf{G}$ is neither surjective (Theorem C.2) nor injective (Theorem C.1). In particular, resolution proofs that differ from one another only in inessential ways (such as immaterial sequentialization of inferences) are mapped to the same hypergraph.

Example 4.6. Let ψ be the proof shown in Example 2.3. Its resolution hypergraph $\mathbf{G}(\psi)$ appears in Fig. 2. The strict partial order is such that $e_i \prec e_j$ for $i \in \{8,9,10,11\}$ and $j \in \{3,4\}$. Note that each edge e_i has a corresponding resolution inference $\mathbf{r}_i$ in ψ.

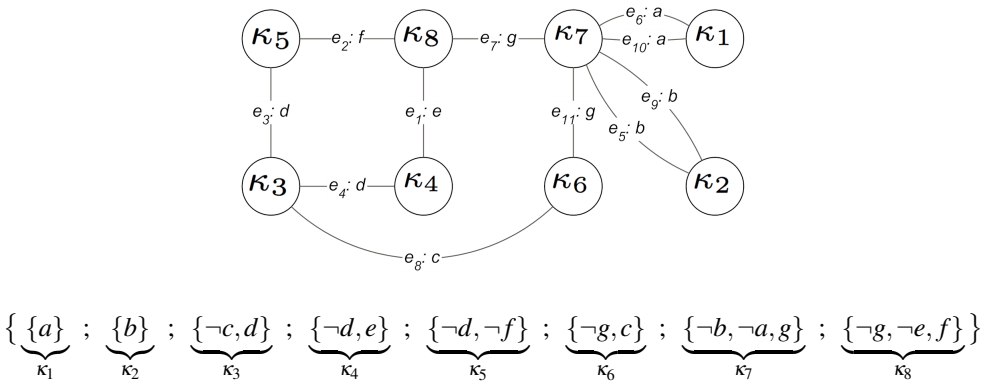


Figure 2: Resolution hypergraph for the running example.

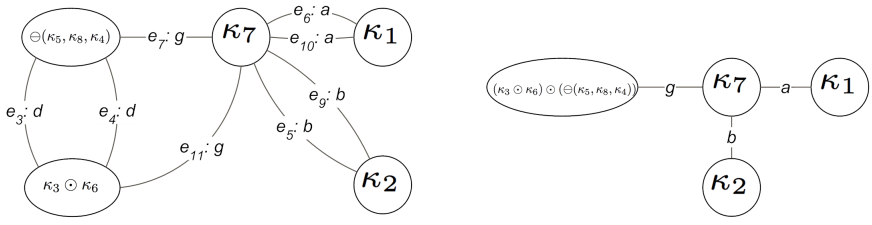
4.2 Simplifying Hypergraphs and Reconstructing Proofs

Once we have constructed the hypergraph representing the original proof (trace), we can use it to obtain an evaluation equivalent, but hopefully shorter proof (trace) that is represented by the same hypergraph. Intuitively, we do so by "applying the rewriting rules of section 4.1 backwards".[3] Aiming at some shortest proof corresponding to the given hypergraph, we can greedily search for the longest factoring-free chains or the largest star-shaped subgraphs in the hypergraph. The clauses in such factoring-free chains are chain associable, and hence reverse node expansion can easily transform such a chain into a single node containing a trace with a chain-resolution gate. The clauses in star-shaped subgraphs are star resolvable, and the subgraph can analogously be rewritten to a single node containing a trace with a star resolution gate.

While constructing hypergraphs or reconstructing proofs, we can also attempt to use the following simplifying graph rewriting rule, which is a more general way of exploiting distributivity.

Definition 4.7 (Edge merging)**.** If a subgraph contains two equal edges, then these edges can be replaced by a single edge. □

Example 4.8. The hypergraph that appears below on the left-hand side has been obtained from the hypergraph in Fig. 2 by applying two reconstruction steps, including the reverse node splitting that transformed the chain between κ_5, κ_8 and κ_4 into a single node with a trace having a chain-resolution gate. Subsequently, the graph on the right-hand side is obtained by performing four edge mergings and one more reverse node splitting.



[3] When applying node expansion in the "backward" direction, the strict partial order has to be respected: the backward node splitting can only be applied if the edge that disappears during the rewriting is a minimal edge with respect to the partial order. Similarly, when undoing node merging, we must choose how to distribute the edges of the merged node among its several copies. How this is done is important only if these edges are related by the partial order. In this case, an implementation should do some bookkeeping during the hypergraph construction such that the correct distribution can be computed efficiently.

A final reverse node splitting transforms the star-shaped graph on the right into a graph containing the following trace (a star resolution gate with κ_7 as its core clause):

$$t_{\psi^*} = \circledast(\kappa_7; \kappa_1, \kappa_2, (\kappa_3 \odot \kappa_6) \odot (\ominus(\kappa_5, \kappa_8, \kappa_4)))$$

The refutation ψ^* corresponding to the trace above is:

$$\dfrac{\dfrac{\dfrac{\neg b, \neg a, g \qquad a \qquad b}{\bot}\ \mathbf{r^\star} \qquad \dfrac{\neg c, d \qquad \neg g, c}{d, \neg g}\ \mathbf{r}}{\neg g}\ \mathbf{r} \qquad \dfrac{\neg d, \neg f \qquad \neg g, \neg e, f \qquad \neg d, e}{\neg d, \neg g}\ \tilde{\mathbf{r}}}{\neg g}\ \mathbf{r}$$

The final proof ψ^* has length (number of inferences) 4 and size (number of literals) 21. It is significantly shorter and smaller than the initial proof ψ, whose length and size are, respectively, 11 and 35.

5 Conclusions

We have presented work in progress towards a novel approach for transforming resolution proofs that are produced by SAT- and SMT-solvers, with the goal of obtaining shorter proofs. Based on an algebraic view of proofs and proof traces (developed in sections 2 and 3) we introduced our main concept of resolution hypergraphs in section 4. Given a proof trace, its hypergraph representation can be constructed in time and space linear in the length of the trace. We believe that this representation is interesting because it helps us to study the essence of a proof, identifying accidental sequentializations in the original proof trace and indicating opportunities for simplification that arise from reordering resolution steps.

The techniques that exploit chain and star resolution are straightforward to implement. In the case of tree-like factoring-free proofs, we can provably expect a compression of 66% by using chain resolution and star resolution alone. Using chain resolution only (for the same type of proofs), we can expect a compression of 50%. In practice, proofs are represented as dags and involve factoring, and we expect that the degree of reduction varies. Experimentation is therefore required to validate the effectiveness of our techniques, in particular within proof assistants.

The use of distributivity corresponds to merging edges in hypergraphs, and this is another promising direction for proof compression, but which remains to be studied in more detail. Indeed, edge merging may result in hypergraphs that do no longer correspond to a proof. Further work is required to identify sufficient conditions that allow the safe use of edge merging.

Algebraic properties of resolution have already been found important for computing interpolants [13], or for finding unsatisfiable cores (see for instance [7]), which are two side applications of proof production for the SAT or SMT problem. We intend to explore if resolution hypergraphs may contribute to those two applications.

References

[1] H. Amjad. Compressing propositional refutations. In *Sixth International Workshop on Automated Verification of Critical Systems (AVOCS '06) – Preliminary Proceedings*, pages 7–18, 2006.

[2] H. Amjad. Data compression for proof replay. *J. Autom. Reasoning*, 41(3-4):193–218, 2008.

[3] O. Bar-Ilan, O. Fuhrmann, S. Hoory, O. Shacham, and O. Strichman. Linear-time reductions of resolution proofs. In H. Chockler and A. J. Hu, editors, *HVC '08: 4th Intl. Haifa Verification Conf. on Hardware and Software*, volume 5394 of *Lecture Notes in Computer Science*, pages 114–128, Haifa, Israel, 2009. Springer.

[4] S. Böhme and T. Weber. Fast LCF-style proof reconstruction for Z3. In M. Kaufmann and L. Paulson, editors, *Interactive Theorem Proving (ITP 2010)*, Lecture Notes in Computer Science, Edinburgh, UK, 2010. Springer. To appear.

[5] T. Bouton, D. Caminha B. de Oliveira, D. Deharbe, and P. Fontaine. veriT: an open, trustable and efficient SMT-solver. In R. A. Schmidt, editor, *22nd Intl. Conf. Automated Deduction (CADE-22)*, 2009.

[6] A. Ciabattoni and A. Leitsch. Towards an algorithmic construction of cut-elimination procedures. *Mathematical Structures in Computer Science*, 2008.

[7] A. Cimatti, A. Griggio, and R. Sebastiani. A simple and flexible way of computing small unsatisfiable cores in SAT modulo theories. In J. Marques-Silva and K. A. Sakallah, editors, *Theory and Applications of Satisfiability Testing (SAT)*, volume 4501 of *Lecture Notes in Computer Science*, pages 334–339. Springer, 2007.

[8] S. Cotton. Some techniques for minimizing resolution proofs. In *Theory and Applications of Satisfiability Testing (SAT)*, Lecture Notes in Computer Science, Edinburgh, UK, 2010. Springer. To appear.

[9] P. De Groote, editor. *The Curry-Howard Isomorphism*. Université catholique de Louvain, 1995.

[10] P. Fontaine, J.-Y. Marion, S. Merz, L. P. Nieto, and A. Tiu. Expressiveness + automation + soundness: Towards combining SMT solvers and interactive proof assistants. In H. Hermanns and J. Palsberg, editors, *12th Intl. Conf. Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2006)*, volume 3920 of *Lecture Notes in Computer Science*, pages 167–181. Springer, 2006.

[11] J.-Y. Girard. Proof-nets: The parallel syntax for proof-theory. In P. Agliano and A. Ursini, editors, *Logic and Algebra*, 1996.

[12] A. Guglielmi and T. Gundersen. Normalisation control in deep inference via atomic flows. *Logical Methods in Computer Science*, 4(1:9):1–36, 2008.

[13] R. Jhala and K. L. McMillan. Interpolant-based transition relation approximation. *Logical Methods in Computer Science*, 3(4), 2007.

[14] A. Leitsch. *The resolution calculus*. Springer-Verlag, New York, NY, USA, 1997.

[15] M. Moskal. Rocket-fast proof checking for SMT solvers. In C. R. Ramakrishnan and J. Rehof, editors, *Tools and Algorithms for Construction and Analysis of Systems (TACAS)*, volume 4963 of *Lecture Notes in Computer Science*, pages 486–500. Springer, 2008.

[16] D. Oe, A. Reynolds, and A. Stump. Fast and flexible proof checking for SMT. In *7th Intl. Workshop Satisfiability Modulo Theories (SMT 2009)*, pages 14–29, 2009.

A Resolution Traces

Theorem A.1 (Closure of $\mathscr{C}_\circ^\top$ and Uniqueness of Resolvents)**.** Let $\kappa_1, \kappa_2 \in \mathscr{C}_\circ^\top$. Then there is a unique resolvent κ of κ_1 and κ_2, and $\kappa \in \mathscr{C}_\circ^\top$.

Proof. A resolvent κ exists because the definition by cases in Definition 2.1 exhausts all possible cases for κ_1 and κ_2. Moreover, κ is unique, because the definition by cases is deterministic. □

Definition A.2 (Representing Proofs by Traces)**.** The function $\mathbf{T} : \mathscr{P} \to \mathscr{T}$ is defined inductively:

- If ψ contains only an axiom clause κ, then $\mathbf{T}(\psi) \doteq \kappa$.
- If ψ is a proof-link to a proof named φ (i.e. $\psi = \eta[\varphi]$), then $\mathbf{T}(\psi) \doteq \eta[t_\varphi]$, where t_φ is the name of $\mathbf{T}(\varphi)$.
- If ψ ends in a resolution inference with subproofs ψ_1 and ψ_2 then $\mathbf{T}(\psi) \doteq (\mathbf{T}(\psi_1) \odot \mathbf{T}(\psi_2))$.

Theorem A.3. $\mathbf{T}$ is an isomorphism.

Proof. By definition A.2, it is clear that $\mathbf{T}$ is a homomorphism.

It remains to show that $\mathbf{T}$ is bijective. For surjectivity, let t be an arbitrary trace in $\mathscr{T}$. Construct a proof ψ_t whose clauses are the clauses of t and whose inferences structurally correspond to the occurrences of $\odot$ in t. These inferences are correct and well-defined, due to the closure of $\mathscr{C}_\circ^\top$ under $\mathbf{r}$ and the uniqueness of resolvents (Theorem A.1). Moreover, it is easy to see that, by Definition A.2, $\mathbf{T}(\psi_t) = t$.

For injectivity, let ψ_1 and ψ_2 be proofs such that $\mathbf{T}(\psi_1) = \mathbf{T}(\psi_2) = t$. By definition A.2, ψ_1 and ψ_2 must have the same skeleton of inferences and the same axiom clauses. Theorem A.1 guarantees that the resolvents are uniquely defined by the premises and the inferences. Therefore, $\psi_1 = \psi_2$. □

A.1 Algebraic Properties of Traces

Theorem A.4. $(\mathscr{C}_\circ^\top, \odot)$ is a magma.

Proof. $\mathscr{C}_\circ^\top$ is closed under $\odot$, since $\mathscr{C}_\circ^\top$ is closed under $\mathbf{r}$ (Theorem A.1) and $\odot$ corresponds to $\mathbf{r}$ by the isomorphism $\mathbf{T}$ (Theorem A.3). □

Theorem A.5. $\odot$ is commutative.

Proof. It is easy to observe in Definition 2.1 that the resolvent of κ_1 and κ_2 is equal to the resolvent of κ_2 and κ_1. The resolvent does not depend on the order of the premises. By the fact that $\mathbf{T}$ is an isomorphism (Theorem A.3), it follows that $\kappa_1 \odot \kappa_2 \equiv \kappa_2 \odot \kappa_1$. □

Theorem A.6 (Identity)**.** $\bot$ is an identity element for $\odot$: for all clauses κ,

$$\kappa \odot \bot \equiv \kappa$$

Theorem A.7 (Zero)**.** $\top$ is a zero element for $\odot$: for all clauses $\kappa \in \mathscr{C}_\circ^\top$,

$$\kappa \odot \top \equiv \top$$

Theorem A.8. Only unit clauses and $\bot$ have inverses with respect to $\odot$: if $\kappa \odot \kappa' = \bot$ then $\kappa = \kappa' = \bot$ or $\kappa = \{\ell\}$ and $\kappa' = \{\overline{\ell}\}$ for some literal ℓ.

A.2 Distributivity

Theorem A.9 (Subsumption-Free Distributivity)**.** $\odot$ is distributive for subsumption-free traces: for all clauses $\kappa, \kappa_1, \kappa_2 \in \mathscr{C}_\circ^\top$, if $(\kappa \odot \kappa_1) \odot (\kappa \odot \kappa_2)$ and $\kappa \odot (\kappa_1 \odot \kappa_2)$ are subsumption-free traces, then

$$(\kappa \odot \kappa_1) \odot (\kappa \odot \kappa_2) \equiv \kappa \odot (\kappa_1 \odot \kappa_2)$$

Proof sketch. In order to be able to refer to each $\odot$ symbol, let them be indexed as follows:

$$(\kappa \odot_1 \kappa_1) \odot_2 (\kappa \odot_3 \kappa_2) \equiv \kappa \odot_4 (\kappa_1 \odot_5 \kappa_2)$$

Since the traces are subsumption-free, each $\odot_i$ has a resolved atom ℓ_i. We show that $\ell_1 = \ell_3$: assume the contrary for the sake of contradiction. Assume w.l.o.g. (since the other cases are analogous) that $\ell_1, \ell_3 \in \kappa$, $\neg\ell_1 \in \kappa_1$ and $\neg\ell_3 \in \kappa_2$. Then there are three cases for ℓ_5:

- $\ell_5 = \ell_1$: then $\ell_1 \in \kappa_2$. We consider two sub-cases.
  - $\ell_2 \notin \kappa$: in this case, either $\ell_2 \in \kappa_1$ and $\neg\ell_2 \in \kappa_2$ (and hence $\{\ell_1, \neg\ell_2\} \subseteq \overline{\kappa_1} \cap \kappa_2$) or $\ell_2 \in \kappa_2$ and $\neg\ell_2 \in \kappa_1$ (and hence $\{\ell_1, \ell_2\} \subseteq \overline{\kappa_1} \cap \kappa_2$). In either case, $|\overline{\kappa_1} \cap \kappa_2| > 1$, and therefore $\odot_5$ has no resolved atom, $\langle \kappa_1 \odot_5 \kappa_2 \rangle = \top$ and $\kappa \odot_4 (\kappa_1 \odot_5 \kappa_2)$ is not subsumption-free. Contradiction!
  - $\ell_2 \in \kappa$: in this case, either $\neg\ell_2 \in \kappa_1$ (and hence $\langle \kappa \odot_1 \kappa_1 \rangle = \top$) or $\neg\ell_2 \in \kappa_2$ (and hence $\langle \kappa \odot_3 \kappa_2 \rangle = \top$). In either case, $(\kappa \odot_1 \kappa_1) \odot_2 (\kappa \odot_3 \kappa_2)$ is not subsumption-free. Contradiction!
- $\ell_5 = \ell_3$: analogous to the case above.

- $\ell_5 \neq \ell_1$ and $\ell_5 \neq \ell_3$: then $\neg\ell_1, \neg\ell_3 \in \langle \kappa_1 \odot_5 \kappa_2 \rangle$. Consequently, $\langle \kappa \odot_4 (\kappa_1 \odot_5 \kappa_2) \rangle = \top$ and the trace is not subsumption-free. Contradiction!

Since $\ell_1 = \ell_3$, it is easy to see that $\ell_4 = \ell_1 = \ell_3$. Assume w.l.o.g. that $\ell_5 \in \kappa_1$ and $\neg\ell_5 \in \kappa_2$. Then, since $(\kappa \odot_1 \kappa_1) \odot_2 (\kappa \odot_3 \kappa_2)$ is subsumption-free, it must be the case that $\ell_2 = \ell_5$ and $\ell_5 \notin \kappa$ and $\neg\ell_5 \notin \kappa$.

$$
\begin{array}{rcl}
\langle (\kappa \odot_1 \kappa_1) \odot_2 (\kappa \odot_3 \kappa_2) \rangle & = & (((\kappa \setminus \{\ell_1\}) \cup (\kappa_1 \setminus \{\neg\ell_1\})) \setminus \{\ell_2\}) \cup (((\kappa \setminus \{\ell_3\}) \cup (\kappa_2 \setminus \{\neg\ell_3\})) \setminus \{\neg\ell_2\}) \\
& = & (((\kappa \setminus \{\ell_1\}) \cup (\kappa_1 \setminus \{\neg\ell_1\})) \setminus \{\ell_5\}) \cup (((\kappa \setminus \{\ell_1\}) \cup (\kappa_2 \setminus \{\neg\ell_1\})) \setminus \{\neg\ell_5\}) \\
& = & ((\kappa \setminus \{\ell_1\}) \cup ((\kappa_1 \setminus \{\neg\ell_1\}) \setminus \{\ell_5\})) \cup ((\kappa \setminus \{\ell_1\}) \cup ((\kappa_2 \setminus \{\neg\ell_1\}) \setminus \{\neg\ell_5\})) \\
& = & (\kappa \setminus \{\ell_1\}) \cup ((\kappa_1 \setminus \{\neg\ell_1\}) \setminus \{\ell_5\}) \cup (\kappa \setminus \{\ell_1\}) \cup ((\kappa_2 \setminus \{\neg\ell_1\}) \setminus \{\neg\ell_5\}) \\
& = & (\kappa \setminus \{\ell_1\}) \cup ((\kappa_1 \setminus \{\neg\ell_1\}) \setminus \{\ell_5\}) \cup ((\kappa_2 \setminus \{\neg\ell_1\}) \setminus \{\neg\ell_5\}) \\
& = & (\kappa \setminus \{\ell_1\}) \cup ((\kappa_1 \setminus \{\ell_5\}) \setminus \{\neg\ell_1\}) \cup ((\kappa_2 \setminus \{\neg\ell_5\}) \setminus \{\neg\ell_1\}) \\
& = & (\kappa \setminus \{\ell_1\}) \cup (((\kappa_1 \setminus \{\ell_5\}) \cup (\kappa_2 \setminus \{\neg\ell_5\})) \setminus \{\neg\ell_1\}) \\
& = & (\kappa \setminus \{\ell_4\}) \cup (((\kappa_1 \setminus \{\ell_5\}) \cup (\kappa_2 \setminus \{\neg\ell_5\})) \setminus \{\neg\ell_4\}) \\
& = & \langle \kappa \odot_4 (\kappa_1 \odot_5 \kappa_2) \rangle
\end{array}
$$

B Precise Definitions for Resolution Hypergraph Rewriting

Definition B.1 (Graph rewriting rule for node splitting)**.** Let $G \doteq (V, M, E, \beta_v, E, \beta_e, \prec)$ be a resolution hypergraph with a node $v \in V$ such that $\beta_v(v) = t_1 \odot t_2$ for arbitrary traces t_1 and t_2. Then, by the *graph rewriting rule for node splitting*, it can be rewritten to the hypergraph $G' \doteq ((V \setminus \{v\}) \cup \{v_1, v_2\}, M \setminus \{v\}, \beta'_v, (E \setminus E_v) \cup E'_v \cup \{e_{v_1 v_2}\}, \beta'_e, \prec')$ where:

- $\beta'_v(x) \doteq \begin{cases} t_1 & \text{if } x = v_1 \\ t_2 & \text{if } x = v_2 \\ \beta_v(x) & \text{otherwise} \end{cases}$
- $e_{v_1 v_2} \doteq \{v_1, v_2\}$.
- E_v is the set of edges $e \in E$ such that $v \in e$, and $E'_v \doteq \varepsilon(E_v)$, where $\varepsilon : E \to (E \setminus E_v) \cup E'_v$ is the bijective function defined below (with either $\ell = \beta_e(e)$ or $\ell = \neg\beta_e(e)$):

$$
\varepsilon(e) \doteq \begin{cases} \{v_1, v_2\} \cup (e \setminus \{v\}) & \text{if } e \in E_v,\ \ell \in \langle t_1 \rangle, \text{ and } \ell \in \langle t_2 \rangle \\ \{v_1\} \cup (e \setminus \{v\}) & \text{if } e \in E_v,\ \ell \in \langle t_1 \rangle, \text{ and } \ell \notin \langle t_2 \rangle \\ \{v_2\} \cup (e \setminus \{v\}) & \text{if } e \in E_v,\ \ell \notin \langle t_1 \rangle, \text{ and } \ell \in \langle t_2 \rangle \\ e & \text{otherwise (i.e. if } e \notin E_v) \end{cases}
$$

- $\beta'_e(e) \doteq \begin{cases} \ell & \text{where } \ell \text{ is the resolved atom between } \langle t_1 \rangle \text{ and } \langle t_2 \rangle \text{ if } e = e_{v_1 v_2} \\ \beta_e(\varepsilon^{-1}(e)) & \text{otherwise} \end{cases}$
- $\prec'$ is defined as follows:
  - for all edges $e_1, e_2 \in (E \setminus E_v) \cup E'_v$, $e_1 \prec' e_2$ if and only if $\varepsilon^{-1}(e_1) \prec \varepsilon^{-1}(e_2)$.
  - for every edge $e \in (E \setminus E_v) \cup E'_v$, $e_{v_1 v_2} \prec' e$ if and only if either there exists an edge $e^* \in E_v$ such that $e^* \prec \varepsilon^{-1}(e)$ or $e \in E'_v$ and $v \in M$ (i.e. v is a merged node).
  - for every edge $e \in (E \setminus E_v) \cup E'_v$, $e \not\prec' e_{v_1 v_2}$.

Definition B.2 (Graph rewriting rule for node merging)**.** Let $G \doteq (V, M, E, \beta_v, E, \beta_e, \prec)$ be a resolution hypergraph with nodes $v_1, \ldots, v_m \in V$ such that $\beta_v(v_i) = \eta[n_t]$ where n_t links to a subtrace t whose last gate has outdegree m. Then, by the *graph rewriting rule for node merging*, it can be rewritten to the hypergraph $G' \doteq ((V \setminus \{v_1, \ldots, v_m\}) \cup \{v\}, M \cup \{v\}, \beta'_v, (E \setminus E_v) \cup E'_v), \beta'_e, \prec')$, where:

- For all nodes $x \in V'$:
$$\beta'_v(x) \doteq \begin{cases} t & \text{if } x = v \\ \beta_v(x) & \text{otherwise} \end{cases}$$

- E_v is the set of edges having non-empty intersection with $\{v_1, \ldots, v_m\}$ and $E'_v = \varepsilon(E_v)$, where $\varepsilon : E \to (E \setminus E_v) \cup E'_v$ is the bijective function defined below:
$$\varepsilon(e) \doteq \begin{cases} (e \setminus \{v_1, \ldots, v_m\}) \cup \{v\} & \text{if } e \in E_v \\ e & \text{otherwise (i.e. if } e \notin E_v) \end{cases}$$

- For all edges $e \in (E \setminus E_v) \cup E'_v$):
$$\beta'_e(e) \doteq \begin{cases} \beta_e(\varepsilon^{-1}(e)) & \text{if } e \in E'_v \\ \beta_e(e) & \text{otherwise} \end{cases}$$

- $\prec'$ is such that, for all edges $e_1, e_2 \in (E \setminus E_v) \cup E'_v$, $e_1 \prec' e_2$ if and only if $\varepsilon^{-1}(e_1) \prec \varepsilon^{-1}(e_2)$.

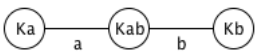
C Some Properties of Resolution Hypergraphs

Theorem C.1. **G** is not injective.

Proof sketch. Let $\kappa_a \doteq \{a\}$, $\kappa_b \doteq \{b\}$, $\kappa_{ab} \doteq \{\neg a, \neg b\}$ and let

$$\begin{array}{ll} t_1 \doteq (\kappa_a \odot \kappa_{ab}) \odot \kappa_b & t_2 \doteq \kappa_b \odot (\kappa_a \odot \kappa_{ab}) \\ t_3 \doteq (\kappa_{ab} \odot \kappa_a) \odot \kappa_b & t_4 \doteq \kappa_b \odot (\kappa_{ab} \odot \kappa_a) \\ t_5 \doteq (\kappa_b \odot \kappa_{ab}) \odot \kappa_a & t_6 \doteq \kappa_a \odot (\kappa_b \odot \kappa_{ab}) \\ t_7 \doteq (\kappa_{ab} \odot \kappa_b) \odot \kappa_a & t_8 \doteq \kappa_a \odot (\kappa_{ab} \odot \kappa_b) \end{array}$$

It is easy to see that $\mathbf{G}(t_1) = \mathbf{G}(t_2) = \mathbf{G}(t_3) = \mathbf{G}(t_4) = \mathbf{G}(t_5) = \mathbf{G}(t_6) = \mathbf{G}(t_7) = \mathbf{G}(t_8)$:



Theorem C.2. **G** is not surjective.

Proof sketch. The two resolution hypergraphs shown below are examples of hypergraphs that are not the image of any proper resolution proof. It is easy to verify that any attempt to construct a proof whose image is one of these graphs will result in a proof that is not proper. For the graph in the left, independently of which of the two resolution inferences (i.e. with resolved atom a or b) is performed first, the result is the tautological clause $\top$, and hence the proof is not proper. For the graph in the right (and for any cyclic graph with an analogous form), it is possible to avoid deriving the tautological clause, but then a non-tautological clause subsumed by one of the axiom clauses must be derived, and hence the proof is not proper.