Journal of Automated Reasoning manuscript No.
(will be inserted by the editor)

Conflict Resolution

a First-Order Resolution Calculus with Decision Literals and Conflict-Driven Clause Learning

John Slaney · Bruno Woltzenlogel Paleo

Received: date / Accepted: date

Abstract This paper defines the (first-order) conflict resolution calculus: an extension of the resolution calculus inspired by techniques used in modern SAT-solvers. The resolution inference rule is restricted to (first-order) unit propagation and the calculus is extended with a mechanism for assuming *decision literals* and with a new inference rule for *clause learning*, which is a first-order generalization of the propositional *conflict-driven clause learning* (CDCL) procedure. The calculus is sound (because it can be simulated by natural deduction) and refutationally complete (because it can simulate resolution), and these facts are proven in detail here.

Keywords Mathematical Logic · Automated Reasoning · Proof Theory · Resolution · Natural Deduction · Conflict-Driven Clause Learning

1 Introduction

Modern SAT-solvers are famously effective for solving the decision problem of satisfiability of propositional formulas, and we may wonder whether the ideas used in SAT-solvers could be generalized to the first-order case. This paper addresses this question from a purely proof-theoretical perspective.

We briefly recall the first-order resolution calculus (in Section 2), which is the theoretical foundation for many current state-of-the-art first-order theorem provers (e.g. [27,30,37]), and the DPLL and CDCL procedures used by SAT-solvers (in Section 3). The main contribution of this paper (presented in Section 4) is the *conflict resolution* calculus **CR**. It extends the first-order

J. Slaney
Australian National University
E-mail: John.Slaney@anu.edu.au

B. Woltzenlogel Paleo
Australian National University
E-mail: bruno.wp@gmail.com

resolution calculus with *decision literals* and a new inference rule for *clause learning* and restricts the resolution rule in order to force it to behave like *unit propagation*. As discussed in Subsection 4.1, a certain subclass of **CR** derivations is isomorphic to the abstract data structure known as *conflict graphs* or *implication graphs* and widely used to describe the procedures of modern SAT-solvers. Furthermore, as shown in Section 7, whereas the *splitting* technique used by modern first-order provers must either be handled at an extra-logical level or lead to an unacceptable increase in proof size if simulated in the resolution calculus, its simulation by **CR**'s decisions and clause learning is lean and straightforward. Therefore, the new **CR** calculus provides a more adequate proof-theoretical foundation for procedures currently implemented by SAT-solvers and first-order provers.

In **CR**, it becomes evident that decision literals are analogous to assumptions in natural deduction, whereas clause learning resembles natural deduction's implication introduction rule. This fact is crucial for the proof of soundness of **CR** (shown in Section 6) and it illustrates an insightful novelty of the calculus: while the resolution inference proposed by Robinson [28] can be regarded as a first-order generalization of modus ponens (a.k.a. natural deduction's implication *elimination*) by taking unification into account, the clause learning rule proposed here (and inspired by the propositional CDCL technique) can be considered a first-order generalization of implication *introduction*, as it discharges decision literals in a way that allows for unification.

Any resolution refutation can be translated into a refutation in **CR**. Therefore, **CR**'s refutational completeness follows easily from the refutational completeness of the resolution calculus (as proven in Section 5).

A main motivation for the development of the conflict resolution calculus was that it might eventually serve as a theoretical common ground for existing first-order provers that try to harness or mimic the power of SAT-solvers (cf. Section 9)) or as a starting point for the development of new provers, in the same way that the pure resolution calculus provided the basic foundation for several generations of automated theorem provers in the last decades. To achieve this goal, the calculus is presented in a general way, avoiding premature optimizations and refinements, so that future work may easily build on it and explore various proof search strategies and implementation techniques.

2 Recalling Resolution

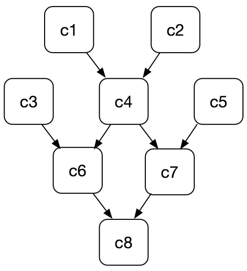
Clauses (denoted c, possibly subscripted) are disjunctions of literals. A *literal* is either an atom or a negated atom, and an *atom* is a n-ary predicate (denoted P or Q) applied to n terms. A *term* is either a constant (denoted a or b), a variable (denoted x, y, v or z) or an n-ary function (denoted f or g) applied to n terms. Variables in a clause are assumed to be implicitly universally quantified. A clause having a single literal is called *unit*. If ℓ is a literal, $\overline{\ell}$ denotes its dual (i.e. $\overline{P} = \neg P$ and $\overline{\neg P} = P$). The nullary atoms $\top$ (*verum*) and $\bot$ (*falsum*) have special meanings characterized by the following equations: $\Gamma \vee \bot = \Gamma$

and $\Gamma \vee \top = \top$. All inference rules operating on clauses are assumed to be modulo disjunction's associativity, commutativity and idempotence, modulo negation's involutivity and modulo the equations for $\top$ and $\bot$. The empty clause is logically equivalent to the clause containing only $\bot$. Therefore, slightly abusing notation, it is denoted by $\bot$. Substitutions (denoted by σ, possibly sub- and superscripted) are assumed to implicitly avoid variable capture. The empty (i.e. identity) substitution is denoted ε.

The inference rules of the resolution calculus are shown in Fig. 1. A resolution *proof* of a clause c from a set of clauses S is a directed acyclic graph (DAG) where leaves (i.e. input nodes) are clauses from S, internal nodes are obtained from their parents through application of the inference rules and the sink node is the clause c. A resolution *refutation* of a set of clauses S is a proof of the empty clause (denoted $\bot$) from S. It is assumed that distinct input clauses do not share variables and that the inference rules implicitly generate fresh symbols for variables.

Proof DAGs are sometimes displayed as a collection of trees through the following convention: nodes used as premises more than once are given names (e.g. φ, ψ or ξ) when they are used for the first time, and the names are used to refer to the nodes whenever they are used again. By naming and referring, wide proof trees can also be broken down in smaller displayable parts.

Example 1 Consider a proof with the following non-tree form:



It can be displayed as the single tree with names and references below, where the second (rightmost) occurrence of the name ψ is to be understood as a reference to the node named ψ by the first (leftmost) occurrence of ψ:

$$\dfrac{\dfrac{c_3 \qquad \dfrac{c_1 \qquad c_2}{\psi : c_4}}{c_6} \qquad \dfrac{\psi \qquad c_5}{c_7}}{c_8}$$

Or it can also be displayed as the following forest, where the two occurrences of the name ψ in the rightmost tree are to be understood as references to the node named ψ in the leftmost tree:

$$\dfrac{c_1 \qquad c_2}{\psi : c_4} \qquad\qquad \dfrac{\dfrac{c_3 \qquad \psi}{c_6} \qquad \dfrac{\psi \qquad c_5}{c_7}}{c_8}$$

Resolution:

$$\frac{\varGamma \vee \ell \quad \overline{\ell'} \vee \varDelta}{(\varGamma \vee \varDelta)\ \sigma}\ \mathbf{r}(\sigma)$$

where σ is a unifier of ℓ and ℓ'.

Factoring:

$$\frac{\ell \vee \ell' \vee \varGamma}{(\ell \vee \varGamma)\ \sigma}\ \mathbf{f}(\sigma)$$

where σ is a unifier of ℓ and ℓ'.

Fig. 1: Resolution Calculus

Given a set of clauses, a resolution prover exhaustively applies the inference rules, generating more and more clauses. If the initial clause set is unsatisfiable and a fair clause/rule selection strategy is used, the empty clause is eventually derived, because resolution is refutationally complete [28]. If the set is satisfiable, the prover will either never terminate or will terminate in a state where the set of initial and derived clauses is saturated with respect to redundancy criteria (i.e. only redundant clauses would still be derivable) (cf. [34]).

One practical problem in this saturation approach is the vast number of clauses that are generated. This led to research on refinements of the resolution calculus, aiming at restricting the inference rules in order to generate fewer clauses, and on efficient ways to detect and delete redundant (e.g. subsumed) clauses. These efforts culminated in the *superposition*[1] calculus [3,4, 33], which extends the resolution calculus with the equality resolution and paramodulation rules [29] for equality reasoning and refines them with ordering restrictions on terms and literals, preserving refutational completeness by adding either the equality factoring rule or the merging paramodulation and ordered factoring rules [33].

Another practical problem is that the resolvent of a clause with n literals and another clause with m literals has $n + m - 2$ literals. When iterated, this results in long clauses and, consequently, loss of efficiency. This practical problem has been solved with a technique known as *splitting* [35]: if the current set of clauses is $S \cup \{\varGamma_1 \vee \ldots \vee \varGamma_k\}$ and the sets of variables V_i of $\varGamma_i$ are mutually disjoint, we can split the clause $\varGamma_1 \vee \ldots \vee \varGamma_k$ into its variable-disjoint components and the clause set into the k sets $S \cup \{\varGamma_i\}$ (for $1 \leq i \leq k$). The disjointness ensures that we can check the unsatisfiability of each resulting clause set separately: $S \cup \{\varGamma_1 \vee \ldots \vee \varGamma_k\}$ is unsatisfiable iff $S \cup \{\varGamma_i\}$ is unsatisfiable for every variable-disjoint component $\varGamma_i$.

From a proof-theoretical perspective, splitting resembles the β-rule of free-variable tableaux [2,36]. Therefore, superposition provers that implement splitting [37,30,27] can be seen as hybrids combining resolution/superposition and

[1] **CR** is based on resolution instead of superposition, because superposition's ordering-based refinements would restrict unit-propagation and the selection of decision literals. In Sat-solvers unit-propagation is unrestricted (because it is very effective anyway) and the best literal selection strategies are not based on orderings. By extending unrestricted resolution, **CR** remains general enough to admit the strategies used by Sat-solvers.

tableaux; and hence they lack a theoretical characterization in terms of a single pure proof system. This gap between theory and practice is something that can be remedied with the adoption of the **CR** calculus (cf. Section 7).

3 Recalling DPLL and CDCL

In the propositional case, Davis, Logemann and Loveland [17] had already noticed that the propositional resolution rule [16] "can easily increase the number and the lengths of the clauses" and proposed to replace it by a form of splitting, which is, however, different from the later notion of splitting described in Section 2. Instead of splitting a clause into variable-disjoint components, we select a propositional atom P and split the problem in two subproblems: one where P is assumed to be true and the other where it is assumed to be false. Nowadays, the so-called DPLL procedure is presented slightly differently, but equivalently. We *decide* to assign the truth value `true` (or `false`) to an atom; then, through *unit propagation*, other atoms will be assigned truth values as well. Repeating this process of decisions and propagations, we will either reach an assignment that satisfies all clauses (if the clause set is satisfiable) or we will reach a *conflict* where we are to assign both `true` and `false` to an atom. In the latter case, we backtrack some of our decisions, and try different assignments.

In contrast to saturation-based theorem proving, DPLL-based sat-solving does not generate any clause at all. But this is, of course, dependent on the fact that in propositional logic it suffices to consider only two truth-value assignments for each atom. In a naïve adaptation of this idea to first-order logic, on the other hand, we would need to consider truth-value assignments for each instance of an atom containing variables. We would need to generate possibly several[2] instances.

In practice, it is, nevertheless, beneficial to generate *some* clauses when backtracking from conflicts. For example, suppose that the backtracking DPPL procedure decided to assign `true` to P and Q, and this led to a conflict. It is then forced to backtrack these decisions and try other decisions. Without clause learning, it could happen that, after assigning truth values to other atoms, it would again consider the possibility of assigning `true` to P and Q, even though it is clear (from the previous conflict) that P and Q cannot be both `true`, independently of later assignments to other atoms. To prevent this from happening, we can generate and add the clause $\neg P \vee \neg Q$ to the set of clauses. Then, whenever the procedure retries assigning, for instance, `true` to P it will immediately conclude (by unit propagation) that `false` should be assigned to Q. This idea is known as *conflict-driven clause learning.*

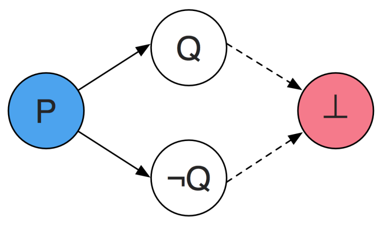
The procedure up to a conflict can be understood as the construction of a directed graph. Nodes are literals which have been assigned `true`. A *decision literal* (i.e. a literal with truth value assigned by decision) has no incoming

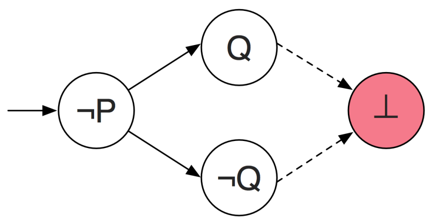
[2] By Herbrand's theorem, a finite number of instances would suffice in the case of an unsatisfiable clause set. As the logic is undecidable, however, there is no way to know in advance how many "several" might be.

edge. A *propagated literal* (i.e. a literal with truth value assigned by unit propagation) ℓ has incoming edges (ℓ_i, ℓ) for $0 < i \leq n$ if and only if the clause $\overline{\ell_1} \lor \ldots \lor \overline{\ell_n} \lor \ell$ was the clause used by unit propagation to assign a truth value to ℓ. A conflict is indicated by the simultaneous presence of any literal and its dual in the graph. When a conflict is detected, the graph can be analyzed to determine clauses that should be learned. Various conflict analysis algorithms exist [23,24]. The conceptually simplest one recommends learning a clause that is a disjunction of the negations of the decision literals. More sophisticated algorithms [39] are capable of learning stronger clauses. An important benefit of conflict-driven clause learning is that redundant (i.e. subsumed) clauses are never derived.

The learned clause can be derived by a sequence of resolution steps using the clauses corresponding to the edges in the graph as premises. When this is done, a SAT-solver is capable of outputting a propositional resolution refutation for an unsatisfiable clause set [9]. However, most developers of SAT-solvers consider the overhead (in both proving time and memory consumption) of doing so unacceptable, especially when in-processing techniques and advanced techniques for minimizing learned clauses are used. Instead, they prefer to generate proof certificates in the DRUP or DRAT formats [38], which record clauses that have been learned, but do not inform which premises are needed to derive them. A consequence of this lack of information is that checking a DRUP/DRAT certificate or converting it to a resolution refutation (using the DRAT-Trim tool) can take as long as solving the problem in the first place.

Example 2 Consider the clause set $\{P \lor Q,\ P \lor \neg Q,\ \neg P \lor Q,\ \neg P \lor \neg Q\}$. Deciding P and propagating units results in the conflict graph at the left side below. We backtrack and learn the unit clause $\neg P$, whose propagation leads to the conflict graph in the right side below. Since this last conflict does not depend on any decision literal, no backtracking is possible, and we may conclude that the clause set is unsatisfiable.





The resolution proofs extracted from these conflict graphs are shown below:

$$\cfrac{\cfrac{\neg P \lor Q \qquad \neg P \lor \neg Q}{\neg P \lor \neg P}}{\neg P} \qquad\qquad \cfrac{\cfrac{\cfrac{P \lor Q \qquad P \lor \neg Q}{P \lor P}}{P} \qquad \neg P}{\bot}$$

4 The Conflict Resolution Calculus

As we have seen in the previous two sections, both propositional and first-order automated deduction have progressed (in different ways) much beyond their historical common roots in resolution. Techniques such as splitting, conflict graphs and conflict-driven clause learning are not so easily explained in terms of a pure resolution calculus. There is a growing gap between the current state-of-the-art in automated deduction and its original proof-theoretical foundation. In this section, we propose the **CR** calculus, which modifies the first-order resolution calculus by incorporating ideas from Sat-solving, in an attempt to reduce not only the gap between automated deduction and proof theory but also between the first-order and the propositional cases.

As in resolution, a **CR** *derivation* is a directed acyclic graph where nodes are clauses and internal nodes are obtained from their parents by one of the inference rules shown in Fig. 2.

Unit-Propagating Resolution:

$$\frac{\ell_1 \quad \dots \quad \ell_n \quad \overline{\ell'_1} \vee \dots \vee \overline{\ell'_n} \vee \ell}{\ell\ \sigma}\ \mathbf{u}(\sigma)$$

where σ is a unifier of ℓ_k and ℓ'_k, for all $k \in \{1, \dots, n\}$.

Conflict:

$$\frac{\ell \quad \overline{\ell'}}{\bot}\ \mathbf{c}(\sigma)$$

where σ is a unifier of ℓ and ℓ'.

Conflict-Driven Clause Learning:

$$\frac{\begin{array}{ccc} [\ell_1]^{i}_{1} & & [\ell_n]^{i}_{n} \\ \vdots\ (\sigma^1_1, \dots, \sigma^1_{m_1}) & & \vdots\ (\sigma^n_1, \dots, \sigma^n_{m_n}) \\ & \vdots & \\ & \bot & \end{array}}{(\overline{\ell_1}\sigma^1_1 \vee \dots \vee \overline{\ell_1}\sigma^1_{m_1}) \vee \dots \vee (\overline{\ell_n}\sigma^n_1 \vee \dots \vee \overline{\ell_n}\sigma^n_{m_n})}\ \mathbf{cl}^i$$

where σ^k_j (for $1 \leq k \leq n$ and $1 \leq j \leq m_k$) is the composition of all substitutions used on the j-th path[a] from ℓ_k to $\bot$.

[a] Since a proof is generally a DAG and not necessarily a tree, there may be more than one path connecting ℓ_k to $\bot$ in the DAG-like proof.

Fig. 2: The Conflict Resolution Calculus **CR**

The *conflict* rule is just a restriction of the resolution rule. The *unit-propagating resolution*[3] rule is essentially a sequence of applications of the resolution rule where the left premises must always be unit clauses; the conclusion clause must be unit as well, and its literal is called a *propagated literal.*

The main innovation lies in the *conflict-driven clause learning* rule. The literals within brackets are the *decision literals* that have been assumed. The superscript index i indicates that this assumption is discharged by the **cl** inference with index i. It is not required that a **cl** inference discharge all decision literals above it. Some decision literals may be left undischarged, to be discharged by future **cl** inferences. The vertical dots denote any derivation of $\perp$ using the decision literals, input clauses and previously derived clauses. The conclusion clause of this rule is the *learned clause.* In contrast to the propositional case, the learned clause must be a disjunction of duals of *instances* of the discharged decision literals[4], because variables occurring in the discharged decision literals may be instantiated by unifications performed during the proof. Since the derivation of $\perp$ need not be tree-like, we may need to consider several instances of each decision literal.

A **CR** derivation is a **CR** *proof* if and only if all its decision literals have been discharged. A **CR** *refutation* is a **CR** proof of $\perp$.

Example 3 Consider the following clause set:

$$\{P(z) \vee Q,\ P(y) \vee \neg Q,\ \neg P(a) \vee Q,\ \neg P(b) \vee \neg Q\}$$

It admits the **CR** refutation shown in Fig. 3. A shorter refutation would be possible if we had taken, for instance, Q as a decision literal. But taking $P(x)$ as a decision literal instead, as done in the refutation in Fig. 3, we can see how conflict driven clause learning behaves in the first-order case, when decision literals can contain variables, that can be instantiated during the process of propagation. In one path from ψ^1 to $\perp$ just above the $\mathbf{cl}^1$ inference, the unification performed by the unit-propagating resolution inference instantiates x with a, whereas in the other path x is instantiated with b. Therefore, the $\mathbf{cl}^1$ inference learns the clause $\neg P(a) \vee \neg P(b)$, which is the disjunction of the duals of all the instances of the decision literal $P(x)$. This is in contrast with (and a generalization of) the propositional case, where instances did not need to be considered. As in the propositional case, the decisions and unit propagations can be represented graphically, as shown in the bottom of Fig. 3 as well.

[3] This rule is also known as *unit-resulting resolution* [25,26]. Here we use the name *unit-propagating resolution* instead in order to make the connection with the technique of unit-propagation more explicit.

[4] As mentioned in Section 3, modern SAT-solvers implement optimized clause learning procedures that may also contain duals of propagated literals. A modification of **CR**'s **cl** inference rule to cover more sophisticated clause learning procedures is briefly discussed in Subsection 4.2.

$$\dfrac{\dfrac{\dfrac{\psi_1 : [P(x)]^1 \qquad \xi_1 : \neg P(a) \vee Q}{Q}\,\mathbf{u}(\{x\backslash a\}) \qquad \dfrac{\psi_1 \qquad \neg P(b) \vee \neg Q}{\neg Q}\,\mathbf{u}(\{x\backslash b\})}{\bot}\,\mathbf{c}(\varepsilon)}{\varphi_1 : \neg P(a) \vee \neg P(b)}\,\mathbf{cl}^1$$

$$\dfrac{\dfrac{\dfrac{\psi_2 : [\neg P(a)]^2 \qquad P(z) \vee Q}{Q}\,\mathbf{u}(\{z\backslash a\}) \qquad \dfrac{\psi_2 \qquad \xi_2 : P(y) \vee \neg Q}{\neg Q}\,\mathbf{u}(\{y\backslash a\})}{\bot}\,\mathbf{c}(\varepsilon)}{\varphi_2 : P(a)}\,\mathbf{cl}^2$$

$$\dfrac{\dfrac{\dfrac{\varphi_2 \qquad \varphi_1}{\neg P(b)}\,\mathbf{u}(\varepsilon) \qquad \xi_2}{\neg Q}\,\mathbf{u}(\{y\backslash b\}) \qquad \dfrac{\varphi_2 \qquad \xi_1}{Q}\,\mathbf{u}(\varepsilon)}{\bot}\,\mathbf{c}(\sigma)$$

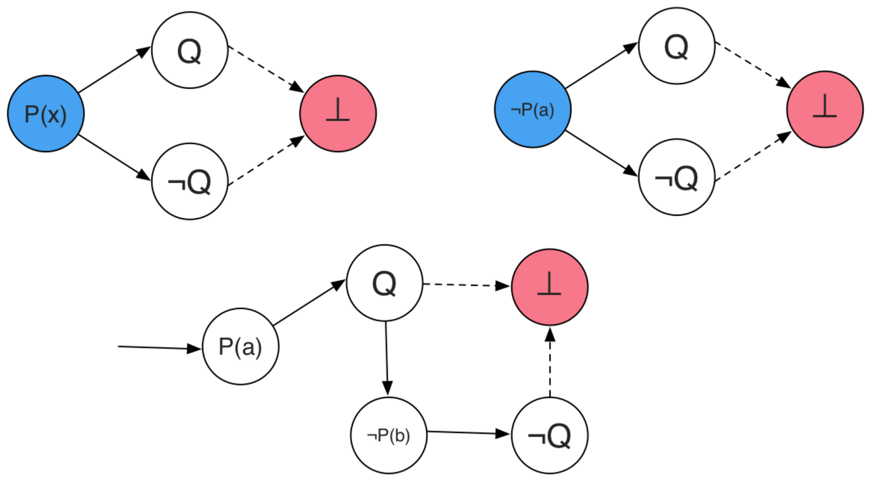


Fig. 3: **CR** Refutation and conflict graphs for the clause set from Example 3.

Unlike the Resolution calculus, **CR** does not need a primitive *factoring* rule. This rule can be simulated by a sequence of decisions, one unit-propagation, one conflict and one conflict-driven clause learning. In this way, we can prove the following lemma.

Lemma 1 *Resolution's factoring rule is admissible in* ***CR****.*

Proof Let φ' be a **CR** derivation of $\ell \vee \ell' \vee \ell_1 \vee \ldots \vee \ell_m$ and consider constructing φ by applying the factoring inference to the conclusion of φ', as shown below:

$$\dfrac{\begin{array}{c}\vdots\ \varphi' \\ \ell \vee \ell' \vee \ell_1 \vee \ldots \vee \ell_m\end{array}}{(\ell \vee \ell_1 \vee \ldots \vee \ell_m)\ \sigma}\,\mathbf{f}(\sigma)$$

$$\cfrac{\cfrac{\cfrac{\psi : [\overline{\ell}\sigma]^{\overset{1}{1}} \quad \psi \quad [\overline{\ell_1}]^{\overset{1}{2}} \quad \dots \quad [\overline{\ell_{m-1}}]^{\overset{1}{m}} \quad \overset{\vdots\ \varphi'}{\ell \vee \ell' \vee \ell_1 \vee \dots \vee \ell_m}}{\ell_m\ \sigma}\ \mathbf{u}(\sigma) \qquad [\overline{\ell_m}]^{\overset{1}{m+1}}}{\bot}\ \mathbf{c}(\sigma)}{(\ell \vee \ell_1 \vee \dots \vee \ell_m)\ \sigma}\ \mathbf{cl}^1$$

where σ is a unifier of ℓ and ℓ'.

Fig. 4: Simulation of a Factoring Inference in **CR**

This is admissible because, instead of using the factoring inference, we could have used the sequence of **CR** inferences shown in Fig. 4.

The simulation of factoring depends on a sufficiently high degree of freedom in the choice of decision literals. We must be allowed (as indeed we are in **CR**) to assume a decision literal ($\overline{\ell}\sigma$) that is the dual of a common instance of ℓ and ℓ'.

4.1 An Isomorphism between Conflict Graphs and Single-Conflict Sub-Derivations in Conflict Resolution

By comparing the conflict graphs and **CR** derivations in Example 3, it is noticeable that there is a straightforward isomorphism between conflict graphs and **CR** sub-derivations with a single conflict inference. Every decision literal in a conflict graph appears as a decision literal in the corresponding **CR** derivation. Every propagated literal in the conflict graph appears as a propagated literal derived by a unit-propagating resolution inference, and the clause associated to the incoming edges of the propagated literal is exactly the non-unit clause used as the rightmost premise of the unit-propagating inference. Finally, the conflict in the conflict graph is a conflict inference in the corresponding **CR** derivation.

In contrast, the correspondence between resolution derivations and conflict graphs is imperfect. As illustrated in Example 2, we have a map from conflict graphs to resolution derivations; however, this map is not an isomorphism, simply because it is not even surjective. Furthermore, there is a mismatch between the conflict graph operations (i.e. decisions, propagations and conflict) and the operations of the resolution calculus (i.e. the resolution and factoring inference rules). In other words, no map from conflict graphs to resolution derivations could be an isomorphism, because the algebraic structure cannot be preserved. From this algebraic point of view, we may conjecture that the popular belief that (propositional) resolution is the underlying proof system of modern SAT-solvers (which actually implement the CDCL procedure based on conflict graphs) is mistaken. We also speculate that the mismatch is the theoretical explanation for the overhead experienced in the transformation of conflict graphs to resolution derivations (as discussed in the end of Section 3).

First UIP Conflict-Driven Clause Learning:

$$\dfrac{\begin{array}{ccc} [\ell'_1]^{i}_{1} \quad [\ell'_h]^{i}_{1} & & \\ \vdots & & \\ \ell_1 & & [\ell_n]^{i}_{n} \\ \vdots\ (\sigma^1_1,\ldots,\sigma^1_{m_1}) & & \vdots\ (\sigma^n_1,\ldots,\sigma^n_{m_n}) \\ & \vdots & \\ & \bot & \end{array}}{(\overline{\ell_1}\sigma^1_1 \vee \ldots \vee \overline{\ell_1}\sigma^1_{m_1}) \vee \ldots \vee (\overline{\ell_n}\sigma^n_1 \vee \ldots \vee \overline{\ell_n}\sigma^n_{m_n})}\ \textbf{uipcl}^i$$

where:

– σ^k_j (for $1 \leq k \leq n$ and $1 \leq j \leq m_k$) is the composition of all substitutions used on the j-th path from ℓ_k to $\bot$;
– The propagated literal ℓ_1 must be a *UIP*: all paths from ℓ'_g (for $1 \leq g \leq h$) to $\bot$ must pass through ℓ_1.
– ℓ_1 must be a *first UIP*: there should be no other UIP on a path from ℓ_1 to $\bot$.

Fig. 5: First UIP Clause Learning

Perhaps a calculus such as **CR**, that enjoys a better correspondence to conflict graphs, could enable proof production with less overhead.

4.2 Clause Learning Optimizations

Learning clauses that contain only duals of decision literals is the simplest conflict-driven clause learning (CDCL) strategy. It is the strategy explained in Section 4.4.1 of the Handbook of Satisfiability [24]. Therefore, **CR**'s clause learning rule mimics (and generalizes to the first-order case) the simplest CDCL mechanism. To mimic more sophisticated and empirically more effective CDCL mechanisms, such as the *first UIP* [39], it suffices to define a modification of **CR**'s **cl** rule that allows learning clauses containing instances of the dual of a propagated literal that satisfies the conditions to be a *first unit implication point* (UIP). A precise definition of these conditions is possible, because the isomorphism discussed in Section 4.1 allows a translation of the usual definition[5] of UIP in terms of conflict graphs (cf. Section 4.4.3 of the Handbook of Satisfiability) to the terminology of **CR** proofs, as done in the **uipcl** rule shown in Figure 5. For the sake of simplicity and generality, however, the remaining sections of this paper focus on conflict resolution with the simpler **cl** inference rule, which is sufficient for this paper's goals.

[5] The usual definition of UIP in terms of conflict graphs also relies on the notion of *decision level* of assigned literals and requires that the propagated literal be assigned at the current decision level. Therefore, our definition is slightly more general. This is intentional. The notion of decision level, which keeps track of the order in which decision literals were assigned, is beyond the purely proof-theoretical scope of this paper, as it relates to particular proof search and backtracking procedures.

5 Refutational Completeness

A proof system **P** is *refutationally complete* if and only if any unsatisfiable clause set has a refutation in **P**. Instead of proving refutational completeness for **CR** directly, we will prove it indirectly, showing that **CR** can simulate another refutationally complete proof system. A proof system **P** simulates another proof system **Q** if and only if there is a map transforming any **Q**-derivation of c from S to **P**-derivation of c from S.

This indirect approach to proving completeness can be traced back at least to Gentzen's work [19], who applied it to his natural deduction and sequent calculi. In our case, the target proof system for the simulation is resolution, and the key idea of the simulation is that every resolution step that is not a unit-propagating resolution inference can be simulated by several decisions, two unit-propagating resolution inferences, one conflict inference and one conflict-driven clause learning inference.

Theorem 1 ***CR*** *simulates Resolution.*

Proof Let ψ be a Resolution derivation of a clause c from a set of clauses S. We show that there is a **CR** derivation φ of c from S, proceeding by induction:

– *Base Case:* ψ is just a single node c. In this case, φ is just the single node c as well.
– *Induction Case 1:* ψ ends with a factoring inference ρ. In this case, let ψ' be the sub-derivation whose conclusion c' is the premise of ρ. By induction hypothesis, there is a **CR** derivation φ' of c' from S. And then φ can be constructed as the **CR** derivation of c from S obtained from φ' by applying the admissible factoring inference rule to its conclusion in the same way as ρ in ψ or by simulating factoring as shown in Fig. 4. In any case, the conclusion of φ is c, as desired.
– *Induction Case 2:* ψ ends with a resolution inference. In this case, ψ is of the following form:

$$\dfrac{\overset{\vdots\, \psi_1}{\ell_1 \vee \ldots \vee \ell_n \vee \ell} \qquad \overset{\vdots\, \psi_2}{\overline{\ell'} \vee \ell'_1 \vee \ldots \vee \ell'_m}}{(\ell_1 \vee \ldots \vee \ell_n \vee \ell'_1 \vee \ldots \vee \ell'_m)\ \sigma}\ \mathbf{r}(\sigma)$$

By induction hypothesis, we have a **CR** derivation φ_1 of $\ell_1 \vee \ldots \vee \ell_n \vee \ell$ from S and a **CR** derivation φ_2 of $\overline{\ell'} \vee \ell'_1 \vee \ldots \vee \ell'_m$ from S. Then a **CR** derivation φ of $(\ell_1 \vee \ldots \vee \ell_n \vee \ell'_1 \vee \ldots \vee \ell'_m)\ \sigma$ can be constructed as shown in Fig. 6.

Corollary 1 ***CR*** *is refutationally complete.*

Proof Let C be an unsatisfiable clause set. As resolution is a refutationally complete calculus [28], there is a resolution refutation ψ of C. By Theorem 1, ψ can be transformed to a **CR** refutation φ of C.

$$\dfrac{\dfrac{\dfrac{[\overline{\ell_1}]^1 \quad \dots \quad [\overline{\ell_n}]^1 \quad \overset{\vdots\,\varphi_1}{\ell_1 \vee \dots \vee \ell_n \vee \ell}}{\ell}\,\mathbf{u}(\varepsilon) \qquad \dfrac{[\overline{\ell'_1}]^1 \quad \dots \quad [\overline{\ell'_m}]^1 \quad \overset{\vdots\,\varphi_2}{\overline{\ell'} \vee \ell'_1 \vee \dots \vee \ell'_m}}{\overline{\ell'}}\,\mathbf{u}(\varepsilon)}{\bot}\,\mathbf{c}(\sigma)}{(\ell_1 \vee \dots \vee \ell_n \vee \ell'_1 \vee \dots \vee \ell'_m)\ \sigma}\,\mathbf{cl}^1$$

Fig. 6: Simulation of a Resolution Inference in **CR**

A mere restriction of resolution to unit-propagating resolution would have resulted in a refutationally incomplete calculus. The unsatisfiable clause sets from Examples 2 and 3, for instance, would not be refutable. By incorporating decision literals, as well as the conflict rule and the conflict-driven clause learning rule, we regain refutational completeness.

An interesting insight learned from the completeness proof is the following corollary.

Corollary 2 *CR with the additional restriction that every decision literal must be the dual of a most general common instance of a non-empty subset of literals occurring in a single clause is still a refutationally complete calculus.*

Proof Only duals of literals occurring in resolved clauses are needed as decision literals to simulate resolution by **CR** in the proof of Theorem 1, and only duals of literals occurring in the factored clause and the dual of a common instance of the factored literals are needed to simulate factoring by **CR** in the proof of Theorem 1. Therefore, the same proof would still succeed for **CR** with the additional restriction that decision literals must be common instances of a non-empty subset of literals occurring in a single clause. Moreover, resolution is still refutationally complete when the unifiers in the resolution and factoring inferences are required to be most general unifiers. Consequently, it suffices to consider most general common instances as decision literals.

This corollary is useful for proof search, because it entails that a decision literal does not need to be arbitrarily guessed. It can be picked from a finite set of alternatives. An effective implementation of **CR** is, therefore, possible.

The fact that we need two unit-propagating resolution inferences, one conflict and one conflict-driven clause learning to simulate a single resolution inference (as shown in Fig. 6) may lead us to think that **CR** is more bureaucratic and more inefficient than resolution. However, efficiency of proof search is not directly correlated with proof length. In **CR** much fewer clauses are generated by unit-propagating resolution than by unrestricted resolution and the clause sizes are reduced through decisions and propagations. Moreover, in any case, any resolution proof search, as well as any resolution proof, can be simulated in the **CR** calculus with only a (small) linear increase in *length* (i.e. number of inferences) and *size* (i.e. number of literals).

Theorem 2 *For every resolution refutation ψ, there exists a* ***CR*** *refutation φ such that $length(\varphi) \in O(length(\psi))$ and $size(\varphi) \in O(size(\psi))$.*

Proof Given ψ, let φ be obtained from ψ by simulating all its inferences as described in the proof of Theorem 1. Then, if ψ has n resolutions and m factorings, φ has $n + m$ clause learning inferences, $n + m$ conflicts and $2n + m$ unit propagations. Hence, $length(\varphi) = 4n+3m \in O(n+m) = O(length(\psi))$. If ψ has a total of h' literals occurring in all its h leaf clauses, n' literals occurring in conclusions of all its n resolution inferences and m' literals occurring in conclusions of all its m factoring inferences, then φ has h' literals occurring in its leaf clauses, $n' + m'$ literals occurring in conclusions of clause learning inferences, $2(n + m)$ literals occurring in premises of conflict inferences, n' literals occurring as decision literals for unit propagations in the simulations of resolution inferences and m' literals occurring as decision literals for unit propagations in the simulations of factoring inferences. Hence, $size(\varphi) = h' + (n' + m') + 2(n + m) + n' + m' = h' + 2(n' + m') + 2(n + m)$. Since every resolution inference in ψ has at least one literal in its conclusion, except for the last one deriving the empty clause, $n \leq n' + 1$. Moreover, since the number of literals in the conclusion of every factoring inference is at least 1, $m \leq m'$. Therefore, $size(\varphi) \leq h' + 2(n' + m') + 2(n' + 1 + m') = h' + 4(n' + m') + 2 \leq 4(h' + n' + m') + 2 \in O(size(\psi))$.

6 Soundness

To prove soundness, we exploit the key observation that decision literals resemble natural deduction's *assumptions* and conflict-driven clause learning resembles *implication/negation introduction*, decision literals are discharged by conflict-driven clause learning as assumptions are discharged by implication or negation introduction. Therefore, natural deduction is an excellent candidate for proving soundness indirectly by simulation. However, typical natural deduction rules (cf. Appendix) operate on general formulas, which are not necessarily in clause form, and this makes a direct simulation technically difficult. In order to overcome this challenge, we define an intermediary *clausal natural deduction calculus* (abbreviated as **CND**) with inference rules that operate on clauses, as shown in Fig. 7.

The clausal natural deduction calculus **CND** can be simulated by any standard non-clausal natural deduction calculus extended with a classical rule for double negation elimination (e.g. the calculus shown in the Appendix). The key idea is to use the well-known classical equivalence $A \vee B \equiv (\dot{\neg} A \rightarrow B)$ (where $\dot{\neg} A$ is an abbreviation for $A \rightarrow \bot$), in order to transform the clauses in a **CND** proof into formulas containing only implication, which are therefore suitable for a minimal non-clausal natural deduction calculus. When transforming a **CND** proof into a standard non-clausal natural deduction proof, sequences of implication introduction/elimination rules may have to be added to the natural deduction proof, in order to reorder literals (because

Implication Elimination (Modus Ponens):

$$\frac{\ell \quad \bar{\ell} \vee \Gamma}{\Gamma} \rightarrow_E$$

Implication Introduction:

$$\frac{\begin{array}{c}[\ell]^i \\ \vdots \\ \Gamma\end{array}}{\bar{\ell} \vee \Gamma} \rightarrow_I^i$$

$[\ell]$ is an *assumption* discharged by the implication introduction.

Universal Quantification Elimination:

$$\frac{\Gamma}{\Gamma\sigma} \forall_E$$

Universal Quantification Introduction:

$$\frac{\Gamma\{x_1 \backslash \alpha_1, \ldots, x_n \backslash \alpha_n\}}{\Gamma} \forall_I$$

α_k must be a distinct *eigen-variable*:
it should occur neither in Γ nor in any undischarged assumption.

Fig. 7: The Clausal Natural Deduction Calculus **CND**

associativity and commutativity of disjunction is implicitly taken into account by **CND**'s inference rules, but must be handled explicitly in a standard natural deduction calculus). The classical rule of double negation elimination is needed in order to handle the involutivity of classical negation, which is implicit in **CND**[6]. A more detailed proof of this simulation is omitted because it would be tedious and space-consuming. Soundness of **CND** is a corollary of the simulation, since natural deduction is sound.

Remembering that all clausal rules are assumed to be modulo negation's involutivity and modulo the neutrality of $\perp$ w.r.t. disjunction, the rules for negation introduction and elimination shown in Fig. 8 are admissible in **CND**, since they are just special cases of, respectively, implication introduction and elimination, when $\Gamma = \perp$. We are now ready to prove the following theorem.

Theorem 3 ***CND*** *simulates* ***CR***.

Proof Given a **CR** derivation ψ of a clause c from a set of clauses S, we must construct a **CND** derivation φ of c from S (modulo variable renaming). We

[6] **CND** is a calculus for *classical* logic: the law of excluded middle can be easily derived with a single application of implication introduction. Interestingly, its classicality is implicit in the use of involutive negation and the equivalence involving disjunctions and implications.

Negation Elimination:

$$\frac{\ell \quad \overline{\ell}}{\bot}\ \neg_E$$

Negation Introduction:

$$\frac{\begin{array}{c}[\ell]^i \\ \vdots \\ \bot\end{array}}{\overline{\ell}}\ \neg_I^i$$

Fig. 8: **CND**'s Admissible Rules for Negation

first expand ψ into a tree-like proof ψ': for each clause c' with several children $c_1, \ldots, c_n$ (where $n > 1$), we create n copies $c^1, \ldots, c^n$ of c' and use each c^k (for $1 \leq k \leq n$) as a parent for c_k. The variables in each copy are renamed to fresh variables and all substitutions in the proof are updated accordingly, in order to maintain the property that distinct clauses in ψ' do not share variables (cf. Section 2). Now that ψ' is tree-like, we may compute its *global substitution* σ^* (i.e. the composition (in topological order) of all the substitutions used in the proof)[7].

We now do a recursive top-down traversal of ψ' and for each sub-derivation η deriving a clause c' from S with decision literals $[\ell_1], \ldots, [\ell_n]$, we construct a corresponding sub-derivation ξ deriving $c'\ \sigma^*$ from S with assumptions $[\ell_1\ \sigma^*], \ldots, [\ell_n\ \sigma^*]$:

– *Base Case 1:* η has just a leaf node containing a decision literal $[\ell]$. In this case, ξ is the leaf node containing the assumption $[\ell\ \sigma^*]$.
– *Base Case 2:* η has just a leaf node containing a clause Γ. In this case, ξ is:

$$\frac{\Gamma}{\Gamma\ \sigma^*}\ \forall_E$$

– *Induction Case 1:* η ends with a unit-propagating resolution inference, as shown below:

$$\frac{\begin{array}{c}\vdots\ \eta_1 \\ \ell_1\end{array} \quad \ldots \quad \begin{array}{c}\vdots\ \eta_n \\ \ell_n\end{array} \quad \begin{array}{c}\vdots\ \eta' \\ \overline{\ell'_1} \vee \ldots \vee \overline{\ell'_n} \vee \ell\end{array}}{\ell\ \sigma}\ \mathbf{u}(\sigma)$$

By induction hypothesis, there are **CND** derivations $\xi_1, \ldots, \xi_n, \xi'$ of, respectively, $\ell_1\ \sigma^*, \ldots, \ell_n\ \sigma^*, (\overline{\ell'_1} \vee \ldots \vee \overline{\ell'_n} \vee \ell)\ \sigma^*$. We then construct ξ

[7] Since we assume that distinct clauses in ψ' do not share variables and ψ' is tree-like, we do not need to worry about variable clashes in the composition of all substitutions. The topological order is needed because a variable x introduced by a substitution σ_1 may be in the domain of another substitution σ_2 occurring below σ_1. In this case, the topologically ordered composition is $\sigma_1\sigma_2$ (i.e. apply first σ_1 and then σ_2).

by applying implication elimination n times, as shown below:

$$\dfrac{\begin{array}{c}\vdots\ \xi_n\\ \ell_n\ \sigma^*\end{array} \qquad \dfrac{\vdots \qquad \dfrac{\begin{array}{c}\vdots\ \xi_1\\ \ell_1\ \sigma^*\end{array}\quad \begin{array}{c}\vdots\ \xi'\\ (\overline{\ell'_1}\vee\ldots\vee\overline{\ell'_n}\vee\ell)\ \sigma^*\end{array}}{(\overline{\ell'_2}\vee\ldots\vee\overline{\ell'_n}\vee\ell)\ \sigma^*}\rightarrow_E}{\vdots}\rightarrow_E}{\ell\ \sigma^*}\rightarrow_E$$

– *Induction Case 2:* η ends with a conflict inference. This case is analogous to the case above. But, instead of n implication elimination inferences, a single negation elimination inference suffices.
– *Induction Case 3:* η ends with a conflict-driven clause learning inference. In this case, the corresponding subproof in ψ used to have the following form:

$$\dfrac{\begin{array}{c}\begin{array}{ccc}[\ell_1]^{\overset{i}{1}} & & [\ell_n]^{\overset{i}{n}}\\ \vdots\ (\sigma^1_1,\ldots,\sigma^1_{m_1}) & & \vdots\ (\sigma^n_1,\ldots,\sigma^n_{m_n})\end{array}\\ \vdots\\ \bot\end{array}}{(\overline{\ell_1}\sigma^1_1\vee\ldots\vee\overline{\ell_1}\sigma^1_{m_1})\vee\ldots\vee(\overline{\ell_n}\sigma^n_1\vee\ldots\vee\overline{\ell_n}\sigma^n_{m_n})}\ \mathbf{cl}^i$$

But due to the expansion to a tree, the subproof η in ψ' has the form shown below, where there is a copy $[\ell^j_k]$ of a decision literal $[\ell_k]$ for every path j that existed from $[\ell_k]$ to $\bot$ in ψ. The copies have fresh variables, but are identical modulo variable renaming. For every k and j, the substitution $\sigma^{j'}_k$ is essentially identical to σ^j_k, except for the use of different variable names.

$$\dfrac{\begin{array}{c}\begin{array}{ccccccc}[\ell^1_1]^i & & [\ell^{m_1}_1]^i & & [\ell^1_n]^i & & [\ell^{m_n}_n]^i\\ \vdots\ \sigma^{1'}_1 & \ldots & \vdots\ \sigma^{1'}_{m_1} & \ldots & \vdots\ \sigma^{n'}_1 & \ldots & \vdots\ \sigma^{n'}_{m_n}\end{array}\\ \vdots\\ \bot\end{array}}{(\overline{\ell_1}\sigma^{1'}_1\vee\ldots\vee\overline{\ell_1}\sigma^{1'}_{m_1})\vee\ldots\vee(\overline{\ell_n}\sigma^{n'}_1\vee\ldots\vee\overline{\ell_n}\sigma^{n'}_{m_n})}\ \mathbf{cl}^i$$

By induction hypothesis, there is a derivation ξ' with the form:

$$\begin{array}{c}\begin{array}{ccccccc}[\ell^1_1\ \sigma^*] & & [\ell^{m_1}_1\ \sigma^*] & & [\ell^1_n\ \sigma^*] & & [\ell^{m_n}_n\ \sigma^*]\\ \vdots & \ldots & \vdots & \ldots & \vdots & \ldots & \vdots\end{array}\\ \vdots\\ \bot\end{array}$$

And then a derivation ξ can be constructed by applying the implication introduction rule k times, where k is the number of assumptions $[\ell^1_1\ \sigma^*]$,

$$\dfrac{\dfrac{\dfrac{\dfrac{[P(a)]^2 \qquad \neg P(a)\vee Q}{Q}\to_E \qquad \dfrac{[P(b)]^1 \qquad \neg P(b)\vee\neg Q}{\neg Q}\to_E}{\bot}\neg_E}{\neg P(b)}\neg_I^1}{\varphi_1 : \neg P(a)\vee\neg P(b)}\to_I^2$$

$$\dfrac{\dfrac{\dfrac{[\neg P(a)]^3 \qquad \dfrac{P(z)\vee Q}{P(a)\vee Q}\forall_E}{Q}\to_E \qquad \dfrac{[\neg P(a)]^3 \qquad \dfrac{P(y)\vee\neg Q}{P(a)\vee\neg Q}\forall_E}{\neg Q}\to_E}{\bot}\neg_E}{\varphi_2 : P(a)}\neg_I^3$$

$$\dfrac{\dfrac{\dfrac{\varphi_2 \qquad \varphi_1}{\neg P(b)}\to_E \qquad \dfrac{P(v)\vee\neg Q}{P(b)\vee\neg Q}\forall_E}{\neg Q}\to_E \qquad \dfrac{\varphi_2' \qquad \neg P(a)\vee Q}{Q}\to_E}{\bot}\neg_E$$

where φ_2' is a reference to a copy of φ_2.

Fig. 9: **CND** Refutation Simulating the **CR** Refutation from Fig. 3.

$\ldots, [\ell_1^{m_1}\ \sigma^*], \ldots, [\ell_n^1\ \sigma^*], \ldots, [\ell_n^{m_n}\ \sigma^*]$ to be discharged, as depicted below:

$$\dfrac{\begin{array}{c}[\ell_1^1\ \sigma^*]^1 \qquad\qquad [\ell_n^{m_n}\ \sigma^*]^k\\ \vdots \qquad \ldots \qquad \vdots\\ \vdots\\ \bot\end{array}}{(\overline{\ell_1^1}\sigma^* \vee \ldots \vee \overline{\ell_1^{m_1}}\sigma^*) \vee \ldots \vee (\overline{\ell_n^1}\sigma^* \vee \ldots \vee \overline{\ell_n^{m_n}}\sigma^*)}\to_I^1,\ldots,\to_I^k$$

Since σ^* is the composition of all substitutions in ψ', including every $\sigma_k^{j'}$, we have that $\sigma_k^{j'}\sigma^* = \sigma^*$. Therefore, the conclusion of ξ is identical to:

$$((\overline{\ell_1}\sigma_1^{1'} \vee \ldots \vee \overline{\ell_1}\sigma_{m_1}^{1'}) \vee \ldots \vee (\overline{\ell_n}\sigma_1^{n'} \vee \ldots \vee \overline{\ell_n}\sigma_{m_n}^{n'}))\ \sigma^*$$

At the end of the top-down traversal, we have a **CND** proof φ of $c\ \sigma^*$ from S. Since σ^* is the global substitution of all substitutions used in ψ' and ψ' derives c, we have that $c\ \sigma^* = c$. Therefore, φ is a **CND** proof of c from S, as desired.

Example 4 To illustrate the transformation of **CR** derivations into **CND** derivations used in the proof of Theorem 3, Fig. 9 shows the **CND** derivation obtained by transforming the **CR** derivation shown in Fig. 3.

Corollary 3 *CR is sound.*

Proof Let φ be an arbitrary **CR** proof of c from S. Then, by Theorem 3, there is a **CND** proof of c from S. Since the natural deduction calculus **CND** is sound, c is entailed by S. Therefore, **CR** is sound.

7 Simulation of Splitting

Suppose that a prover refutes the set of clauses $S \cup \{\Gamma_1 \vee \ldots \vee \Gamma_k\}$ (where the sets of variables V_i of Γ_i are mutually disjoint), by splitting it into the k sets $S \cup \{\Gamma_i\}$ (for $1 \leq i \leq k$) and finding a Resolution refutation ψ_i for each set $S \cup \{\Gamma_i\}$. One way to combine these proofs into a single resolution refutation of $S \cup \{\Gamma_1 \vee \ldots \vee \Gamma_k\}$ would be to use the following recursive method:

- *For $i = 1$:* construct ψ'_1 by replacing every leaf occurrence of Γ_1 in ψ_1 by $\Gamma_1 \vee \ldots \vee \Gamma_k$, propagating the added literals downwards and factoring the added literals when possible; then ψ'_1 is not a refutation, but a derivation of $\Gamma_2 \vee \ldots \vee \Gamma_k$.
- *For i from 2 to k:* construct ψ'_i by replacing every leaf occurrence of Γ_i in ψ_i by the subproof ψ'_{i-1} deriving $\Gamma_i \vee \ldots \vee \Gamma_k$; as before, propagate the added literals downwards and factor them when possible, so that ψ'_i is a proof of $\Gamma_{i+1} \vee \ldots \vee \Gamma_k$, if $i < k$, or $\bot$, otherwise.

However, this method is undesirable, because it requires a substantial modification of the component proofs ψ_i. The modified subproofs are larger (because of all the additional literals), and this may hinder readability of the proof by humans and reduce the efficiency of automatic proof checking.

A pragmatic approach is to disregard the attempt to output a single refutation for the original problem and simply output all the separate proofs for the split problems instead. Keeping track of all splittings is important, particularly in the more general case where splitting is done recursively (i.e. where each set $S \cup \{\Gamma_i\}$ can be split further). This seems to be the approach taken by many automated theorem provers. Splittings performed during the proof search are recorded in the proof file in an extra-logical way, which may even violate informal semantic requirements of the TPTP proof format[8].

In **CR**, splitting can be simulated in such a way that the refutations for the split sub-problems can be combined without the drawbacks that are incurred when this is done in Resolution. Suppose that φ_i are refutations[9] of $S \cup \{\Gamma_i\}$. Then a refutation φ of $S \cup \{\Gamma_1 \vee \ldots \vee \Gamma_k\}$ can be constructed by combining all the φ_i (for $1 \leq i \leq k$) using the following recursive method:

- *For $i = 1$:* construct φ'_1 by replacing every leaf occurrence of Γ_1 in φ_1 by the following **CR** subproof (where $\ell_i^1, \ldots, \ell_i^{n_i}$ are duals of the literals in Γ_i):

$$\cfrac{\cfrac{[\ell_1^1]^1 \qquad \cfrac{[\ell_1^2]^1 \ldots\ [\ell_1^{n_1}]^1 \quad \ldots \quad [\ell_k^1]^k \quad \ldots \quad [\ell_k^{n_k}]^k \quad \Gamma_1 \vee \ldots \vee \Gamma_k}{\overline{\ell_1^1}}\ \mathbf{u}(\varepsilon)}{\bot}\ \mathbf{c}(\varepsilon)}{\Gamma_1}\ \mathbf{cl}^1$$

[8] TPTP's general proof format [31] requires that the conclusion of an inference rule be a logical consequence of (or equi-satisfiable to) its premises. This limitation prevents an easy representation of natural deduction's implication introduction rule, tableaux's β rule or splitting. **CR**'s conflict-driven clause learning is also affected by this limitation.

[9] It is assumed (without loss of generality) that, for every i, Γ_i is actually used in φ_i. Otherwise, if Γ_j were not used in φ_j for some j, φ_j would already be a refutation of S, and it would not be necessary to split $\Gamma_1 \vee \ldots \vee \Gamma_k$.

– *For i from* 2 *to k:* construct φ'_i by replacing every leaf occurrence of Γ_i in φ_i by the following subproof:

$$\dfrac{\begin{array}{c}\vdots\ \varphi'_{i-1}\\ \bot\end{array}}{\Gamma_i}\ \mathbf{cl}^i$$

The desired refutation φ of $S \cup \{\Gamma_1 \vee \ldots \vee \Gamma_k\}$ is then φ'_k.

This method of simulating splitting in **CR** requires no internal modification of the proofs φ_i: the modified proofs φ'_i $(2 \leq i \leq k)$ are just φ_i with a few **cl** inferences on top. Hence, there is no loss in readability, and the only overhead for automatic proof checking is caused by the need to check the additional **cl** inferences. If the leaf clause Γ_i occurs only once[10], a single **cl** inference suffices, in fact. Therefore, the increase in proof size and the overhead for proof checking are negligible.

The simulation described here shows that splitting can be seen as a macro-rule that performs, for a variable-disjoint component Γ_i, batch decisions assuming the duals of all literals not in Γ_i. The first-order mechanism of decisions and conflict-drive clause learning provided by **CR** is, however, more general, because it allows splitting even when the components are not variable-disjoint.

8 CR with Sequent Notation

The proof of **CR**'s soundness in Section 6 demonstrates that there is a lot in common between **CR** and natural deduction. In the same way that natural deduction can be presented with a sequent notation, in which assumptions are listed in the antecedent of the sequent (i.e. at the left side of the turnstile symbol), **CR** can also be presented with a sequent notation, with decision literals kept at the antecedent[11]. This is shown in Fig. 10.

With the sequent notation, it is easier to state the inference rule for conflict-driven clause learning. All the substitutions that should be applied to the literals whose duals will be part of the learned clause have already been applied to the literals in the antecedent. There is no need to look at the substitutions that have been used in the paths above. On the other hand, the presentation with sequent notation is much more redundant and bureaucratic. Whereas in the standard presentation, the use of decision literals is a powerful way to reduce the size of clauses (as in the simulation of splitting), this beneficial effect is lost in the presentation with the sequent notation, because the decision literals are carried along in the antecedents.

[10] It may be reused many times, since φ_i does not need to be tree-like.

[11] Although antecedents of sequents and a Sat-solver's *trail* resemble each other, as both store decision literals, they are not quite the same thing. Whereas a Sat-solver's trail is typically a global structure, each sequent's antecedent is local and stores only the decisions that have been necessary to derive the sequent's succedent.

Decision:

$$\frac{}{\ell^i \vdash \ell^i}$$

Initial:

$$\frac{}{\vdash c}$$

if c is an input clause

Unit-Propagating Resolution:

$$\frac{\Delta_1 \vdash \ell_1 \quad \dots \quad \Delta_n \vdash \ell_n \quad \Delta \vdash \overline{\ell'_1} \vee \dots \vee \overline{\ell'_n} \vee \ell}{\Delta_1\ \sigma, \dots, \Delta_n\ \sigma, \Delta\ \sigma \vdash \ell\ \sigma}\ \mathbf{u}(\sigma)$$

where σ is a unifier of ℓ_k and ℓ'_k, for all $k \in \{1, \dots, n\}$.

Conflict:

$$\frac{\Delta_1 \vdash \ell \quad \Delta_2 \vdash \overline{\ell'}}{\Delta_1\ \sigma, \Delta_2\ \sigma \vdash \bot}\ \mathbf{c}(\sigma)$$

where σ is a unifier of ℓ and ℓ'.

Conflict-Driven Clause Learning:

$$\frac{\Delta, \ell_1^i\sigma_1^1, \dots, \ell_1^i\sigma_{m_1}^1, \dots, \ell_n^i\sigma_1^n, \dots, \ell_n^i\sigma_{m_n}^n \vdash \bot}{\Delta \vdash (\overline{\ell_1}\sigma_1^1 \vee \dots \vee \overline{\ell_1}\sigma_{m_1}^1) \vee \dots \vee (\overline{\ell_n}\sigma_1^n \vee \dots \vee \overline{\ell_n}\sigma_{m_n}^n)}\ \mathbf{cl}^i$$

Fig. 10: **CR** with Sequent Notation

For example, if we have the clause $\neg\ell_1 \vee \dots \vee \neg\ell_n \vee \ell$, then assuming the duals of the first n literals and resolving them with the clause through unit-propagation would result in the unit clause ℓ in the standard presentation. With sequent notation, on the other hand, we would obtain $\ell_1, \dots, \ell_n \vdash \ell$. While this may be conceptually convenient, because it reminds us explicitly that the unit clause ℓ holds only under the assumptions $\ell_1, \dots, \ell_n$, we have no reduction in size if we also count the antecedent's size.

In fact, because the proof may be a non-tree-like DAG, and decision literals may be instantiated by different substitutions along different paths of the DAG, several instances of the decision literal will accumulate in the antecedent. The number of instances may be in the worst case exponential in the height of the derivation. Although, in this case, the number of instances will be exponential both in the conclusion of a **cl** inference in the standard presentation of **CR** and in the succedent of the conclusion of a **cl** inference in **CR** with sequent notation, in the latter the exponential blow-up is also present in the antecedent of the premise (and possibly in the antecedents of many other sequents occurring above the premise as well). This is particularly important during proof search, when not all inferences lead to a conflict and, therefore, we do not want to accumulate (possibly exponentially many) instances of literals unnecessarily along the derivation. The standard presentation of **CR** (without sequent notation) avoids this accumulation; the derivation records only uninstantiated decision literals and a linear (w.r.t. the length of the derivation) number of unifiers that will only ever be combined and applied to instantiate the decision literals if a conflict is reached.

Moreover, the size of a single instance of a decision literal may be, in the worst case, exponential in the number of substitutions applied to the literal. Consider, for example, a decision literal of the form $p(x)$ and the following sequence of unifiers: $\sigma_1 = \{x \longrightarrow f_1(x_1, x_1)\}$, $\sigma_2 = \{x_1 \longrightarrow f_2(x_2, x_2)\}$, $\sigma_n = \{x_{n-1} \longrightarrow f_n(x_n, x_n)\}$. In the standard **CR** (without sequent notation), the substitutions are only applied to a decision literal if a conflict is reached and a conflict-driven clause learning inference is performed. In **CR** with sequent notation, on the other hand, substitutions are applied eagerly whenever a resolution or conflict inference is performed. Therefore, **CR** with sequent notation is more vulnerable to exponential blow-ups in instantiations.

9 Related Work

The seminal work of Baumgartner [5] was probably the first lifting of DPLL to the first-order case, and it soon led to further work by Baumgartner and Tinelli [7,6] on the *Model Evolution* (ME) procedure. It was later extended with a lemma learning rule [8], while retaining a traditional DPLL flavor (distinct from the *conflict graph* approach). In model evolution, decision literals do not contain standard variables, but *parameters*, which are variables with special semantics and behavior in the case of backtracking and clause learning. **CR** may be considered simpler, because it does not introduce the notion of parameter; however, in contrast to model evolution, for **CR** the problem of interpreting decision literals as a model has not been investigated yet.

In the model search procedure of de Nivelle's *Geometric Resolution* (GR) [18] implemented in the Geo prover, the literals chosen when geometric formulas of disjunctive or existential form are applied can be regarded as decision literals, or as propagated literals in the case of disjunctive geometric formulas with a single disjunct. However, in contrast to **CR**, such propagated and decision literals are always ground. Although the model search procedure in Geometric Resolution does resemble a CDCL procedure on first-order clauses of geometric form, the calculus lacks an assumption mechanism and a conflict-driven clause learning rule. Instead, learned clauses (called *closing lemmas* in [18]) are derived through a series of applications of special resolution rules.

More recently, Alagi and Weidenbach [1] proposed the *Non-Redundant Clause Learning* (NRCL) procedure, which generalizes CDCL to the Bernays-Schönfinkel fragment of first-order logic. They introduce the notion of *blocked decisions and clauses*, which restricts the decisions that can be made and thus allows them to prove that the learned clause is non-redundant (whereas in **CR** they might not be). They also introduce the notion of *constrained literals*, which allow more compact representation of the model. In **CR**, such optimizations and restrictions are intentionally avoided, in favor of a simple calculus focused on the core aspects of generalizing decisions and conflict-driven clause learning to *full* first-order logic.

Bonacina and Plaisted [10,11,12] proposed the (first-order) *Semantically-Guided Goal Sensitive* (SGGS) procedure inspired by CDCL. As they observe, there is a symmetry between positive and negative literals in the propositional case (i.e. in the sense that when a decision literal ℓ is false, $\overline{\ell}$ is true) which appears to be lost in the first-order case (i.e. because when ℓ is false, we cannot conclude that $\overline{\ell}$ is true; we can only conclude that $\overline{\ell}\,\sigma$ is true for some σ). One of the main challenges in lifting conflict-driven clause learning to first-order lies precisely in computing and dealing with the substitution σ when a decision literal ℓ leads to a conflict and a clause containing $\overline{\ell}\,\sigma$ must be learned. Instead of addressing this challenge, they circumvent it by introducing the notion of *uniform falsity*, according to which $\overline{\ell}$ must be true when ℓ is uniformly false. With this notion, clause learning is still essentially propositional and it is not triggered at every conflict (in the standard non-uniform sense of conflict). For instance, a conflict between $R(x)$ and $\neg R(b)$ does not lead to clause learning but must be repaired by revising $R(x)$ to $x \neq b \rhd R(x)$ instead.

The variety of approaches attempting to generalize CDCL to first-order logic shows that this is not a trivial task. The most pragmatically successful approaches so far have harnessed the power of Sat-solvers in first-order (or even higher-order) logic not by generalizing their underlying procedures but simply by employing them as *black-boxes* inside a theorem prover [15,21,22, 32,13]. Claessen's `Equinox` [15], for instance, is inspired by the modularity and extensibility of the architecture of SMT-solvers, but employs instead a modular approach that can handle full quantification. In Korovin's `iProver` [21,22], instead of generating the resolvent of two clauses c_1 and c_2 with a most general unifier σ, the instances $c_1\sigma$ and $c_2\sigma$ are generated, and after grounding with an arbitrary constant, sent to an incremental Sat-solver. Voronkov's `AVATAR` [32] abstracts variable disjoint components of clauses by propositional variables and sends the abstracted propositional clause set to a Sat-solver. If the Sat-solver returns a model, `AVATAR` tries to refute (through its internal superposition prover) the clause set obtained by unabstracting the model's propositional literals. If it succeeds, a new abstracted propositional clause, consisting of duals of (a subset of) the model's literals is sent to the Sat-solver. This procedure is repeated until the Sat-solver cannot produce a model anymore, in which case the original clause set is unsatisfiable, or the superposition prover fails to refute the clause set unabstracted from the model, in which case the original clause set is either satisfiable or of unknown status. Because `AVATAR` abstracts variable disjoint components of clauses and the unabstracted models produced by the Sat-solver are sets of such isolated components, its procedure is sometimes described as a form of splitting guided by a Sat-solver. However, this form of splitting is substantially different from the splitting discussed in Section 7. While there the resolution or superposition prover generates a refutation of $S \cup \{\Gamma_i^k\}$ for each split component Γ_i^k of a single clause c^k of the form $\Gamma_1^k \vee \ldots \vee \Gamma_n^k$, in `AVATAR` the superposition prover has to find a refutation for a set of split components $\{\Gamma_{i_1}^{j^1}, \ldots, \Gamma_{i_m}^{j^m}\}$ stemming from different clauses c^{j_k} (for $1 \leq k \leq m$). Insofar as **CR** might be a better calculus than resolution for proofs produced by Sat-solvers (as discussed in Subsection 4.1), **CR** could be used as the underlying calculus for a proof format for the reasoning performed by the Sat-solver within `AVATAR`, but not for the reasoning performed by its superposition prover. There are at least two ways to combine **CR** within `AVATAR`'s architecture. A first-order prover based on **CR** could be used inside `AVATAR` instead of a Sat-solver; this would eliminate the need for propositional abstraction, and would allow `AVATAR`-style splitting into components that are not necessarily variable disjoint. Alternatively, `AVATAR`'s superposition prover could be replaced by a **CR** prover; this would have the aesthetic advantage that the Sat-solver and the **CR** prover would share the same style of reasoning. However, to fully replace a superposition prover, **CR** would have to be extended with equational reasoning.

10 Conclusion

The development of the Conflict Resolution calculus **CR** was initially motivated by the recent success of CDCL and by the desire to generalize its main ideas to first-order logic. However, **CR** can also be seen as the convergence of two ideas that actually precede CDCL by several decades. The first one is the assumption mechanism introduced by Gentzen [19] in his natural deduction calculus. The second one is Robinson's generalization of the resolution rule to first-order logic through unification [28]. **CR** extends resolution as natural deduction extends Hilbert-style proof systems: decision literals are essentially assumptions, and conflict driven clause learning corresponds to (several applications of) natural deduction's implication introduction rule. And whereas Robinson used unification to generalize resolution, **CR** uses unification to generalize conflict-driven clause learning.

From a historical perspective, what we are seeing today is similar to what happened between 1960 and 1965. In 1960, Davis and Putnam [16] defined the *propositional* resolution rule, which can be regarded as an efficient machine-oriented variant of modus ponens (implication elimination). The first-order case was then handled by *grounding/instantiating* the first-order problem and using the propositional resolution rule. In 1965, Robinson's direct generalization [28] of the resolution rule to the first-order case enabled a breakthrough in first-order automated theorem proving. Nowadays, we have a powerful *propositional* conflict driven clause learning rule, which can be regarded as an efficient machine-oriented variant of implication introduction. The first-order case is being handled by essentially grounding/instantiating the problem in various ways and using the propositional rule. If history repeats itself, we might see another breakthrough when clause learning is directly lifted to the first-order case through unification, as done in the **CR** calculus proposed here.

A well-defined proof system is just a first step towards the development of a proof search procedure that could be implemented as an effective theorem prover. There is much more to the effectiveness of a modern Sat-solver than just the ideas of decision literals, conflict-driven clause learning and unit-propagation. Sat-solvers use restarts, strategies for selecting decision literals and data-structures that allow efficient unit-propagation, fast conflict graph analysis and fast backtracking. Adapting these proof search strategies and implementation techniques to the Conflict Resolution calculus **CR** is beyond the scope of this paper, but is a crucial direction for future work.

Another essential direction for further development would be the extension of **CR** with a special treatment of equality, since equality is a very common and important relation for applications and it is well-known that naive automated reasoning with axiomatizations of equality theory is ineffective. A seemingly straightforward approach would be to add (refinements of) the paramodulation rule to **CR**; alternatively, recent ongoing works [20] generalizing congruence closure algorithms to the first-order case could enable a possibly more effective approach, similar to what is done by SMT-solvers in the propositional case.

Acknowledgements Bruno is grateful to Pascal Fontaine, who supervised him during his first post-doc in 2010, providing a great opportunity for him to learn some of the essential ideas behind current Sat-solvers. We are thankful to Peter Baumgartner, who shared his experience in model evolution and other related methods, when we discussed the idea of **CR** in May 2015. Bruno would also like to thank Hans de Nivelle for discussions about the `Geo` prover at CADE 2015. We thank Pascal Fontaine and the anonymous reviewers for their feedback on an earlier version of this paper. This research was partially funded by the Australian Government through the Australian Research Council.

References

1. Gábor Alagi and Christoph Weidenbach. "Non-Redundant Clause Learning". In: *FroCoS* (2015), pp. 69–84.
2. Evert W. Beth. "Semantic Entailment and Formal Derivability". In: *Mededelingen van de Koninklijke Nederlandse Akademie van Wetenschappen, Afdeling Letterkunde* 18.13 (1955), pp. 309–342.
3. Leo Bachmair and Harald Ganzinger. "Completion of First-Order Clauses with Equality by Strict Superposition (Extended Abstract)". In: *2nd International Workshop Conditional and Typed Rewriting Systems*, LNCS 516, Springer (1990), pp. 162–180.
4. Leo Bachmair and Harald Ganzinger. "Rewrite-based Equational Theorem Proving with Selection and Simplification". In: *Journal of Logic and Computation* 4.3 (1994), pp. 217–247.
5. Peter Baumgartner. "A First Order Davis-Putnam-Longeman-Loveland Procedure". In: *Proceedings of the 17th International Conference on Automated Deduction (CADE)* (2000), pp. 200–219.
6. Peter Baumgartner. "Model Evolution Based Theorem Proving". In: *IEEE Inteligent Systems* 29(1) (2014), pp. 4–10.

7. Peter Baumgartner and Cesare Tinelli. “The Model Evolution Calculus”. In: *CADE* (2003), pp. 350–364.
8. Peter Baumgartner, Alexander Fuchs and Cesare Tinelli. “Lemma Learning in the Model Evolution Calculus”. In: *LPAR* (2006), pp. 572–586.
9. Armin Biere. “Picosat Essentials”. In: *Journal on Satisfiability, Boolean Modelling and Computation (JSAT)* (2008).
10. Maria Paola Bonacina and David A. Plaisted. “SGGS theorem proving: an exposition.” In: *Notes of the Fourth Workshop on Practical Aspects in Automated Reasoning (PAAR), Seventh International Joint Conference on Automated Reasoning (IJCAR) and Sixth Federated Logic Conference (FLoC), Vienna, Austria, July 2014. EasyChair Proceedings in Computing (EPiC)* (2014), pp. 1-14.
11. Maria Paola Bonacina and David A. Plaisted. “Semantically-Guided Goal-Sensitive Reasoning: Inference System and Completeness”. In: *Journal of Automated Reasoning* (2016), pp. 1–54.
12. “Semantically-Guided Goal-Sensitive Reasoning: Model Representation”. In: *Journal of Automated Reasoning* 56 (2) (2016), pp. 113–141.
13. Chad E. Brown. “Satallax: An Automatic Higher-Order Prover”. In: *IJCAR* (2012), pp. 111–117.
14. Chad E. Brown. “Reducing Higher-Order Theorem Proving to a Sequence of SAT Problems”. In: *Journal of Automated Reasoning* (2013), pp. 57–77.
15. Koen Claessen. “The Anatomy of Equinox – An Extensible Automated Reasoning Tool for First-Order Logic and Beyond (Talk Abstract)”. In: *Proceedings of the 23rd International Conference on Automated Deduction (CADE-23)* (2011), pp. 1–3.
16. Martin Davis and Hilary Putnam. “A Computing Procedure for Quantification Theory”. In: *Journal of the ACM* 7 (1960), pp. 201–215.
17. Martin Davis, George Logemann and Donald Loveland. “A Machine Program for Theorem Proving”. In: *Communications of the ACM* 5(7) (1962), pp. 394–397.
18. Hans de Nivelle and Jia Meng. “Geometric Resolution: A Proof Procedure Based on Finite Model Search”. In: *3rd International Joint Conference on Automated Reasoning (IJCAR)* (2006), pp. 303–317.
19. Gerhard Gentzen. “Untersuchungen über das logische Schließen I & II”. In: *Mathematische Zeitschrift* 39.1 (1935), pp. 176–210 & 405–431.
20. Haniel Barbosa and Pascal Fontaine. “Congruence Closure with Free Variables (Work in Progress)”. In: *2nd International Workshop on Quantification* (2015).
21. Konstantin Korovin. “iProver - An Instantiation-Based Theorem Prover for First-Order Logic (System Description)”. In: *International Joint Conference on Automated Reasoning (IJCAR)* (2008), pp. 292–298.
22. Konstantin Korovin. “Inst-Gen - A Modular Approach to Instantiation-Based Automated Reasoning”. In: *Programming Logics* (2013), pp. 239–270.
23. João Marques-Silva and K.A. Sakallah. “GRASP: A New Search Algorithm for Satisfiability”. In: *International Conference on Computer-Aided Design* (1996), pp. 220 – 227.
24. João Marques-Silva, Ines Lynce and Sharad Malik. “Conflict-Driven Clause Learning SAT Solvers”. In: *Handbook of Satisfiability* (2008), pp. 127 – 149.
25. J. McCharen, R. Overbeek and L. Wos. “Complexity and Related Enhancements for Automated Theorem-Proving Programs”. In: *Computers and Mathematics with Applications* 2 (1976), pp. 1–16.
26. W. McCune. “Prover9Manual” (2006).
27. Alexandre Riazanov and Andrei Voronkov. “The Design and Implementation of VAMPIRE”. In: *AI Communications* 15(2-3)(2002), pp. 91–110.
28. John Alan Robinson. “A Machine-Oriented Logic Based on the Resolution Principle”. In: *Journal of the ACM* 12.1 (1965), pp. 23–41.
29. George Robinson and Larry Wos. “Paramodulation and Theorem-Proving in First-Order Thories with Equality”. In: *Machine Intelligence* 4 (1969), pp. 135–150.
30. Stephan Schultz. “System Description: E 1.8”. In: *LPAR* (2013), pp. 735–743.
31. Geoff Sutcliffe. “The TPTP Problem Library and Associated Infrastructure: The FOF and CNF Parts, v3.5.0”. In: *Journal of Automated Reasoning* 43.4 (2009), pp. 337–362.
32. Andrei Voronkov. “AVATAR: The Architecture for First-Order Theorem Provers”. In: *CAV* (2014), pp. 696–710.

33. Uwe Waldmann. “Superposition”. In: *Encyclopedia of Proof Systems* (2015).
34. Uwe Waldmann. “Saturation with Redundancy”. In: *Encyclopedia of Proof Systems* (2015).
35. Christoph Weidenbach. “Combining Superposition, Sorts and Splitting”. In: *Handbook of Automated Reasoning* (2001), pp. 1965–2013.
36. Christoph Weidenbach. “The Theory of SPASS Version 2.0”. In: *SPASS 2.0 Documentation.*
37. Christoph Weidenbach, Dilyana Dimova, Arnaud Fietzke, Rohit Kumar, Martin Suda, Patrick Wischnewski. “SPASS Version 3.5”. In: *CADE* (2009), pp. 140–145.
38. Nathan Wetzler, Marijn Heule and Warren A. Hunt Jr. “DRAT-trim: Efficient Checking and Trimming Using Expressive Clausal Proofs”. In: *SAT* (2014), pp. 422–429.
39. Lintao Zhang, Conor F. Madigan, Matthew H. Moskewicz, Sharad Malik. “Efficient Conflict Driven Learning in a Boolean Satisfiability Solver”. In: *International Conference on Computer-Aided Design* (2001), pp. 279–285.

Appendix - Classical Natural Deduction

A typical non-clausal natural deduction calculus for quantified minimal logic extended with a classical rule for double negation elimination is shown in Fig. 11.

Implication Elimination (Modus Ponens):

$$\frac{A \quad A \to B}{B}\ \to_E$$

Implication Introduction:

$$\begin{array}{c}[A]^i\\ \vdots\\ B\\ \hline A \to B\end{array}\ \to_I^i$$

Universal Quantification Elimination:

$$\frac{A}{A\{x\backslash t\}}\ \forall_E$$

Universal Quantification Introduction:

$$\frac{A\{x\backslash \alpha\}}{A}\ \forall_I$$

α must be an *eigen-variable*:
it should occur neither in A nor in any undischarged assumption.

Double Negation Elimination:

$$\frac{(A \to \bot) \to \bot}{A}\ \dot{\neg}\dot{\neg}_E$$

Fig. 11: A Non-Clausal Natural Deduction Calculus