EPiC Series in Computing

Volume 46, 2017, Pages 65–84

LPAR-21. 21st International Conference on Logic for Programming, Artificial Intelligence and Reasoning

Towards a Semantics of Unsatisfiability Proofs with Inprocessing

Tobias Philipp[1] and Adrián Rebola-Pardo[2]

[1] International Center for Computational Logic
Technische Universität Dresden, 01062 Dresden, Germany
tobias.philipp@tu-dresden.de
[2] TU Wien (Austria)
arebolap@forsyte.at

Abstract

Delete Resolution Asymmetric Tautology (DRAT) proofs have become a *de facto* standard to certify unsatisfiability results from SAT solvers with inprocessing. However, DRAT shows behaviors notably different from other proof systems: DRAT inferences are non-monotonic, and clauses that are not consequences of the premises can be derived. In this paper, we clarify some discrepancies on the notions of reverse unit propagation (RUP) clauses and asymmetric tautologies (AT), and furthermore develop the concept of resolution consequences. This allows us to present an intuitive explanation of RAT in terms of permissive definitions. We prove that a formula derived using RATs can be stratified into clause sets depending on which definitions they require, which give a strong invariant along RAT proofs. We furthermore study its interaction with clause deletion, characterizing DRAT derivability as satisfiability-preservation.

1 Introduction

Over recent decades, SAT solvers have become powerful engines able to solve instances with millions of variables. One of the main reasons for this efficiency leap was the extension of the conflict-driven clause learning (CDCL) algorithm [38] by a number of inprocessing techniques [27], such as clause elimination [12, 3, 2, 26], variable elimination [46, 11], bounded variable addition [33], cardinality resolution [6], symmetry breaking [8, 1] and parity reasoning [44, 45, 31, 32]. These techniques modify the CNF formula in the SAT solver and replace it by a satisfiability-equivalent one, and often semantic equivalence is not preserved.

Due to concerns on reliability of SAT solvers because of possible undiscovered bugs [7, 34, 37], different proof formats for expressing unsatisfiability together with proof emitting solvers were developed to ensure the correctness of unsatisfiability results [15, 14, 5, 51, 35, 13, 25]. Efficiently generating and checking proofs required carefully designed proof systems and checkers [19].

In 2003, Goldberg and Novikov introduced the *Reverse Unit Propagation (RUP) proof system*, which is essentially equivalent to resolution [4]. However, the aforementioned inprocessing techniques were difficult or impossible to express in the RUP proof system. For this reason, the

Delete Resolution Asymmetric Tautology (DRAT) proof system was introduced, as well as proof generation methods for several inprocessing techniques [27, 20, 40]. DRAT has become a *de facto* standard as DRAT proof logging was required in the main track of the SAT Competition 2016. Most famously, the 200 TB proof of the Pythagorean Triples Problem was expressed as a DRAT proof [24]. Likewise, other verified decision procedures, such as for deciding satisfiability of pseudo-Boolean formulas, rely on DRAT proofs [42]. Several modifications to DRAT were proposed, such as annotations allowing the efficient checking of proofs in mechanically verified programs [10, 9], thus allowing checking the PTP proof in reasonable time, annotations for parallel SAT solvers [39], as well as changes to the deletion information [19].

To the best of our knowledge, there is not much one can currently do with a DRAT proof, beyond checking and trimming [21]. This is in contrast to RUP proofs, where one can obtain resolution proofs [4] or interpolants [16]. One of the reasons for this is the ability of DRAT to derive clauses that are not consequences of the original premises, which contradicts an assumption at the very core of many proof systems. This is a property shared by a few other proof systems such as extended resolution [47]. Therefore, to obtain interpolants from SAT solvers, one has to generate a RUP proof from the SAT solver by turning some inprocessing techniques off, which makes SAT solving infeasible in some fragments [48, 49, 17].

We argue that a *semantic* understanding of DRAT is required, enabling the construction of interpolants or resolution proofs. In this paper, we first try to clarify some discrepancies on the notions of reverse unit propagation (RUP) clauses and asymmetric tautologies (AT) that have appeared in previous papers. We discuss the implicit semantics of DRAT, understood as semantic invariants preserved throughout DRAT proofs. One such invariant is straightforward, namely satisfiability; however this is a rather weak invariant, since only two equivalence classes exist. We discuss the existence of stronger invariants that could give a better understanding of how DRAT derivations behave. In doing so, a intuition about the concept of RAT arises, giving a clear meaning to the somewhat intricate definition of RAT.

Contributions

1. We discuss how the different definitions and characterizations of RUP and AT in terms of least fixed point computations relate to each other. In particular, we show that equivalences do not hold in some corner cases, and we propose modifications to fix them.

2. We develop *resolution consequences* as generalizations of several redundancy criteria including RAT [27], and present an intuitive explanation for RAT in terms of *permissive definitions*. Using these foundations, we show that a framework using a *stratification* of tree-shaped formulas can be used to model the RAT proof system, which gives a semantic for DRAT proofs without deletion information.

3. We discuss how clause deletion affects this framework. In particular, we show that arbitrary deletion fails to preserve any semantic invariant stronger than satisfiability. However, this is an inconclusive result, since contemporary SAT solvers only delete clauses under specific conditions. We discuss possible variations of DRAT that may allow to stronger invariants.

Structure This paper is structured as follows. Section 2 introduces some basic notations and conventions. Section 3 discusses characterizations and definitions of RUP. In Section 4, a general framework to study proofs with RATs is presented. The behaviour of this framework under clause deletion is discussed in Section 5. Finally, we outline our results in Section 6.

2 Preliminaries

We consider strings over some given alphabet, and use the standard operations concatenation of strings and characters. When a string is given by its sequence of elements, angle brackets $\langle\ \rangle$ are used. We consider the partial order $\leq$ on strings given by prefixes, i.e. $u \leq w$ if there is a string v with $uv = w$.

We consider an infinite set of *propositional variables*. An *interpretation* maps propositional variables to Boolean truth values 0 or 1, denoting false or true respectively. A *literal* is either a variable or its negation; the *complement* of literal l is denoted by $\bar{l}$, and the semantics of literals are defined as usual. Both *clauses* and *cubes* are regarded as sets of literals. Following [30], we assume that clauses and cubes do not contain complementary literals, and define a *resolvent* of two clauses C and D such that there is *exactly* one literal l with $l \in C$ and $\bar{l} \in D$, as $C \otimes_l D = (C \setminus \{l\}) \cup (D \setminus \{\bar{l}\})$. Otherwise, we say that their resolvent does not exist. For space reasons, we denote clauses by juxtaposition, e.g. $xy\bar{z}$. The empty clause is denoted by $\square$. We say that a clause C *subsumes* another clause D if $C \subseteq D$ when considered as sets of literals.

An interpretation I satisfies a clause C if it satisfies some literal in C; and I satisfies a cube Q if it satisfies all literals in Q. A set of clauses F is called a *CNF formula*, and it is satisfied by I whenever I satisfies all clauses in F. Dually, a set of cubes V is a *DNF formula*, and it is satisfied by I whenever I satisfies some cube in V. By abuse of notation, we may denote by $\overline{C}$ either the cube obtained by complementing all literals in C, or the CNF formula obtained by collecting the unit clauses containing such literals; the choice should be clear from the context.

Given an interpretation I and a literal l, the interpretation $I + l$ is defined by:

$$(I + l)(l) = 1 \qquad (I + l)(\bar{l}) = 0 \qquad (I + l)(k) = I(k)$$

where k is any literal other than l or $\bar{l}$. Entailment, satisfiability and semantical equivalence are defined as usual. Given two CNF formulas F and G, we say that F *satisfiability-entails* G, denoted by $F \models^{\text{sat}} G$, if G is satisfiable or F is unsatisfiable; and that F is *satisfiability-equivalent* to G, denoted by $F \equiv^{\text{sat}} G$ if $F \models^{\text{sat}} G$ and $G \models^{\text{sat}} F$.

2.1 RUPs and RATs

The DRAT proof system relies on the notion of asymmetric tautology (AT) or reverse unit propagation (RUP), which are considered as synonyms. Slightly different definitions of *asymmetric tautologies (AT)* have been introduced in [18, 50]. We take here a slight variation of the definition of RUP that appeared in a recent paper by Heule [19], where DRAT is presented in detail; the variation is introduced only to make unit propagation irreflexive. *Unit constraint propagation* is defined as a binary relation on CNF formulas, where $F \xrightarrow{\text{ucp}} G$ holds whenever F contains a unit clause l, the complement literal $\bar{l}$ occurs in some clause from F, and $G = \{C \setminus \{\bar{l}\} \mid C \in F\}$. We denote the reflexive transitive closure of $\xrightarrow{\text{ucp}}$ by $\xrightarrow{\star}_{\text{ucp}}$. A CNF formula F is $\xrightarrow{\text{ucp}}$*-irreducible* if there is no CNF formula G with $F \xrightarrow{\text{ucp}} G$. A CNF formula G is called a $\xrightarrow{\text{ucp}}$*-normal form* of another CNF formula F if $F \xrightarrow{\star}_{\text{ucp}} G$ and G is $\xrightarrow{\text{ucp}}$-irreducible.

A clause C is called a *reverse unit propagation* (RUP) in a CNF formula F whenever the CNF formula $F \cup \overline{C}$ has a $\xrightarrow{\text{ucp}}$-normal form containing $\square$. Furthermore, C is a *resolution asymmetric tautology* (RAT) in F upon a literal l whenever the resolvent $C \otimes_l D$ is a RUP in F, for every $D \in F$ for which such resolvent exists.

2.2 DRAT proofs

The RUP inference rule is essentially as powerful as the resolution proof system [4], but adding RAT inferences makes it at least as powerful as extended resolution [27]. This is very convenient from the perspective of SAT solving: state-of-the-art solvers can perform exponentially better than pure CDCL solvers due to the use of inprocessing techniques such as Gaussian elimination, cardinality resolution and symmetry breaking [44, 45, 31, 32, 6, 8, 1]. Expressing such techniques as RUP inferences can lead to much longer proofs, both in theory [17, 48, 49] and in practice [40]. This can be avoided by introducing RATs, due to the exponential gap between resolution and extended resolution [29]. On the other hand, contemporary SAT solvers aggressively delete clauses from their clause databases. Redundant clauses are then deleted to prevent memory depletion and reduce the number of triggered propagations [23]. To avoid the same problems when checking a proof, deletion information must be provided together with the proof.

The *Delete Resolution Asymmetric Tautology (DRAT) proof system* was introduced in [51] to provide a solution for these problems. We model here *DRAT derivations* as strings of CNF formulas, inductively defined as follows:

- If F is a CNF formula, then $\langle F \rangle$ is a DRAT proof of F from F.
- If α is a DRAT proof of G from F, and C is a RUP in G or C is a RAT in G upon some literal l, then $\alpha,(G \cup \{C\})$ is a DRAT proof of $G \cup \{C\}$ from F.
- If α is a DRAT proof of G from F, and C is a clause, then $\alpha,(G \setminus \{C\})$ is a DRAT proof of $G \setminus \{C\}$ from F.

A *DRAT refutation of a CNF formula* F is a DRAT derivation of some formula G from F, where $\square \in G$ holds. Since RUP/RAT introduction and arbitrary clause deletion are satisfiability-preserving operations, and furthermore resolvents are always RUPs, a DRAT refutation of a CNF formula F exists if and only if F is unsatisfiable. Storing DRAT proofs in the format above would be a waste of space, since in every inference step only one clause is inserted or deleted. Instead, DRAT proofs are generated and stored in practice as a string of *clause introduction and clause deletion instructions*, denoted by **i:** C and **d:** C, respectively.

Example 1. Consider the following CNF formulas:

$$F = \{xyz\ ,\ \overline{x}yz\ ,\ x\overline{y}z\ ,\ \overline{x}\overline{y}z\} \qquad G = \{xyz\ ,\ \overline{x}yz\ ,\ x\overline{y}z\ ,\ wy\ ,\ wz\ ,\ \overline{w}z\ ,\ z\}$$

The following is a DRAT derivation of G from F:

$$\langle \textbf{i:}\ wy\ ,\ \textbf{i:}\ \overline{w}z\ ,\ \textbf{i:}\ wz\ ,\ \textbf{i:}\ z\ ,\ \textbf{d:}\ \overline{x}\overline{y}z \rangle \tag{1}$$

Let us briefly justify this. Clause wy can be introduced as a RAT upon w because w does not occur in F. However, by the time clause $\overline{w}z$ is introduced, w is not fresh anymore. Nevertheless, it only has one resolvent with clauses from $F \cup \{wy\}$, namely $\overline{w}z \otimes_{\overline{w}} wy = yz$, which is a RUP in that same formula; thus $\overline{w}z$ is a RAT upon $\overline{w}$. Finally, clauses wz and z are RUPs in their corresponding accumulated formulas, and clause $\overline{x}yz$ can be deleted unconditionally.

The CNF formula F is satisfiable, so we cannot expect a DRAT refutation to exist. We can however construct the following DRAT refutation of the unsatisfiable CNF formula $F \cup \{\overline{z}\}$:

$$\langle \textbf{i:}\ wy\ ,\ \textbf{i:}\ \overline{w}z\ ,\ \textbf{i:}\ wz\ ,\ \textbf{i:}\ z\ ,\ \textbf{d:}\ \overline{x}\overline{y}z\ ,\ \textbf{i:}\ \square \rangle \tag{2}$$

■

DRAT proofs exhibit unusual properties for a proof system, although they are shared by some systems such as extended resolution.

Weak soundness Most proof systems, such as resolution, cutting planes or sequent calculus, have the *strong soundness* property: for every formula G derived by the calculus from F, we have $F \models G$. This is not the case for the DRAT proof system. This does not mean that DRAT is unsound: DRAT proofs are satisfiability-preserving, as shown in Theorem 9 in Section 5.

Example 2. Consider any CNF formula F and a variable x not occurring in F. Then, the unit clause $C = x$ is trivially a RAT in F. However, given any model I of F, the interpretation $I + \overline{x}$ satisfies F but not C, so $F \not\models C$. ■

Non-monotonicity Derivations in typical proof systems are expressed as trees, which is only possible because such proof systems are monotonic: if a G is derived from F, then G can be derived from any formula F' subsuming F. This property does not hold for DRAT proofs.

Example 3. Consider again the situation in Example 2. Both unit clauses $C = x$ and $D = \overline{x}$ are RATs in F. However, D is not a RAT in $F \cup \{C\}$; and conversely C is not a RAT in $F \cup \{D\}$. ■

An effect of the lack of these properties is that deletion information becomes relevant to the proof system itself, as noted in [19, 41]: clauses that could not be derived from a formula might be derived after deleting clauses. This is a strong difference to e.g. resolution proofs, where deletion instructions exist for mere efficiency reasons and can be safely ignored.

3 RUP, Asymmetric Tautologies, and Resolution Chains

The definition of RUP in [19] is the latest occurring in the DRAT literature, but a few others have appeared before in the form of the closely related concept of asymmetric tautology [18, 50]. However, some of these alternatives are not equivalent; by making slight modifications the equivalence can be established.

Reverse Unit Propagation The definition of RUP that we use, given in Section 2.1 and based on unit constraint propagation, does not seem very useful from a practical perspective, since all $\xrightarrow{\text{ucp}}$-normal forms need to be inspected. Notice that, in general, $\xrightarrow{\text{ucp}}$-normal forms are not unique, as shown in Example 4. However, it holds that either all $\xrightarrow{\text{ucp}}$-normal forms contain the empty clause, or none does. Hence, to determine if a clause C is a RUP in F, it suffices to compute one of the $\xrightarrow{\text{ucp}}$-normal forms of $F \cup \overline{C}$, and check if it contains $\square$.

Example 4. Consider the CNF formula $F = \{x, \overline{x}\}$. Then, both CNF formulas $\{\square, \overline{x}\}$ and $\{x, \square\}$ are $\xrightarrow{\text{ucp}}$-normal forms of F. ■

Proposition 1. *For all CNF formulas F, there is some $\xrightarrow{\text{ucp}}$-normal form of F. Furthermore, given two $\xrightarrow{\text{ucp}}$-normal forms G and G' of F, we have $\square \in G$ if and only if $\square \in G'$.*

3.1 Asymmetric Tautologies

One of the most widespread characterizations of RUP is that of *asymmetric tautologies (AT)*, first introduced in [18]. Given a clause C and a CNF formula F, the set of literals $\text{ala}(F, C)$ is defined there as the result of repeating the following *asymmetric literal addition* operation until fixpoint: if there exist literals $l_1, \ldots, l_n, k$ such that $\{l_1, \ldots, l_n\} \subseteq C$ and the clause $l_1 \ldots l_n k$ is

in $F \setminus \{C\}$, then let $C \coloneqq C \cup \{\overline{k}\}$. A clause C is then defined to be an AT in F if $\text{ala}(F, C)$ contains a pair of complementary literals.

This definition has a minor issue: $\text{ala}(F, C)$ may have several fixpoints, and they need not agree on having complementary literals. This problem arises because asymmetric literal addition is a nondeterministic operation (hence it is modeled by a non-functional binary relation), and the notion of fixpoint is only well-defined for mappings, i.e. deterministic operations. If we extend the notion of fixpoints to binary relations by saying that x is a fixpoint for R whenever $(x, x) \in R$, the problem becomes apparent:

Example 5. Let $F = \{x\ ,\ \overline{x}y\}$ and $C = y$. The tables below illustrate the application of the ala operation, and the existence of two fixpoints where one contains complementary literals and the other one does not.

C	$F \setminus \{C\}$	considered clause	result
y	F	$\overline{x}y$	xy
xy	F	$\overline{x}y$	xy

C	$F \setminus \{C\}$	considered clause	result
y	F	$\overline{x}y$	xy
xy	F	x	$x\overline{x}y$
$x\overline{x}y$	F	x	$x\overline{x}y$

■

In [36], a fixpoint definition was proposed, based on a function ala which is defined for a set of literals L and a CNF formula F as

$$\text{ala}_F(L) = L \cup \{\overline{l} \mid \text{there is a clause } C \in F \text{ with } l \in C \text{ and } C \setminus \{l\} \subseteq L\}$$

The justification for this definition is that $\text{ala}_F(L)$ collects all possible applications of the asymmetric literal addition operation to the set of literals L. This allows us to turn the (relational) fixpoint definition from [18] into an actual (functional) fixpoint definition. Note that our $\text{ala}_F(L)$ function differs from the $\text{ala}(F, L)$ defined in [23], insofar as ours represent a single computation of all *simultaneously* applicable asymmetric literal additions; furthermore, we do not consider the removal of C from F in the definition of ala_F.

Given a set of literals L, let us consider the set Lit_L which contains all literal sets K with $L \subseteq K$. Observe that ala_F is monotonic, which implies that it can be viewed as a function $\text{Lit}_L \longrightarrow \text{Lit}_L$ for every set of literals L. Furthermore, Lit_L is a complete lattice with minimum element L. We can then show the following:

Proposition 2. *For every set of literals L, the function $\text{ala}_F \colon \text{Lit}_L \longrightarrow \text{Lit}_L$ has a least fixed point $\text{lfp}_L(\text{ala}_F)$. Furthermore, it can be computed as $\text{lfp}_L(\text{ala}_F) = \text{ala}_F \uparrow \omega(L)$, i.e. the set of literals obtained by iteratively applying ala_F to L until fixpoint.*

This allows us to redefine ATs: a clause C is an AT in a CNF formula F whenever $\text{lfp}_C(\text{ala}_F)$ contains a pair of complementary literals. This solves the problem exposed in Example 5. Nevertheless, even with this redefinition, ATs do not completely characterize RUPs; but the discrepancy only occurs when the empty clause occurs in the CNF formula. Although the issue has been previously reported [23], ATs are often cited as equivalent to RUP [20]. As far as DRAT proof generation and checking is concerned, the confusion between ATs and RUPs is not very relevant, since DRAT proofs are not checked after the empty clause has been found.

Example 6. Let $F = \{\square\}$ and $C = x$. Then, C is a RUP in F, but it is not an AT in F, since $\text{lfp}_C(\text{ala}_F) = \{x\}$. ■

Theorem 1. *Let F be a CNF formula and C be a clause. Then, C is a RUP in F if and only if C is an AT in F or $\square \in F$.*

3.2 Resolution Chains

A related notion is that of *trivial resolution chains* (TVR), a notion introduced in [13, 4]. A clause C is derived by a TVR from a CNF formula F if C can be obtained as a iterated resolvent $C_0 \otimes_{l_1} C_1 \otimes_{l_2} \cdots \otimes_{l_n} C_n$ of clauses $C_i \in F$, and the literals $l_1, \ldots, l_n$ are all distinct.

TVR-derived clauses are sometimes understood as a characterization of RUP clauses, e.g. [16]. Clauses learnt by a purely CDCL SAT solver are all derived by TVR [4], and so a RUP proof generated by such a solver would contain only TVR-derived clauses, which are all RUPs. Nevertheless, it is not true that all RUPs can be derived as TVRs. Furthermore, state-of-the-art solvers feature many inprocessing techniques besides CDCL solving, and generated proofs may contain RUPs that do not correspond to TVRs.

Example 7. Let $F = \{x\}$ and $C = xy$. Then, C is a RUP in F, but it cannot be derived by a TVR from F. ■

On the other hand, a proof-theoretical characterization of RUP is very desirable, since it enables the use of proof manipulation techniques and interpolation methods.

Example 7 implies that resolution alone cannot express the full power of RUP. This is due to resolution being only *weakly complete*: whenever F is unsatisfiable, resolution can derive $\square$, but it cannot derive every consequence of F. However, if $F \models C$, then a subclause $D \subseteq C$ can always be derived by resolution [28]; in other words, subsumption is necessary to model RUP.

We give a characterization in terms of subsumption and *self-subsuming resolution* [11]. A resolution operation $C \otimes_l D$ is self-subsuming whenever $D \setminus \{\bar{l}\} \subseteq C$; in this case, the resolvent is $C \setminus \{l\}$. A *subsumed, self-subsuming resolution* (SSSR) proof from a CNF formula F is a proof of the following form:

$$\cfrac{\cfrac{\cfrac{\cfrac{C_0}{D_0}\,\text{subsumption} \quad C_1}{D_1}\,\text{SSR} \quad C_2}{\ddots}\,\text{SSR}}{D_n}\,\text{SSR}$$

where all premises C_i occur in F, the SSR inferences are self-subsuming resolution inferences (i.e. $D_i \subseteq D_{i-1}$), and D_0 is obtained from C_0 by a subsumption inference. The number of self-subsuming resolutions is arbitrary, possibly zero. A clause C is said to be derivable by SSSR from F if there is a SSSR proof as above from F with $D_n = C$. The following result characterizes RUPs as SSSR-derivable clauses. It is important to observe that this characterization fails for tautological clauses, which in general cannot be derived by SSSR.

Theorem 2. *Let F be a CNF formula and C be a clause. Then, C is a RUP in F if and only if C is derivable by SSSR in F.*

4 Resolution Asymmetric Tautologies as Definitions

The introduction of RAT clauses preserves satisfiability, but not the semantics of the original formula [27]. In this section, we present properties more specific than satisfiability that are preserved by RAT introduction. We furthermore argue that the syntax of propositional logic fails to express these invariants, and we propose a stronger syntax that reflects the properties of derived clauses when these are not consequences.

4.1 Latent inferences

Following [27], we revisit RATs as a particular case of a general phenomenon, which we call *latent inferences*. Let $\vdash$ be a binary relation between CNF formulas and clauses. We say that $\vdash$ is a *sound inference* whenever $F \vdash C$ implies $F \models C$ for every CNF formula F and clause C. Every sound inference gives rise to a latent inference. If $\vdash$ is sound, then we say that C *latently follows* by $\vdash$ from F whenever there is a literal $l \in C$ satisfying the following: every resolvent $C \otimes_l D$ with $D \in F$ satisfies then $F \vdash C \otimes_l D$. Then we say that C *latently follows* by $\vdash$ from F *upon* l. We call this property the *latent inference* of $\vdash$. In [27], it is called "resolution-$\vdash$"; we propose renaming them to "latent inferences" to avoid overloading the concept of resolution.

Observe that a clause C is a RAT in F upon l whenever C latently follows by RUP from F upon l. We can at this point define another two latent inferences for the extreme cases where $\vdash$ is either $\emptyset$ (i.e. the empty binary relation between CNF formulas and clauses) or $\models$; they both are straightforward sound relations.

Blocked clauses A clause C is said to be a *blocked clause* [30] in a CNF formula F upon a literal l whenever C follows latently by $\emptyset$ from F upon l. In this case, C is blocked in F upon l if and only if C cannot be resolved with any clause in F upon l. Blocked clauses were introduced in [30]. In papers where tautologies are allowed as clauses [27], blocked clauses are defined by requiring every resolvent as above to be a tautology.

Resolution consequences A clause C is said to be a *resolution consequence (RC)* in a CNF formula F upon a literal l whenever C follows latently by $\models$ from F upon l. Notice that, whenever a clause C follows latently by some sound inference $\vdash$ from F upon l, then C is a resolution consequence in F upon l; in particular, all RATs in F upon l are resolution consequences in F upon l. Resolution consequences are a novel notion, whose utility will become apparent in Section 4.3 as the invariant preserved by all latent inferences.

The following result allows the use of latent inferences as satisfiability-preserving inferences in a proof system. Thus, if a proof derives the unsatisfiable clause from a CNF formula F, then we can conclude that F is unsatisfiable, although the intermediate lemmas in the proof may not entailed by F.

Theorem 3. *Let $\vdash$ be a sound inference, F a CNF formula and C a clause that follows latently by $\vdash$ from F upon some literal l. Then, $F \equiv^{\text{sat}} F \cup \{C\}$.*

The proof of this result was published in [27], although the claim proved there is weaker. We provide a sketch of the proof here. Showing $F \cup \{C\} \overset{\text{sat}}{\models} F$ is straightforward; for the converse, take any interpretation I satisfying F. If I satisfies C, there is nothing to do; if it falsifies C, then one can show that $I + l$ satisfies $F \cup \{C\}$.

4.2 A Semantic Account of Latent Inferences

From the definition of latent inferences and the proof of Theorem 3, the intutive meaning of introducing a clause that follows latently is not clear. It has been noted [30] that definitions introduced in extended resolution are simulated by blocked clause introduction. However, blocked clauses are a somewhat degenerate case of latent inferences, where only the syntax of F is involved (no resolvent between C and clauses from F must exist upon l), whereas general latent inferences also depend on the semantics of F (every resolvent must be implied by F). We first explain the case of blocked clauses and then extend it to the most general case of resolution consequences.

Definition introduction as blocked clause introduction Consider an arbitrary CNF formula F and a variable x not occurring in F. We aim to introduce extra clauses to define x as the truth value of the propositional formula $\varphi = (a \oplus b) \wedge (b \vee \overline{c})$. By transforming both φ and $\neg\varphi$ into DNF we obtain:

$$\varphi = (a \wedge \overline{b} \wedge \overline{c}) \vee (\overline{a} \wedge b) \qquad \neg\varphi = (\overline{a} \wedge \overline{b}) \vee (a \wedge b) \vee (\overline{b} \wedge c)$$

We can obtain the clauses to be introduced as the CNF translations of $\varphi \to x$ and $\neg\varphi \to \overline{x}$:

$$a \wedge \overline{b} \wedge \overline{c} \to x \qquad \overline{a} \wedge b \to x \qquad \overline{a} \wedge \overline{b} \to \overline{x} \qquad a \wedge b \to \overline{x} \qquad \overline{b} \wedge c \to \overline{x} \tag{3}$$

or equivalently as the CNF formula $\left\{\overline{a}bcx\ ,\ a\overline{b}x\ ,\ ab\overline{x}\ ,\ \overline{a}\overline{b}\overline{x}\ ,\ b\overline{c}\overline{x}\right\}$. One can see that no resolvent upon x can be obtained from this clause set (alternatively, all resolvents upon x are tautologies). This generalizes to the following result:

Proposition 3. *Let φ be a propositional formula. Consider DNF formulae F and G semantically equivalent to φ and $\neg\varphi$ respectively. Then, for every pair of cubes $Q \in F$ and $R \in G$ there is a literal l such that $l \in Q$ and $\overline{l} \in R$. As a consequence, the clauses $x \vee \overline{Q}$ and $\overline{x} \vee \overline{R}$ cannot be resolved upon x (or, if tautological clauses are allowed, all such resolvents are tautological).*

What this result shows is that clauses from 3 can be introduced one by one as blocked clauses upon x or $\overline{x}$ in any order. In fact, blocked clause introduction allows to introduce *only some* of the clauses, leading to a partial definition: x should be true or false under some conditions, and be otherwise unconstrained.

Corollary 1. *Let F be a CNF formula and x be a variable not occurring in F. Consider a propositional formula φ and DNF formulae G and H semantically equivalent to φ and $\neg\varphi$. Define the CNF formula*

$$F' = \{x \vee \overline{Q} \mid Q \in G\} \cup \{\overline{x} \vee \overline{Q} \mid Q \in H\}$$

and pick some (possibly all) clauses $\{C_1, \ldots, C_n\} \subseteq F'$. Then, for every $1 \leq i \leq n$, the clause C_i is blocked upon x or $\overline{x}$ in $F \cup \{C_1, \ldots, C_{i-1}\}$.

Resolution consequences as definition refinements Resolution consequences are the most general setting of latent inferences and we show that resolution consequences model precisely the notion of *permissive definitions*. Permissive definitions allow features that are unattainable in traditional definitions, interpreted as biimplications $x \leftrightarrow \varphi$ where x is a fresh variable and φ does not contain x. Consider a CNF formula $F = \{\overline{x}\,\overline{y}\}$, and assume we introduce a permissive definition:

$$w \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } x \wedge z \\ 1 & \text{if } y \wedge z \\ \text{don't care} & \text{otherwise} \end{cases} \tag{4}$$

This definition has two *nonstandard* features; we argue that this is still acceptable as a definition. The most obvious rarity is the use of a "otherwise, don't care" value, i.e. if none of the previous conditions holds, then the value of w is unconstrained. As it turns out, this is a very common idea, not only in mathematics, but even in SAT solving itself: the "one-way" implications in the Plaisted-Greenenbaum CNF translation [43] make extensive use of this feature. The second anomaly is that, apparently, w could take two different values if $x \wedge y \wedge z$ is satisfied. However,

it must be noted that, although in general (4) would not be a correct definition, it is in this case, for F forbids the condition above. That is, (4) is a *non-trivially correct* definition.

A consequence of having unconstrained values in a definition is that definitions themselves can be refined. In particular, clauses expressing the definition should be allowed to be introduced one by one. Therefore, the notion of permissive definition should be formalized as a criterion to when is a particular definition refinement allowed. We argue that this criterion corresponds precisely to that of resolution consequences.

Consider a CNF formula F and a variable x. Clauses involving variable x can always be interpreted as definition rules of the form $x \leftarrow Q$ or $\overline{x} \leftarrow Q$, where Q is a cube not containing variable x. Then, we can collect all clauses involving x in a piecewise definition of the form

$$x \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if either } Q_1, \text{ or } Q_2, \ \ldots, \text{ or } Q_n \\ 1 & \text{if either } R_1, \text{ or } R_2, \ \ldots, \text{ or } R_m \\ \text{don't care} & \text{otherwise} \end{cases}$$

where the Q_i and R_i are cubes not involving x. Assume we want to further refine this definition introducing a new definition rule. Without loss of generality we can assume we introduce a rule "1 if R_{m+1}", i.e. we introduce the clause $x \vee \overline{R_{m+1}}$. To ensure correctness, we must check that it does not conflict with other rules of the definition; that is, that there is no interpretation satisfying simultaneously R_{m+1} and one of the Q_i. In fact, it suffices to check that this does not happen for any interpretation satisfying F. Hence, the property to be checked is $F \models \neg(R_{m+1} \wedge Q_i)$ for every $1 \leq i \leq n$. Finally, observe that this is equivalent to check that $F \models (x \vee \overline{R_{m+1}}) \otimes_x (\overline{x} \vee \overline{Q_i})$ for every $1 \leq i \leq n$, which is precisely the condition for the desired clause $x \vee \overline{R_{m+1}}$ to be introduced as a resolution consequence. Thus, resolution consequences can be interpreted precisely as definition refinements.

A semantic notion of definition refinements Let us consider now a CNF formula F and a RC C in F upon a literal l. The proof of Theorem 3 presents a method to construct the models of $F \cup \{C\}$ from the models of F by selectively changing the truth value of l. We abstract this property to obtain an invariant preserved by latent inferences; later we show this characterizes RCs. We consider *mutation rules* of the form $l :\!- Q$ where l is a literal and Q is a cube containing $\overline{l}$. Given an interpretation I, we define the *mutation of I by the rule $l :\!- Q$* as the interpretation

$$I \lhd l :\!- Q = \begin{cases} I & \text{if } I \not\models Q \\ I + l & \text{if } I \models Q \end{cases}$$

Intuitively, mutating an interpretation I by a rule $l :\!- Q$ prevents Q from being satisfied.

Lemma 1. *For all mutation rules $l :\!- Q$ and interpretations I, we have $I \lhd l :\!- Q \models \neg Q$.*

We say that a mutation rule $l :\!- Q$ is a *definition refinement of a formula F* whenever, for all interpretations I satisfying F, the mutation $I \lhd l :\!- Q$ also satisfies F. Thus, an RC C in F upon l will be associated to the mutation rule $l :\!- \overline{C}$, and so the corresponding mutation will satisfy C. The definition refinement condition then ensures that F is satisfied as well by the mutation.

The following result shows that RCs are a syntactic expression of the semantic condition given by definition refinements. The sufficient condition was well known: Biere et al. [27] give an algorithm to reconstruct models for the original formula when clauses are deleted using RAT as a redundancy criterion, where reconstruction is performed by a sequence of mutations. However, the necessary condition is a new result to the best of our knowledge.

Theorem 4. *Let F be a CNF formula, l a literal and C a clause containing l. Then, C is a RC in F upon l if and only if $l :- \overline{C}$ is a definition refinement of F.*

Observe that the characterization above is purely semantical: given two equivalent CNF formulas, they have exactly the same RCs upon a given literal. It is possible to obtain a similar characterization of blocked clauses in a formula F, although it is not a semantic characterization: the equivalence depends on the subsets of F, and clause deletion depends on the syntax of F. In fact, it is impossible to obtain a semantic characterization of blocked clauses, since it is not a semantic property, i.e. there are semantically equivalent CNF formulas F and G such that C is blocked in F but not in G, as shown by Example 8.

Corollary 2. *Let F be a CNF formula, l a literal and C a clause containing l. Then, C is a blocked clause in F upon l if and only if $l :- \overline{C}$ is a definition refinement of every subset $G \subseteq F$.*

Example 8. Consider the semantically equivalent CNF formulas $F = \{\overline{z}, x\overline{y}\}$ and $G = \{x\overline{z}, \overline{x}\overline{z}, x\overline{y}\}$. The clause $\overline{x}y$ is blocked in F upon x, but it is not blocked in G upon l because the resolvent $\overline{x}y \otimes_{\overline{x}} x\overline{z} = y\overline{z}$ exists (or alternatively, because the resolvent is a tautology). This shows that blocked clauses are not a semantic notion, but they do not exclusively depend on the semantics of F. Observe that $\overline{x}y$ is nevertheless a RC of G (i.e. the resolvent $y\overline{z}$ is implied by G). This happens because it is blocked in F, so in particular it is a RC in F. Since RCs *are* a semantic notion, we know that $\overline{x}y$ is an RC in any formula semantically equivalent to F. ■

4.3 Definitional Formulas

The previous section revealed resolution consequences as the semantic invariant preserved by RAT introduction. Nevertheless, the syntax of propositional logic fails to express this semantic perspective on resolution consequences. We now propose a more expressive syntax and an associated semantics that easily admits RAT introduction (in fact, RC introduction) as an equivalence-preserving operation; we will refer to formulas in our syntax as *definitional formulas*. When DRAT proofs are understood as derivations on formulas of our syntax, RAT introduction inferences can be interpreted as sound inferences, in the sense that the derived formula *is a consequence* of the premise in the associated semantics.

Let us reconsider the DRAT refutation (2) from Example 1, which can be summarized as the following resolution proof, where introduced RATs have been highlighted:

$$\dfrac{\dfrac{\dfrac{\dfrac{x\overline{y}z \quad \overline{x}\,\overline{y}z}{\overline{y}z} \quad wy}{wz} \quad \overline{w}z}{z} \quad \overline{z}}{\square}$$

Since clauses wy and $w\overline{z}$ are not consequences of the premise $F = \{xyz\ ,\ \overline{x}yz\ ,\ x\overline{y}z\ ,\ \overline{x}\,\overline{y}z\ ,\ z\}$, in principle any clause derived using them may or may not be entailed by F. However, since DRAT is a satisfiability-preserving proof system, we know that at least $F \models \square$. In fact, z follows already from F, but wz does not. This suggests that some invariant is preserved along the part of the proof containing non-entailed clauses; this invariant turns out to be being a RC.

Theorem 5. *Let F be a CNF formula and C an RC in F upon l. Let D be a clause such that $F \cup \{C\} \models D$. Then, either $F \models D$, or D is a RC in F upon l.*

Observe that the alternatives in Theorem 5 are not exclusive. In fact, whenever $F \models D$ and $l \in D$, the clause D is a RC in F upon l. Checking either of the alternatives is in general coNP-complete. However, the observation above gives a useful criterion: if $F \cup \{C\} \models D$ and $l \notin D$, then $F \models D$. This justifies our choice for the word "latent": if a clause latently follows from F upon l, then every consequence will latently follow from F until, by resolution, it loses the literal l, at which point it will "genuinely" follow.

Theorem 5 implies a separation between clauses that follow from a formula, and clauses that require an additional RC to be entailed. The rest of this section is devoted to formalizing a syntactic and semantic framework that explicitly provides such separation.

We formalize this by tree-shaped constructs called *definitional formulas*. Let us first generalize the notion of interpretation mutation. A *rule path* is a string of mutation rules. We recursively define the *mutation of an interpretation I through a rule path* by

$$I \lhd \langle\rangle = I \qquad\qquad I \lhd (l :\!- Q, \pi) = (I \lhd l :\!- Q) \lhd \pi$$

A *rule tree* is a set of rule paths T closed under prefixes, i.e. whenever $\pi \in T$ and σ is a prefix of π, then $\sigma \in T$ also holds. A *tree formula* is a mapping $\Phi \colon T \longrightarrow \mathrm{Cnf}$ where T is a rule tree and Cnf is the set of all CNF formulas; we will normally declare Φ as a tree formula and refer to T as $\mathrm{Dom}(\Phi)$.

Tree formulas provide stratification between clauses in a formula. Intuitively, clauses in $\Phi(\langle l_1 :\!- Q_1, \ldots, l_n :\!- Q_n\rangle)$ follow from $\Phi(\langle\rangle)$ together with the definition refinements $l_i :\!- Q_i$. In particular, CNF formulas F can be represented as tree formulas Φ over the one-element tree $\{\langle\rangle\}$ with $\Phi(\langle\rangle) = F$.

We now give a semantics of tree formulas. If Φ is a tree formula, then for every path $\pi \in \mathrm{Dom}(\Phi)$ we define the *accumulated formula along* π, denoted by $\Phi{\downarrow}(\pi)$, as $\bigcup_{\sigma \leq \pi} \Phi(\sigma)$. An interpretation I *satisfies* Φ if, for every path $\pi \in \mathrm{Dom}(\Phi)$ we have $I \lhd \pi \models \Phi{\downarrow}(\pi)$. Entailment and equivalence of tree formulas are defined in the usual way: $\Phi \models \Psi$ whenever every model $I \models \Phi$ satisfies $I \models \Psi$; and $\Phi \equiv \Phi$ whenever $\Phi \models \Psi$ and $\Psi \models \Phi$.

We call the formula $\Phi(\langle\rangle)$ the *root formula* of Φ. The root formula $\Phi(\langle\rangle)$ represents the formula on which definitions are introduced. In turn, introduced definitions are represented by the labels of the tree edges; a path on the tree represents a sequence on which definitions are introduced. Tree formulas provide a syntactic separation of clauses in a CNF formula depending on which definitions each clause relies upon; we refer to this separation property as *stratification*. Hence, a clause occurring in $\Phi(\langle l_1 :\!- Q_1, l_2 :\!- Q_2\rangle)$ is intended to be a consequence of the root formula after introducing definitions $l_1 :\!- Q_1$ and $l_2 :\!- Q_2$. A precise formalization of stratification is provided later; the following example is intended to clarify the syntax and semantics of arbitrary tree formulas.

Example 9. Let us consider mutation rules $\delta = (w :\!- \overline{w} \wedge \overline{y})$ and $\varepsilon = \overline{w} :\!- w \wedge \overline{z}$. The set of mutation paths $T = \{\langle\rangle, \langle\delta\rangle, \langle\delta, \varepsilon\rangle\}$ constitutes a rule tree, so a tree formula can be obtained by assigning a CNF formula to every path in T. As an example, we assign:

$$\Phi(\langle\rangle) = \{xyz, \overline{x}yz, x\overline{y}z, \overline{x}\,\overline{y}z\} \qquad \Phi(\langle\delta\rangle) = \{wy, wz\} \qquad \Phi(\langle\delta, \varepsilon\rangle) = \{\overline{w}z, z\}$$

We graphically represent this tree formula as follows:

$$\begin{array}{l} xyz\ ,\ \overline{x}yz \\ x\overline{y}z\ ,\ \overline{x}\,\overline{y}z \end{array} \xrightarrow{\ w :\!- \overline{w} \wedge \overline{y}\ } \begin{array}{l} wy \\ wz \end{array} \xrightarrow{\ \overline{w} :\!- w \wedge \overline{z}\ } \begin{array}{l} \overline{w}z \\ z \end{array}$$

Within this representation of tree formulas, we can compute $\Phi{\downarrow}(\pi)$ by collecting all the clauses along the path π. For example, we have $\Phi{\downarrow}(\langle\delta\rangle) = \{xyz, \overline{x}yz, x\overline{y}z, \overline{x}\,\overline{y}z, wy, wz\}$.

Now consider an interpretation I satisfying literals x, $\overline{y}$, z and $\overline{w}$. Observe that I satisfies $\Phi{\downarrow}(\langle\rangle)$, but it does not satisfy $\Phi{\downarrow}(\langle\delta\rangle)$ or $\Phi{\downarrow}(\langle\delta,\varepsilon\rangle)$. However, I does satisfy Φ, since the requirements that $I \lhd \langle\delta\rangle \models \Phi{\downarrow}(\langle\delta\rangle)$ and $I \lhd \langle\delta,\varepsilon\rangle \models \Phi{\downarrow}(\langle\delta,\varepsilon\rangle)$ are met.

In this example, we define w as true if $\overline{y}$ holds; and false if $\overline{z}$ holds. This is done by first introducing the definition refinement δ together with the clause yw; and then introducing the definition refinement ε together with the clause $z\overline{w}$. It can be checked that δ is indeed a definition refinement of $\Phi{\downarrow}(\langle\rangle)$ (or equivalently, that wy is an RC in $\Phi{\downarrow}(\langle\rangle)$) upon w); and ε is a definition refinement in $\Phi{\downarrow}(\langle\delta\rangle)$ ($z\overline{w}$ is an RC upon $\overline{w}$, respectively). ■

Observe that tree formulas do not provide any mechanism to make sure that definitions are, in some sense, *correct*. That is, our syntax allows us to express situations that go beyond the idea of introducing a definition. In particular, we would like to make sure that the models for the root formula and the models for the formula with definitions are *essentially the same*, only up to differences prescribed by the definitions. This amounts to the following two properties.

Model-preservation along definitions Definitions should not eliminate models of the root formula beyond those lost to the definition itself. That is, given any model for a path of the tree formula, mutating the model along a deeper path should yield a model of the deeper path. We formalize this as follows: given any mutation paths π and σ with $\pi\sigma \in \mathrm{Dom}(\Phi)$, every model $I \models \Phi{\downarrow}(\pi)$ fulfills $I \lhd \sigma \models \Phi{\downarrow}(\pi\sigma)$. In particular, this implies that any model for the root formula can be mutated to satisfy all clauses along any path.

Model-reflection along definitions Definitions should not introduce spurious models that are not obtained by enforcing definitions over a model of the root formula. This is, given any model for a path of the tree formula, this model should be the mutation of some model of a shallower path. We formalize this as follows: given any mutation paths π and σ with $\pi\sigma \in \mathrm{Dom}(\Phi)$, and any model $I \models \Phi{\downarrow}(\pi\sigma)$, there exists a model $J \models \Phi{\downarrow}(\pi)$ such that $J \lhd \sigma = I$.

Model-preservation and -reflection along definitions can be expressed into a single condition. A tree formula Φ is said to be a *definitional formula* if, for every interpretation I and every path $\pi \in \mathrm{Dom}(\Phi)$, whenever $I \models \Phi{\downarrow}(\pi)$, then there is a model $J \models \Phi$ such that $J \lhd \pi = I$. Definitional formulas are intended to implement both the idea of stratification described above and the properties of model-preservation and -reflection. The following result formalizes this.

Proposition 4 (Stratification)**.** *Let Φ be a definitional formula. The following hold:*

1. $\Phi \equiv \Phi(\langle\rangle)$
2. Model-preservation*: for all mutation paths π, σ with $\pi\sigma \in \mathrm{Dom}(\Phi)$, if $I \models \Phi{\downarrow}(\pi)$, then $I \lhd \sigma \models \Phi{\downarrow}(\pi\sigma)$.*
3. Model-reflection*: for all mutation paths π, σ with $\pi\sigma \in \mathrm{Dom}(\Phi)$, if $I \models \Phi{\downarrow}(\pi\sigma)$, then there is some model $J \models \Phi{\downarrow}(\pi)$ with $J \lhd \sigma = I$.*
4. *For all nonempty mutation paths π,$(l :\!- Q) \in \mathrm{Dom}(\Phi)$, the mutation rule $l :\!- Q$ is a definition refinement of $\Phi{\downarrow}(\pi)$, and furthermore $\Phi{\downarrow}(\pi) \cup \{\overline{Q}\} \models \Phi{\downarrow}(\pi$,$(l :\!- Q))$.*

4.4 Modeling RAT Inferences with Definitional Formulas

We now show that definitional formulas, together with tree formula semantics, adequately model both inference and definition refinement — and in particular, RUP and RAT introduction. To

do so, we introduce inference rules on definitional formulas; each rule is accompanied by a result that states that the premise and the conclusion of each inference rule are equivalent, and that the conclusion of each inference is definitional whenever the premise is definitional.

Let us introduce some auxiliary notation. Given a tree formula Φ, a CNF formula F and a path π (not necessarily in Φ), we define the mapping $\Phi[\pi \mapsto F]$ with domain $\mathrm{Dom}(\Phi) \cup \{\pi\}$ such that

$$\Phi[\pi \mapsto F](\sigma) = \begin{cases} F & \text{if } \sigma = \pi \\ \Phi(\sigma) & \text{if } \sigma \neq \pi \end{cases} \qquad \text{for all } \sigma \in \mathrm{Dom}(\Phi) \cup \{\pi\}$$

Observe that this is not necessarily a tree formula, but it will be in all our use cases. If π is a path and C is a clause, we further define:

$$\Phi[\pi \overset{+}{\mapsto} C] = \Phi[\pi \mapsto \Phi(\pi) \cup \{C\}] \qquad \Phi[\pi \overset{-}{\mapsto} C] = \Phi[\pi \mapsto \Phi(\pi) \setminus \{C\}]$$

where $\Phi(\pi)$ is taken to be $\emptyset$ if $\pi \notin \mathrm{Dom}(\Phi)$. Intuitively, $\Phi[\pi \mapsto F]$ replaces the CNF formula at mutation path π in Φ by F; and $\Phi[\pi \overset{+}{\mapsto} C]$ introduces clause C in the CNF formula at mutation path π in Φ; and respectively, $\Phi[\pi \overset{-}{\mapsto} C]$ removes C from it. In all cases, the path π is added to the domain of Φ if it is not yet present.

We now introduce the three inference rules that can be applied to tree formulas. The first inference is consequence introduction: if a clause follows from $\Phi{\downarrow}(\pi)$, then it can be introduced at position π. This inference rule subsumes RUP introduction. The other two inference rules involve the path structure of definitional formulas. Path extension allows to add a new node to the tree formula; this corresponds to RAT (or RC) introduction. On the other hand, clause upgrade moves a clause one level closer to the root. The latter inference does not have a DRAT correspondent; it rather expresses Theorem 5 by allowing entailment of clauses to be recovered after introduction of a RAT. The condition for a clause C to be upgraded through a mutation $l \coloneq Q$ is called symmetry. We say that C is *symmetric w.r.t.* $l \coloneq Q$ whenever, for every interpretation I, if $I \lhd l \coloneq Q \models C$, then $I \models C$.

Theorem 6 (Consequence introduction)**.** *Let Φ be a definitional formula, $\pi \in \mathrm{Dom}(\Phi)$ and C be a clause formula such that $\Phi{\downarrow}(\pi) \models C$. Consider the tree formula $\Psi = \Phi[\pi \overset{+}{\mapsto} C]$. Then, $\Phi \equiv \Psi$, and Ψ is definitional.*

Theorem 7 (Path extension)**.** *Let Φ be a tree formula, $\pi \in \mathrm{Dom}(\Phi)$ and $l \coloneq Q$ be a mutation rule with the following properties:*

1. *$\pi,(l \coloneq Q)$ is not in $\mathrm{Dom}(\Phi)$*
2. *$l \coloneq Q$ is a definition refinement of $\Phi{\downarrow}(\pi)$.*

Consider the tree formula $\Psi = \Phi[\pi \overset{+}{\mapsto} \overline{Q}]$. Then, $\Phi \equiv \Psi$, and Ψ is definitional.

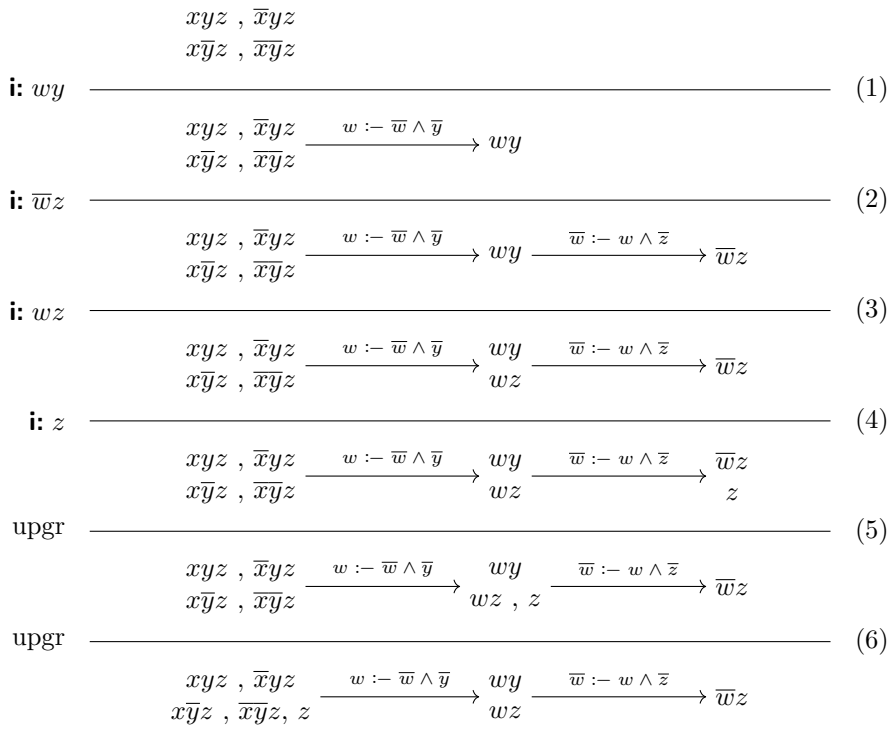
Theorem 8 (Clause upgrade)**.** *Let Φ be a tree formula, $\pi, l \coloneq Q$ be a nonempty mutation path. Let $C \in \Phi(\pi, l \coloneq Q)$ be a symmetric clause w.r.t. $l \coloneq Q$, and define the tree formula $\Psi = \Phi[\pi,(l \coloneq Q) \overset{-}{\mapsto} C][\pi \overset{+}{\mapsto} C]$. Then, $\Phi \equiv \Psi$ and Ψ is definitional.*

Observe that, in general, the preconditions for these inference rules are coNP-hard to check. Nevertheless, the three of them hold for efficiently verifiable cases.

- Theorem 6 can be used to replace $\Phi(\pi)$ by $\Phi(\pi) \cup \{C\}$ when $\Phi{\downarrow}(\pi) \models C$, thus covering RUP introduction.

- Theorem 7 can be used to introduce any RAT C in $\Phi{\downarrow}(\pi)$ upon l by introducing the path $\pi,(l :- \overline{C})$ in the tree.
- Theorem 8 requires symmetry, which holds when neither l nor $\bar{l}$ occur in C.

Example 10. Recall the DRAT proof $\langle$**i:** wy , **i:** $\overline{w}z$, **i:** wz , **i:** z , **d:** $\overline{xy}z\rangle$ from the CNF formula $\{xyz, \overline{x}yz, x\overline{y}z, \overline{xy}z\}$ presented in Example 1. This example shows how our inference rules over definitional formulas model the first four DRAT inferences in the proof; deletion is discussed in Section 5. We first give a schematic overview of the modeling of the DRAT inferences by definitional formula inferences. Each inference is discussed below.



The first two introduction inferences derive clauses wy and $\overline{w}z$ as RATs in the accumulated formula upon w and $\overline{w}$ respectively. This is modeled in inferences (1) and (2) by applying the path extension inference from Theorem 7 twice. Since every RAT is an RC, Theorem 4 guarantees that the rules $w :- \overline{w} \wedge \overline{y}$ and $\overline{w} :- w \wedge \overline{z}$ are definition refinements of their respective accumulated formulas.

The latter two introduction inferences obtain clauses wz and z as RUPs in the accumulated formula, which is simulated in inferences (3) and (4) by consequence introduction from Theorem 6. Here we can see the purpose of stratification. On the one hand, the clause wz does not require the clause $\overline{w}z$ to be derived, so we can introduce it at position $\langle w :- \overline{w} \wedge \overline{y}\rangle$. Introducing it at position $\langle w :- \overline{w} \wedge \overline{y}, \overline{w} :- w \wedge \overline{z}\rangle$ would be allowed too, but it cannot be introduced at position $\langle\rangle$, precisely because they are not consequences of the root formula. On the other hand, deriving z as a RUP requires the use of clause $\overline{w}z$, so it can only be introduced at position $\langle w :- \overline{w} \wedge \overline{y}, \overline{w} :- w \wedge \overline{z}\rangle$.

Observe that clause z actually follows from the root formula alone, although it cannot be derived from it *as a single RUP*. This is problematic from the stratification point of view,

since determining the shallowest path which a clause belongs in is intractable in general. The purpose of clause upgrade from Theorem 8 is to provide a tool to reduce the definitional dependencies of a clause by bringing it closer to the root. Nevertheless, the symmetry condition is straightforwardly met whenever the variable in the head of the mutation rule does not occur in the clause, as discussed above. Because the aforementioned dependencies are not kept track of by DRAT, clause upgrade inferences do not have a DRAT correspondent. In this case, since variable w does not occur in clause z, it can be lifted twice to bring it to the root formula, as shown in inferences (5) and (6).

Finally, observe that the first tree formula in the derivation is straightforwardly definitional, and since every applied inference rule preserves definitional formulas, so is the last one. ■

5 Invariants under Clause Deletion

State-of-the-art SAT solvers frequently delete clauses that are redundant in the current CNF formula. The non-monotonic properties of RC introduction makes clause deletion very relevant: introduced definitions may cease being correct after clauses are deleted. That is, deletion of a clause may turn a definitional formula into a non-definitional one. By virtue of Corollary 2, this does not affect blocked clauses, but other definition refinements may be compromised. Restricting deletion from the current formula F to clauses C that are RATs in the formula $F \setminus \{C\}$ as in [27, 22] does not solve this:

Example 11. Consider again the proof from Example 10, where we had not studied the deletion inference **d:** $\overline{x}\overline{y}z$. Observe that this is a plausible deletion operation, since $\overline{x}\overline{y}z$ is a RAT upon z in the whole CNF formula without the clause, and so it satisfies the conditions to be deleted in a SAT solver. Let us assume that we apply the deletion inference directly after step (4), hence skipping the clause upgrade operations. This turns the tree-formula into a non-definitional one.

$$\begin{array}{l} \begin{array}{lcll} xyz\ ,\ \overline{x}yz & \xrightarrow{w \coloneq \overline{w} \wedge \overline{y}} & wy & \xrightarrow{\overline{w} \coloneq w \wedge \overline{z}} \quad \overline{w}z \\ x\overline{y}z\ ,\ \overline{x}\overline{y}z & & wz & \qquad\qquad\quad z \end{array} \\ \textbf{d: } \overline{x}\overline{y}z \quad \text{---} \\ \begin{array}{lcll} xyz\ ,\ \overline{x}yz & \xrightarrow{w \coloneq \overline{w} \wedge \overline{y}} & wy & \xrightarrow{\overline{w} \coloneq w \wedge \overline{z}} \quad \overline{w}z \\ x\overline{y}z & & wz & \qquad\qquad\quad z \end{array} \end{array}$$

To see that the latter tree formula is non-definitional, let us call the latter formula Φ and consider an interpretation I satisfying x, $\overline{y}$, $\overline{z}$, $\overline{w}$. Then, we have that $I \lhd \langle \delta, \varepsilon \rangle = I$, and it satisfies $\Phi(\langle\rangle)$ but it does not satisfy $\Phi(\langle \delta, \varepsilon \rangle)$. Therefore, models are not preserved in Φ along definitions, so Φ is not definitional. ■

This example shows that stratification cannot be preserved if clauses can be deleted, i.e. separation between root clauses and clauses derived by definitions is only an invariant of DRAT proofs without deletion. A forthright question is whether there is any semantic invariant of DRAT proofs which is stronger than satisfiability. The following result shows there is not:

Theorem 9 (Characterization of DRAT derivability)**.** *Let F and G be two CNF formulas. Then, a DRAT derivation of G from F exists if and only if $F \models^{\text{sat}} G$.*

Proof sketch. The "only if" implication follows from showing that satisfiability is preserved by clause deletion, RUP introduction and RAT introduction. The former two are straightforward, whereas the latter follows from Theorem 3. To show the "if" implication, we distinguish two

cases. If F is unsatisfiable, then there is a proof of a CNF formula F_1 containing $\square$ by completeness of resolution. All clauses from F_1 but $\square$ can be deleted, and so $\{\square\}$ is derived. Then, derive every clause in G by subsumption, which is covered by RUP, and finally delete $\square$ if it is not in G.

Otherwise, if F is satisfiable, then so is G; let I be a model of G. By deleting all clauses in F, the empty CNF formula can be DRAT-derived. Now, for every variable x occurring in G, there is a unique literal $l_x \in \{x, \overline{x}\}$ satisfied by I. Iteratively introduce unit clauses l_x as RATs into the formula for every variable x occurring in G, which can be done because at the point of introduction x is fresh in the accumulated formula. Afterwards, every clause C in G can be derived as a RUP: some literal in C is satisfied by I, which means that some of the l_x occurs in C, or equivalently, C is subsumed by the unit clause l_x, so C is a RUP in the current accumulated formula. Deleting clauses not in G yields a DRAT derivation of G from F. $\square$

The above theorem states that there is no strong semantics preserved along a DRAT derivation, which is due to deletion information. Deletion information was proposed for the RUP format to speed up unit propagation in the checker, but the combination with RAT clauses is known to be problematic: First, while clause deletion could be ignored in DRUP proofs, it cannot be ignored in DRAT as RAT clauses assume that certain clauses do not occur in the formula [19], i.e. RAT introduction is *non-monotonic*. Second, there is a gap between the specification of the DRAT format and one of the most widely used checkers, `drat-trim`, in the sense that the DRAT format allows arbitrary deletion, but `drat-trim` does not apply all of them [19]. This behavior results in incompleteness of `drat-trim` [41]. Third, arbitrary deletion is actually not applied by SAT solvers: Instead, a clause is stored in the redundant clause set at a certain time point it satisfies one of the redundancy criteria, and can be deleted afterwards [27].

We therefore propose that arbitrary deletion should not be considered in combination with RAT. Unfortunately, straightforward restrictions of deletion information do not solve the presented issues: Consider a proof system D*RAT identical to DRAT except for deletion being restricted to RATs in the current formula. Example 11 is still applicable, showing that stratification is not preserved as an invariant.

6 Conclusion

DRAT proofs were developed to guarantee correctness of unsatisfiability results from SAT solvers applying inprocessing techniques. Today, DRAT is a standard in SAT solving, and proofs can be efficiently be generated and checked expressed in this format. Unfortunately, extracting additional information from a DRAT proof, such as interpolants, seems to be difficult, and such issues are conspicuously absent from the literature. We believe that one of the reasons is the lack of understanding of the invariants preserved throughout DRAT proofs, as well as its intricate differences from more standard propositional proof systems.

This paper represents an effort towards a better understanding of DRAT. We analyzed the main pitfalls for each of the three DRAT inference rules. RUP introduction does not show semantic issues, since it is a strongly sound inference. However, several characterizations and definitions have been introduced in the literature, and their properties slightly differ. We presented modifications to make them agree. On the other hand, RAT poses problems from a semantic perspective, since derived clauses are not in general consequences of the premises. We developed the notion of *resolution consequences*, which subsumes RATs. Resolution consequences characterize the idea of introducing a possibly partial, and possibly non-trivially correct

definition, called *permissive definition*. Once this is established, a formula derived using RATs, or RCs for that matter, can be stratified into clause sets depending on which definitions they require. This is done using *definitional formulas*, that maintain both the stratification, and an invariant claiming that the strata correspond to definitions.

Unfortunately, this framework turns out to be incompatible with deletion. Deleting a clause can make a previously introduced definition incorrect, thus disallowing stratification as an invariant throughout DRAT derivations. In fact, we show that unrestricted deletion makes satisfiability the strongest invariant kept on DRAT proofs. We therefore propose not to consider arbitrary deletion information in combination with RAT introduction.

Future work Our main interest to continue this work is on restrictions of DRAT. We know that unrestricted deletion is more powerful than needed for unsatisfiability proofs generated by SAT solvers; and on the other hand our proof of Theorem 9 only works if unrestricted deletion is allowed. Alternatively, Corollary 2 shows that blocked clauses remain blocked after arbitrary clauses have been deleted, and most proof generation methods only rely on blocked clause introduction. This suggests a line of work towards invariants in restrictions of DRAT that still can express proofs generated by state-of-the-art solvers.

Acknowledgments We would like to thank Georg Weissenbacher and Martin Suda for their very valuable comments while writing this paper, and Björn Lellmann, Steffen Hölldobler and Christoph Wernhard for the fruitful discussions about the semantics of RATs, as well as reviewers for their comments and suggestions. This work was supported by the Austrian National Research Network S11403-N23 (RiSE), the LogiCS doctoral program W1255-N23 of the Austrian Science Fund (FWF), the Vienna Science and Technology Fund (WWTF) through grant VRG11-005, Microsoft Research through its PhD Scholarship Programme, and the Graduate Academy TU Dresden.

References

[1] F. A. Aloul, A. Ramani, I. L. Markov, and K. A. Sakallah. Solving difficult SAT instances in the presence of symmetry. In *DAC 2002*, pages 731–736. ACM, 2002.

[2] G. Audemard, J.-M. Lagniez, B. Mazure, and L. Sais. On freezing and reactivating learnt clauses. In K. A. Sakallah and L. Simon, editors, *SAT 2011*, volume 6695 of *LNCS*, pages 188–200, Heidelberg, 2011. Springer.

[3] G. Audemard and L. Simon. Predicting learnt clauses quality in modern SAT solvers. In C. Boutilier, editor, *IJCAI 2009*, pages 399–404, Pasadena, 2009. Morgan Kaufmann Publishers Inc.

[4] P. Beame, H. Kautz, and A. Sabharwal. Towards understanding and harnessing the potential of clause learning. *Journal of Artificial Intelligene Research*, 22(1):319–351, 2004.

[5] A. Biere. BooleForce and TraceCheck, 2014.

[6] A. Biere, D. L. Berre, E. Lonca, and N. Manthey. Detecting cardinality constraints in CNF. In C. Sinz and U. Egly, editors, *SAT 2014*, volume 8561 of *LNCS*, pages 285–301. Springer, 2014.

[7] R. Brummayer and A. Biere. Fuzzing and delta-debugging SMT solvers. pages 1–5, 2009.

[8] J. M. Crawford, M. L. Ginsberg, E. M. Luks, and A. Roy. Symmetry-breaking predicates for search problems. In L. C. Aiello, J. Doyle, and S. C. Shapiro, editors, *KR 1996*, pages 148–159. Morgan Kaufmann, 1996.

[9] L. Cruz-Filipe, M. J. H. Heule, W. A. H. Jr., M. Kaufmann, and P. Schneider-Kamp. Efficient certified RAT verification. *CoRR*, abs/1612.02353, 2016.

[10] L. Cruz-Filipe, J. Marques-Silva, and P. Schneider-Kamp. Efficient certified resolution proof checking. *CoRR*, abs/1610.06984, 2016.

[11] N. Eén and A. Biere. Effective preprocessing in SAT through variable and clause elimination. In F. Bacchus and T. Walsh, editors, *SAT 2005*, volume 3569 of *LNCS*, pages 61–75, Heidelberg, 2005. Springer.

[12] N. Eén and N. Sörensson. An extensible SAT-solver. In E. Giunchiglia and A. Tacchella, editors, *SAT 2003*, volume 2919 of *LNCS*, pages 502–518, Heidelberg, 2004. Springer.

[13] A. V. Gelder. Extracting (easily) checkable proofs from a satisfiability solver that employs both preorder and postorder resolution. In *ISAIM 2002*, 2002.

[14] A. V. Gelder. Producing and verifying extremely large propositional refutations - have your cake and eat it too. *Annals of Mathematics and Artificial Intelligence*, 65(4):329–372, 2012.

[15] E. Goldberg and Y. Novikov. Verification of proofs of unsatisfiability for CNF formulas. In *DATE 2003*, pages 10886–10891, Washington, DC, USA, 2003. IEEE Computer Society.

[16] A. Gurfinkel and Y. Vizel. Druping for interpolates. In *FMCAD 2014*, pages 99–106. IEEE, 2014.

[17] A. Haken. The intractability of resolution. *Theor. Comput. Sci.*, 39:297–308, 1985.

[18] M. Heule, M. Järvisalo, and A. Biere. Clause elimination procedures for cnf formulas. In C. Fermüller and A. Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning*, volume 6397 of *Lecture Notes in Computer Science*, pages 357–371. Springer Berlin Heidelberg, 2010.

[19] M. J. H. Heule. The DRAT format and drat-trim checker. *CoRR*, abs/1610.06229, 2016.

[20] M. J. H. Heule, W. A. Hunt, Jr, and N. Wetzler. Expressing symmetry breaking in DRAT proofs. In A. P. Felty and A. Middeldorp, editors, *CADE 25*, volume 9195 of *LNCS*, pages 591–606. Springer, 2015.

[21] M. J. H. Heule, W. A. Hunt Jr., and N. Wetzler. Trimming while checking clausal proofs. In *FMCAD 2013*, pages 181–188. IEEE, 2013.

[22] M. J. H. Heule, M. Järvisalo, and A. Biere. Clause elimination procedures for CNF formulas. In C. G. Fermüller and A. Voronkov, editors, *LPAR 2010*, volume 6397 of *LNCS*, pages 357–371, Heidelberg, 2010. Springer.

[23] M. J. H. Heule, M. Järvisalo, F. Lonsing, M. Seidl, and A. Biere. Clause elimination for SAT and QSAT. *Journal of Artificial Intelligene Research*, 53:127–168, 2015.

[24] M. J. H. Heule, O. Kullmann, and V. W. Marek. Solving and verifying the Boolean pythagorean triples problem via cube-and-conquer. In N. Creignou and D. L. Berre, editors, *SAT 2016*, volume 9710 of *LNCS*, pages 228–245. Springer, 2016.

[25] M. J. H. Heule, N. Manthey, and T. Philipp. Validating unsatisfiability results of clause sharing parallel SAT solvers. In *POS 2014*, 2014.

[26] M. Järvisalo, A. Biere, and M. J. H. Heule. Blocked clause elimination. In J. Esparza and R. Majumdar, editors, *TACAS 2010*, volume 6015 of *LNCS*, pages 129–144, Heidelberg, 2010. Springer.

[27] M. Järvisalo, M. J. H. Heule, and A. Biere. Inprocessing rules. In B. Gramlich, D. Miller, and U. Sattler, editors, *IJCAR 2012*, volume 7364 of *LNCS*, pages 355–370, Heidelberg, 2012. Springer.

[28] H. Kleine Büning and T. Lettmann. *Aussagenlogik - Deduktion und Algorithmen*. Leitfäden und Monographien der Informatik. Teubner, 1994.

[29] J. Krajicek. *Bounded Arithmetic, Propositional Logic and Complexity Theory*. Encyclopedia of Mathematics an. Cambridge University Press, 1995.

[30] O. Kullmann. On a generalization of extended resolution. *Discrete Applied Mathematics*, 96-97:149–176, 1999.

[31] T. Laitinen. *Extending SAT Solver with Parity Reasoning*. PhD thesis.

[32] T. Laitinen, T. A. Junttila, and I. Niemelä. Classifying and propagating parity constraints. In M. Milano, editor, *CP 2012*, volume 7514 of *LNCS*, pages 357–372. Springer, 2012.

[33] N. Manthey, M. J. H. Heule, and A. Biere. Automated reencoding of Boolean formulas. In A. Biere, A. Nahir, and T. Vos, editors, *HVC 2012*, volume 7857 of *LNCS*, pages 102–117, Heidelberg, 2013. Springer.

[34] N. Manthey and M. Lindauer. Spybug: Automated bug detection in the configuration space of SAT solvers. In N. Creignou and D. L. Berre, editors, *Theory and Applications of Satisfiability Testing - SAT 2016 - 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings*, volume 9710 of *Lecture Notes in Computer Science*, pages 554–561. Springer, 2016.

[35] N. Manthey and T. Philipp. Checking unsatisfiability proofs in parallel. In D. L. Berre, O. Roussel, and A. V. Gelder, editors, *POS 2015*. EasyChair.

[36] N. Manthey and T. Philipp. Formula simplifications as DRAT derivations. In C. Lutz and M. Thielscher, editors, *Proceedings of the 17th German German Conference on Artificial Intelligence*, volume 8736 of *Lecture Notes in Computer Science*, pages 111–122. Springer, 2014.

[37] N. Manthey, T. Philipp, and C. Wernhard. Soundness of inprocessing in clause sharing SAT solvers. volume 7962, pages 22–39, Heidelberg, 2013. Springer.

[38] J. P. Marques Silva and K. A. Sakallah. GRASP: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48(5):506–521, 1999.

[39] T. Philipp. An expressive model for instance decomposition based parallel SAT solvers. In C. Lutz and S. Ranise, editors, *Proceedings of the 10th International Symposium on Frontiers of Combining Systems (FroCos 2015)*, volume 9322 of *LNCS*, pages 101–116. Springer, 2015.

[40] T. Philipp and A. Rebola-Pardo. DRAT proofs for XOR reasoning. In L. Michael and A. C. Kakas, editors, *JELIA 2016*, volume 10021 of *LNCS*, pages 415–429, 2016.

[41] T. Philipp and A. Rebola-Pardo. Fuzzing and verifying RAT refutations with deletion information. In *30th International FLAIRS Conference, Marco Island, USA, May 22-24, 2017, Proceedings*, 2017. Forthcoming.

[42] T. Philipp and A. Tigunova. A verified decision procedure for pseudo-Boolean formulas. In *PlanSIG 2016*, 2016.

[43] D. A. Plaisted and S. Greenbaum. A structure-preserving clause form translation. *Journal of Symbolic Computation*, 2(3):293–304, 1986.

[44] M. Soos. Enhanced gaussian elimination in DPLL-based SAT solvers. In *POS 2010*, 2010.

[45] M. Soos, K. Nohl, and C. Castelluccia. Extending SAT solvers to cryptographic problems. In O. Kullmann, editor, *SAT 2009*, pages 244–257, Heidelberg, 2009. Springer.

[46] S. Subbarayan and D. K. Pradhan. NiVER: Non-increasing variable elimination resolution for preprocessing SAT instances. In H. H. Hoos and D. G. Mitchell, editors, *SAT 2004*, volume 3542 of *LNCS*, pages 276–291, Heidelberg, 2005. Springer.

[47] G. S. Tseitin. On the complexity of derivations in propositional calculus. In A. O. Slisenko, editor, *Studies in Constructive Mathematics and Mathematical Logic*, pages 115–125. Consultants Bureau, New York, 1970.

[48] A. Urquhart. Hard examples for resolution. *J. ACM*, 34(1):209–219, 1987.

[49] A. Urquhart. The symmetry rule in propositional logic. *Discrete Applied Mathematics*, 96-97:177–193, 1999.

[50] N. Wetzler, M. J. H. Heule, and W. A. Hunt Jr. Mechanical verification of SAT refutations with extended resolution. In S. Blazy, C. Paulin-Mohring, and D. Pichardie, editors, *ITP 2013*, volume 7998 of *LNCS*, pages 229–244, Heidelberg, 2013. Springer.

[51] N. Wetzler, M. J. H. Heule, and W. A. Hunt Jr. DRAT-trim: Efficient checking and trimming using expressive clausal proofs. In C. Sinz and U. Egly, editors, *SAT 2014*, volume 8561 of *LNCS*, pages 422–429. Springer, 2014.