DRAT Proofs of Unsatisfiability for SAT Modulo Monotonic Theories

Nick Feng[1(✉)], Alan J. Hu[2], Sam Bayless[3], Syed M. Iqbal[3], Patrick Trentin[3], Mike Whalen[3], Lee Pike[3], and John Backes[3]

[1] Dept. of Computer Science, University of Toronto, Toronto, Canada
fengnick@cs.toronto.edu
[2] Dept. of Computer Science, University of British Columbia, Vancouver, Canada
ajh@cs.ubc.ca
[3] Amazon Web Services, Seattle, Minneapolis, Portland, USA
{sabayles,iqsye,trentinp,mww,leepike,jbackes}@amazon.com

Abstract. Generating proofs of unsatisfiability is a valuable capability of most SAT solvers, and is an active area of research for SMT solvers. This paper introduces the first method to efficiently generate proofs of unsatisfiability specifically for an important subset of SMT: SAT Modulo *Monotonic* Theories (SMMT), which includes many useful finite-domain theories (e.g., bit vectors and many graph-theoretic properties) and is used in production at Amazon Web Services. Our method uses propositional definitions of the theory predicates, from which it generates compact Horn approximations of the definitions, which lead to efficient DRAT proofs, leveraging the large investment the SAT community has made in DRAT. In experiments on practical SMMT problems, our proof generation overhead is minimal (7.41% geometric mean slowdown, 28.8% worst-case), and we can generate and check proofs for many problems that were previously intractable.

An extended version of this paper, which includes appendices with proofs and additional results, is available at ***https://doi.org/10.48550/arXiv.2401.10703***

1 Introduction

This paper introduces the first method to efficiently generate and check proofs of unsatisfiability for SAT Modulo Monotonic Theories (SMMT), an important fragment of general SMT. The motivation for this work rests on these premises:

– *Proofs of UNSAT are valuable, for propositional SAT as well as SMT.* Obviously, an independently checkable proof increases trust, which is important because an incorrect UNSAT result can result in certifying correctness of an incorrect system. Additionally, proofs are useful for computing abstractions [30,17,25] via interpolation in many application domains including model checking [30] and software analysis [29,23].

© The Author(s) 2024

- *SMMT is a worthy fragment of SMT as a research target.* SMMT [9] is a technique for efficiently supporting finite, monotonic theories in SMT solvers. E.g., reachability in a graph is monotonic in the sense that adding edges to the graph only increases reachability, and an example SMMT query would be whether there exists a configuration of edges such that node a can reach node b, but node c can't reach node d. (More formal background on SMMT is in Sec. 2.2.) The most used SMMT theories are graph reachability and max-flow, along with bit-vector addition and comparison. Applications include circuit escape routing [11], CTL synthesis [28], virtual data center allocation [12], and cloud network security and debugging [2,8], with the last two applications being deployed in production by Amazon Web Services (AWS). Indeed, our research was specifically driven by industrial demand.
- *DRAT is a desirable proof format.* (Here, we include related formats like DRUP [27], GRIT [19], and LRAT [18]. DRAT is explained in Sec. 2.1.) For an independent assurance of correctness, the proof *checker* is the critical, trusted component, and hence must be as trustworthy as possible. For (propositional) SAT, the community has coalesced around the DRAT proof format [37], for which there exist independent, efficient proof checkers [37], mechanically verified proof checkers [38], and even combinations that are fast as well as mechanically proven [18]. The ability to emit DRAT proof certificates has been required for solvers in the annual SAT Competition since 2014.
  Unfortunately, DRAT is propositional, so general SMT solvers need additional mechanisms to handle theory reasoning [6]. For example, Z3 [32] outputs natural-deduction-style proofs [31], which can be reconstructed inside the interactive theorem prover Isabelle/HOL [14,15]. Similarly, veriT [16] produces resolution proof traces with theory lemmas, and supports proof reconstruction in both Coq [1] and Isabelle [21,5,4]. As a more general approach, CVC4 [7] produces proofs in the LFSC format [36], which is a meta-logic that allows describing theory-specific proof rules for different SMT theories. Nevertheless, given the virtues of DRAT, SMT solvers have started to harness it for the propositional reasoning, e.g., CVC4 supports DRAT proofs for bit-blasting of the bit-vector theory, which are then translated into LFSC [34], and Otoni et al. [33] propose a DRAT-based proof certificate format for propositional reasoning that they extend with theory-specific certificates. However, in both cases, the final proof certificate is not purely DRAT, and any theory lemmas must be checked by theory-specific certificate checkers.
- *For typical finite-domain theories, defining theory predicates propositionally is relatively straightforward.* The skills to design and implement theory-specific proof systems are specialized and not widely taught. In contrast, if we treat a theory predicate as simply a Boolean function, then anyone with basic digital design skills can build a circuit to compute the predicate (possibly using readily available commercial tools) and then apply the Tseitin transform to convert the circuit to CNF. (This is known as "bit-blasting", but we will see later that conventional bit-blasting is too inefficient for SMMT.)

From a practical, user-level perspective, the contribution of this paper is the first efficient proof-generating method for SMMT. Our method scales to industrial-size instances and generates pure DRAT proofs.

From a theoretical perspective, the following contributions underlie our method:

– We introduce the notion of one-sided propositional definitions for refutation proof. Having different definitions for a predicate vs. its complement allows for more compact and efficient constructions.
– We show that SMMT theories expressed in Horn theory enable linear-time (in the size of the Horn definition) theory lemma checking via reverse unit propagation (RUP), and hence DRAT.
– We propose an on-the-fly transformation that uses hints from the SMMT solver to over-approximate any CNF encoding of a monotonic theory predicate into a linear-size Horn upper-bound, and prove that the Horn upper-bound is sufficient for checking theory lemmas in any given proof via RUP.
– We present efficient, practical propositional definitions for the main monotonic theories used in practice: bit-vector summation and comparison, and reachability and max-flow on symbolic graphs.

(As an additional minor contribution, we adapt the BackwardCheck procedure from DRAT-Trim [27] for use with SMT, and evaluate its effectiveness in our proof checker.)

We implemented our method in the MonoSAT SMMT solver [10]. For evaluation, we use two sets of benchmarks derived from practical, industrial problems: multilayer escape routing [11], and cloud network reachability [2].[4] Our results show minimal runtime overhead on the solver (geometric mean slowdown 7.4%, worst-case 28.8% in our experiments), and we generate and check proofs for many problem instances that are otherwise intractable.

2 Background

2.1 Propositional SAT and DRAT

We assume the reader is familiar with standard propositional satisfiability on CNF. Some notational conventions in our paper are: we use lowercase letters for literals and uppercase letters for clauses (or other sets of literals); for a literal x, we denote the variable of x by $var(x)$; we will interchangeably treat an *assignment* either as a mapping of variables to truth values $\top$ (true) or $\bot$ (false), or as a set of non-conflicting (i.e., does not contain both x and its complement $\bar{x}$) literals, with positive (negative) literals for variables assigned $\top$ ($\bot$); assignments can be *total* (assigns truth values to every variable) or *partial* (some variables unassigned); and given a formula F and assignment M, we use the vertical bar $F|_M$ to denote reducing the formula by the assignment, i.e., discarding falsified

[4] Available at `https://github.com/NickF0211/MonoProof`.

literals from clauses and satisfied clauses from the formula. (An empty clause denotes $\bot$; an empty formula, $\top$.)

This paper focuses on proofs of unsatisfiability. In proving a formula F UNSAT, a clause C is *redundant* if F and $F \wedge C$ are equisatisfiable [26]. A proof of unsatisfiability is simply a sequence of redundant clauses culminating in $\bot$, but where the redundancy of each clause can be easily checked. However, checking redundancy is coNP-hard. A clause that is *implied* by F, which we denote by $F \models C$, is guaranteed redundant, and we can check implication by checking the unsatisfiability of $F \wedge \overline{C}$, but this is still coNP-complete. Hence, proofs use restricted proof rules that guarantee redundancy. For example, the first automated proofs of UNSAT used resolution to generate implied clauses, until implying $\bot$ by resolving a literal l with its complement $\overline{l}$ [20,39]. In practice, however, resolution proofs grow too large on industrial-scale problems.

DRAT [37] is a much more compact and efficient system for proving unsatisfiability. It is based on *reverse unit propagation* (RUP), which we explain here.[5] A *unit clause* is a clause containing one literal. If L is the set of literals appearing in the unit clauses of a formula F, the *unit clause rule* computes $F|_L$, and the repeated application of the unit clause rule until a fixpoint is called *unit propagation* (aka *Boolean constraint propagation*). Given a clause C, its negation $\overline{C}$ is a set of unit clauses, and we denote by $F \vdash_1 C$ if $F \wedge \overline{C}$ derives a conflict through unit propagation. Notice that $F \vdash_1 C$ implies $F \models C$, but is computationally easy to check. The key insight [24] behind RUP is that modern CDCL SAT solvers make progress by deriving learned clauses, whose redundancy is, by construction, checkable via unit propagation. Proof generation, therefore, is essentially just logging the sequence of learned clauses leading to $\bot$, and proof checking is efficiently checking $\vdash_1$ of the relevant learned clauses.

2.2 SAT Modular Monotonic Theories (SMMT)

We define a Boolean *positive monotonic predicate* as follows:

Definition 1 (Positive Monotonic Predicate). *A predicate* $p : \{0,1\}^n \rightarrow \{0,1\}$ *is positively monotonic with respect to the input* a_i *iff*

$$p(a_1, \ldots, a_{i-1}, 0, a_{i+1}, \ldots) \implies p(a_1, \ldots, a_{i-1}, 1, a_{i+1}, \ldots)$$

The predicate p *is a positive monotonic predicate iff* p *is positively monotonic with respect to every input.*

Negative monotonic predicates are defined analogously. If a predicate p is positively monotonic w.r.t. some inputs A^+ and negatively monotonic w.r.t. the rest of inputs A^-, it is always possible to rewrite the predicate as a positive monotonic predicate p' over input A^+ and $\{\overline{a} \mid a \in A^-\}$. For ease of exposition,

[5] RUP is all we use in this paper. RAT is a superset of RUP, by essentially doing one step of resolution as a "lookahead" before checking RUP of the resolvents. The "D" in DRAT stands for "deletion", meaning the proof format also records clause deletions.

and without loss of generality, we will describe our theoretical results assuming positive monotonic predicates only (except where noted otherwise).

Given a monotonic predicate p over input A, we will use boldface $\mathbf{p}$ as the *predicate atom* for p, i.e., the predicate atom is a Boolean variable in the CNF encoding of the theory, indicating whether $p(A)$ is true or not. The *theory of* $\mathbf{p}$ is the set of valid implications in the form of $\mathcal{M}_A \Rightarrow \mathbf{p}$ where $\mathcal{M}_A$ is a partial assignment over A.

The following are the most used monotonic theories:

Graph Reachability: Given a graph $G = (V, E)$, where V and E are sets of vertices and edges, the graph reachability theory contains the reachability predicates $reach_u^v$ on the input variables $e_1, e_2 \ldots e_m \in E$, where $u, v \in V$. The predicate holds iff node u can reach v in the graph G by using only the subset of edges whose corresponding variable e_i is true. The predicate is positively monotonic because enabling more edges will not make reachable nodes unreachable, and disabling edge will not make unreachable nodes reachable.

Bit-Vector Summation and Comparison: Given two bit-vectors (BV) $\vec{a}$ and $\vec{b}$, the theory of BV comparison contains the predicate $\vec{a} \geq \vec{b}$, whose inputs are the bits of $\vec{a}$ and $\vec{b}$. The predicate holds iff the value (interpreted as an integer) of $\vec{a}$ is greater or equal to the value of $\vec{b}$. The predicate is positively monotonic for the variables of $\vec{a}$ and negatively monotonic for the variables of $\vec{b}$, because changing any 0 to a 1 in $\vec{a}$ makes it bigger, and changing any 1 to 0 in $\vec{b}$ makes it smaller. Similarly, given two sets of BVs $\vec{A}$ and $\vec{B}$, the theory of comparison between sums contains the predicate $\sum \vec{A} \geq \sum \vec{B}$ whose inputs are the boolean variables from all BVs in $\vec{A}$ and $\vec{B}$. The predicate holds iff the sum of the BVs in $\vec{A}$ is greater or equal to the sum of the BVs in $\vec{B}$, and is positively monotonic in $\vec{A}$ and negatively monotonic in $\vec{B}$.

S-T Max Flow Given a graph $G = (V, E)$, for every edge $e \in E$, let its *capacity* be represented by the BV $\vec{cap}_e$. For two vertices $s, t \in V$, and a BV $\vec{z}$, the max-flow theory contains the predicates $MF_s^t \geq \vec{z}$ over the input variables $e_1, e_2 \ldots e_n \in E$ and $\vec{cap}_{e_1}, \vec{cap}_{e_2} \ldots \vec{cap}_{e_n}$. The predicate holds iff the maximum flow from the source s to the target t is greater or equal to $\vec{z}$, using only the enabled edges (as in the reachability theory) with their specified capacities.

The SMMT Framework [10] describes how to extend a SAT or SMT solver with Boolean monotonic theories. The framework has been implemented in the SMT solver MonoSAT, which has been deployed in production by Amazon Web Services to reason about a wide range of network properties [2,8]. The framework performs theory propagation and clause learning for SMMT theories as follows: (In this description, we use P for the set of positive monotonic predicates, and S for the set of Boolean variables that are arguments to the predicates.)

Theory Propagation: Given a partial assignment M, let M_s be the partial assignment over S. The SMMT framework forms two complete assignments of M_s: one with all unassigned s atoms assigned to false (M_s^-), one with all unassigned s atoms assigned to true (M_s^+). Since M_s^- and M_s^+ are each complete assignments of S, they can be used to determine the value of P atoms. Since every $\mathbf{p} \in P$ is positively monotonic, (1) if $M_s^- \Rightarrow \mathbf{p}$, then $M_s \Rightarrow \mathbf{p}$, and (2) if $M_s^+ \Rightarrow \neg\mathbf{p}$, then $M_s \Rightarrow \neg\mathbf{p}$. The framework uses M_s^- and M_s^+ as the under- and over-approximation for theory propagation over P atoms. Moreover, the framework attaches $M_s \Rightarrow \mathbf{p}$ or $M_s \Rightarrow \neg\mathbf{p}$ as the reason clause for the theory propagation.

Clause Learning: For some predicates, a witness can be efficiently generated during theory propagation, as a sufficient condition to imply the predicate p. For example, in graph reachability, suppose $M_s^- \Rightarrow \mathbf{reach_{u,v,G}}$ for a given under-approximation M_s^-. Standard reachability algorithms can efficiently find a set of edges $M_s' \subseteq M_s$ that forms a path from u to v. When such a witness is available, instead of learning $M_s \Rightarrow \mathbf{p}$, the framework would use the path witness to learn the stronger clause $M_s' \Rightarrow \mathbf{p}$. Witness-based clause learning is theory specific (and implementation specific); if a witness is not available or cannot be efficiently generated in practice for a particular predicate, the framework will learn the weaker clause $M_s \Rightarrow \mathbf{p}$.

3 Overview of Our Method

Most leading SMT solvers, including MonoSAT, use the DPLL(T) framework [22], in which a CDCL propositional SAT solver coordinates one or more theory-specific solvers. A DPLL(T) solver behaves similarly to a CDCL propositional SAT solver — making decisions, performing unit propagation, analyzing conflicts, learning conflict clauses — except that the theory solvers will also introduce new clauses (i.e., theory lemmas) into the clause database, which were derived via theory reasoning, and whose correctness relies on the semantics of the underlying SMT theory. These theory lemmas cannot (in general) be derived from the initial clause database, and so cannot be verified using DRAT. Therefore, the problem of producing a proof of UNSAT in SMT reduces to the problem of proving the theory lemmas.

A direct approach would be to have the SMT solver emit a partial DRAT proof certificate, in which each theory lemma is treated as an axiom. This partial proof is DRAT-checkable, but each theory lemma becomes a new proof obligation. The theory lemmas could subsequently be verified using external (non-DRAT), trusted, theory-specific proof-checking procedures. This is the approach recently proposed by Otoni et al. [33].

We take such an approach as a starting point, but instead of theory-specific proof procedures, we use propositional definitions of the theory semantics to add clauses sufficient to prove (by RUP) the theory lemmas. The resulting proof is purely DRAT, checkable via standard DRAT checkers, with no theory-specific proof rules. Fig. 1 explains our approach in more detail; Sec. 4 dives into how we derive the added clauses; and Sec. 5 gives sample propositional theory definitions.

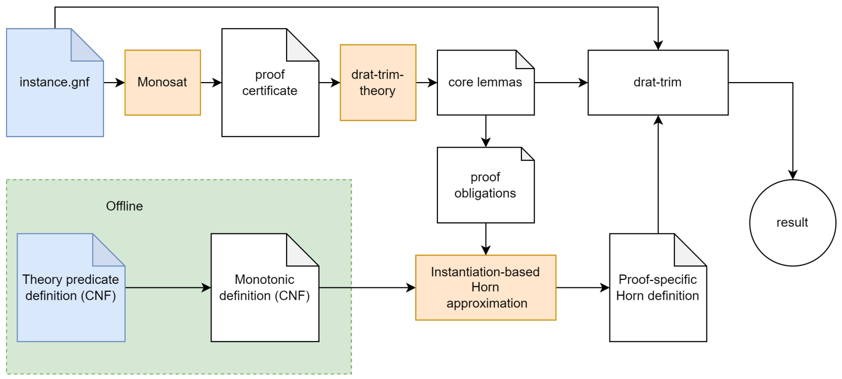
instance.gnf
Monosat
proof certificate
drat-trim-theory
core lemmas
drat-trim
proof obligations
result
Offline
Theory predicate definition (CNF)
Monotonic definition (CNF)
Instantiation-based Horn approximation
Proof-specific Horn definition

Fig. 1. Overview of Our Proof Generation and Checking Method. Inputs (the problem instance file and the propositional definitions of theory predicates) are colored blue; new and modified components are colored orange. Starting from the top-left is the SMMT problem instance, which is solved by MonoSAT. We extended MonoSAT to emit a DRAT-style proof certificate, consisting of learned (via propositional or theory reasoning) clauses, similar to what is proposed in [33]. The proof certificate is optionally pre-processed by *drat-trim-theory*, in which we modified the BackwardCheck procedure [27] to perform a backward traversal from the final ⊥, outputting a subset of lemmas sufficient (combined with the original clause database) to derive ⊥. This is extra work (since a full BackwardCheck is later performed by unmodified *drat-trim* for the final proof verification at the top-right of the figure), but allows us to avoid verifying some theory lemmas that are not relevant to the final proof. The resulting core lemmas are split between the propositional learned clauses, which go straight (right) to *drat-trim*, and the theory learned clauses, which are our proof obligations. The heart of our method is the instantiation-based Horn approximation (bottom-center, described in Sec. 4). In this step, we use the proof obligations as hints to transform the pre-defined, propositional theory definitions (bottom-left, examples in Sec. 5) into proof-specific Horn definitions. The resulting proof-specific definitions together with the CNF from the input instance can efficiently verify UNSAT using unmodified drat-trim [37].

4 Instantiation-Based Horn Approximation

This section describes how we derive a set of clauses sufficient to make theory lemmas DRAT-checkable. Section 4.1 introduces one-sided propositional definitions and motivates the goal of a compact, Horn-clause-based definition. Section 4.2 gives a translation from an arbitrary propositional definition of a monotonic predicate to a *monotonic definition*, as an intermediate step toward constructing the final proof-specific, Horn definition in Section 4.3.

4.1 One-Sided Propositional Definitions and Horn Clauses

Definition 2 (Propositional Definition). *Let* $\mathbf{p}$ *be the positive predicate atom of predicate* p *over Boolean arguments* A*. A* propositional definition *of* $\mathbf{p}$*, denoted as* $\Sigma_{\mathbf{p}}$*, is a CNF formula over variables* $V \supseteq (var(\mathbf{p}) \cup A)$ *such that for every truth assignment* $\mathcal{M}$ *to the variables in* A*, (1)* $\Sigma_{\mathbf{p}}|_{\mathcal{M}}$ *is satisfiable and*

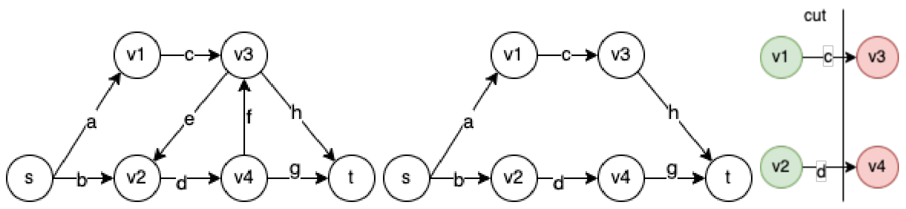

Fig. 2. Directed Graph for Running Example in Sec. 4. In the symbolic graph (left), the reachability predicate $reach_s^t$ is a function of the edge inputs $a, \ldots, h$.

(2) $\Sigma_{\mathbf{p}} \models (\mathcal{M} \Rightarrow \mathbf{p})$ *if and only if* $p(\mathcal{M})$ *is* $\top$. *The propositional definition of* $\bar{\mathbf{p}}$ *is defined analogously.*

For example, the Tseitin-encoding of a logic circuit that computes $p(\mathcal{M})$ satisfies this definition. However, note that a propositional definition for $\mathbf{p}$ can be one-sided: it is not required that $\Sigma_{\mathbf{p}} \models (\mathcal{M} \Rightarrow \bar{\mathbf{p}})$ when $p(\mathcal{M})$ is $\bot$. That case is handled by a separate propositional definition for $\bar{\mathbf{p}}$. We will see that this one-sidedness gives some freedom to admit more compact definitions.

Given a propositional definition $\Sigma_{\mathbf{p}}$, any theory lemma $\mathcal{M}_A \Rightarrow \mathbf{p}$ is a logical consequence of $\Sigma_{\mathbf{p}}$, but this might not be RUP checkable. One could prove $\Sigma_{\mathbf{p}} \models (\mathcal{M}_A \Rightarrow \mathbf{p})$ by calling a proof-generating SAT solver on $\Sigma_{\mathbf{p}} \wedge \overline{\mathcal{M}_A \Rightarrow \mathbf{p}}$, i.e., bit-blasting the specific lemma, but we will see experimentally (in Sec. 6) that this works poorly. However, if the propositional definition is limited to Horn theory (i.e., each clause has at most one positive literal), then every SMMT theory lemma *can* be proven by unit propagation:

Theorem 1. *Let* p *be a positive monotonic predicate over input* A*, and let* $\Sigma_{\mathbf{p}}^h$ *be a propositional definition for the positive atom* $\mathbf{p}$. *If* $\Sigma_{\mathbf{p}}^h$ *is set of Horn clauses, then for any theory lemma* $\mathcal{M}_A \Rightarrow \mathbf{p}$ *where* $\mathcal{M}_A$ *is a set of positive atoms from* A*,* $\Sigma_{\mathbf{p}}^h \models (\mathcal{M}_A \Rightarrow \mathbf{p})$ *if and only if* $\Sigma_{\mathbf{p}}^h \vdash_1 (\mathcal{M}_A \Rightarrow \mathbf{p})$.

Proof. Suppose $\Sigma_{\mathbf{p}}^h \models (\mathcal{M}_A \Rightarrow \mathbf{p})$, then $\Sigma_{\mathbf{p}}^h \wedge (\mathcal{M}_A \wedge \bar{\mathbf{p}})$ is unsatisfiable. Since $\mathcal{M}_A \wedge \bar{\mathbf{p}}$ is equivalent to a set of unit clauses, $\Sigma_{\mathbf{p}}^h \wedge (\mathcal{M}_A \wedge \bar{\mathbf{p}})$ still contains only Horn clauses, so satisfiability can be determined by unit propagation. □

Example 1. Let $reach_s^t$ be the reachability predicate for the directed graph shown in Fig. 2 (left). The definition schema for graph reachability in Sec. 5 yields the following set of Horn clauses: $\Sigma^h_{\mathbf{reach_s^t}} :=$ (1) $\bar{\mathbf{s}} \vee \bar{\mathbf{a}} \vee \mathbf{v1}$, (2) $\overline{\mathbf{v1}} \vee \bar{\mathbf{c}} \vee \mathbf{v3}$, (3) $\overline{\mathbf{v3}} \vee \bar{\mathbf{h}} \vee \mathbf{t}$, (4) $\bar{\mathbf{s}} \vee \bar{\mathbf{b}} \vee \mathbf{v2}$, (5) $\overline{\mathbf{v3}} \vee \bar{\mathbf{e}} \vee \mathbf{v2}$, (6) $\overline{\mathbf{v2}} \vee \bar{\mathbf{d}} \vee \mathbf{v4}$, (7) $\overline{\mathbf{v4}} \vee \bar{\mathbf{f}} \vee \mathbf{v3}$, (8) $\overline{\mathbf{v4}} \vee \bar{\mathbf{g}} \vee \mathbf{t}$, (9) $\bar{\mathbf{t}} \vee \mathbf{reach_s^t}$, (10) $\mathbf{s}$, where $v1, \ldots, v5$, s, and t are auxiliary variables. Any theory lemma of the form $\mathcal{M}_A \Rightarrow \mathbf{p}$, e.g., $\bar{\mathbf{a}} \vee \bar{\mathbf{c}} \vee \bar{\mathbf{h}} \vee \mathbf{reach_s^t}$, can be proven from $\Sigma^h_{\mathbf{reach_s^t}}$ via unit propagation. Also, note that one-sidedness allows a simpler definition, despite the cycle in the graph, e.g., consider assignment $\mathcal{M} = \{\bar{\mathbf{a}}, \bar{\mathbf{b}}, \bar{\mathbf{c}}, \mathbf{d}, \mathbf{e}, \mathbf{f}, \mathbf{g}, \mathbf{h}\}$. Then, $reach_s^t = \bot$, but $\Sigma^h_{\mathbf{reach_s^t}} \not\models (\mathcal{M} \Rightarrow \overline{\mathbf{reach_s^t}})$.

Horn theory has limited expressiveness, but it is always sufficient to encode a propositional definition for any SMMT theory: Given a monotonic predicate atom $\mathbf{p}$, we can always encode a *Horn propositional definition* $\Sigma^h_{\mathbf{p}}$ as the conjunction of all valid theory lemmas from the theory of $\mathbf{p}$. This is because every theory lemma is restricted to the form $(\mathcal{M}_A \Rightarrow \mathbf{p})$, where $\mathcal{M}_A$ is a set of positive atoms (due to monotonicity). Hence, $\Sigma^h_{\mathbf{p}}$ is a set of Horn clauses. However, such a naïve encoding blows up exponentially. Instead, we will seek a compact Horn definition $\Sigma^h_{\mathbf{p}}$ that *approximates* a non-Horn propositional definition $\Sigma_{\mathbf{p}}$:

Definition 3 (Horn Upper-Bound). *Let $\Sigma_{\mathbf{p}}$ be a propositional definition of $\mathbf{p}$. A set of Horn clauses $\Sigma^{h\uparrow}_{\mathbf{p}}$ is a Horn upper-bound if $\Sigma_{\mathbf{p}} \models \Sigma^{h\uparrow}_{\mathbf{p}}$.*

For the strongest proving power, we want the tightest Horn upper-bound possible. Unfortunately, the least Horn upper-bound of a non-Horn theory can still contain exponentially many Horn clauses [35]. Fortunately, we don't actually need a Horn upper-bound on the *exact* theory definition, but only of enough of the definition to prove the fixed set of theory lemmas that constitute the proof obligations. This motivates the next definition.

Definition 4 (Proof-Specific Horn Definition). *Given an exact definition $\Sigma_{\mathbf{p}}$ and a set of theory lemmas $\mathbb{O} := \{C_1, \dots C_n\}$ from the theory of $\mathbf{p}$, a* proof-specific Horn definition *of $\mathbf{p}$ is a Horn upper-bound $\Sigma^{h\uparrow}_{\mathbf{p}}$ of $\Sigma_{\mathbf{p}}$ such that $\Sigma^{h\uparrow}_{\mathbf{p}} \vdash_1 C$ for every $C \in \mathbb{O}$.*

Our goal in the next two subsections is how to derive such compact, proof-specific Horn definitions.

Example 2. Continuing Ex. 1, given a proof obligation $\mathbb{O}$ with two theory lemmas: $\{\overline{\mathbf{a}} \vee \overline{\mathbf{c}} \vee \overline{\mathbf{h}} \vee \mathbf{reach^t_s},\ \overline{\mathbf{b}} \vee \overline{\mathbf{d}} \vee \overline{\mathbf{g}} \vee \mathbf{reach^t_s}\}$, the subset of Horn clauses with IDs (1), (2), (3), (4), (6), (8), (9) and (10) is a proof-specific Horn definition for $\mathbf{reach^t_s}$, which can be visualized in Fig. 2 (middle).

Given a proof obligation $\mathbb{O}$, we can make all theory lemmas in $\mathbb{O}$ DRAT checkable if we have exact propositional definitions for the theories and if we can dynamically transform them into compact, proof-specific Horn definitions at the time of proof checking. We simply add these additional clauses to the input of the DRAT-proof-checker.

4.2 Monotonic Definitions

The derivation of compact, proof-specific Horn definitions from arbitrary propositional definitions is a two-step process: we first show that every propositional definition for a monotonic predicate atom can be converted into a monotonic definition of linear size (this section), and then use theory lemmas in the proof obligations to create the Horn approximation of the definition (Sec. 4.3).

Definition 5 (Monotonic Definition). *Let a monotonic predicate p over input A be given. A CNF formula $\Sigma^+_{\mathbf{p}}$ is a monotonic definition of the positive predicate atom $\mathbf{p}$ if $\Sigma^+_{\mathbf{p}}$ is a propositional definition of $\mathbf{p}$, and it satisfies the following syntax restrictions: (1) $\Sigma^+_{\mathbf{p}}$ does not contain positive atoms from A, (2) $\Sigma^+_{\mathbf{p}}$ does not contain $\bar{\mathbf{p}}$, and (3) $\mathbf{p}$ appears only in Horn clauses. The monotonic definition for $\bar{\mathbf{p}}$ is defined analogously.*

We now define the procedure, MonoT, for transforming a propositional definition into a linear-size monotonic definition:

Definition 6 (Monotonic Transformation). *Let a monotonic predicate p over input A and a propositional definition $\Sigma_{\mathbf{p}}$ for the positive predicate atom $\mathbf{p}$ be given.* MonoT$(\mathbf{p}, \Sigma_{\mathbf{p}})$ *is the result of the following transformations on $\Sigma_{\mathbf{p}}$: (1) replace every occurrence of an input atom ($\mathbf{a}$ for $a \in A$) in $\Sigma_{\mathbf{p}}$ with a new atom $\mathbf{a}'$ ($\overline{\mathbf{a}}$ is replaced with $\overline{\mathbf{a}'}$), (2) replace every occurrence of $\mathbf{p}$ and $\overline{\mathbf{p}}$ with $\mathbf{p}'$ and $\overline{\mathbf{p}'}$ respectively, and (3) add the following Horn clauses: $\mathbf{a} \Rightarrow \mathbf{a}'$ for every $a \in A$, and $\mathbf{p}' \Rightarrow \mathbf{p}$.*

Theorem 2 (Correctness of Monotonic Transformation). *Given a monotonic predicate p over input A and the monotonic predicate atom $\mathbf{p}$, if we have any propositional definition $\Sigma_{\mathbf{p}}$ with n clauses, then* MonoT$(\mathbf{p}, \Sigma_{\mathbf{p}})$ *results in a monotonic definition $\Sigma^{+}_{\mathbf{p}}$ with at most $n + |A| + 1$ clauses.*

The proof of Theorem 2 is in the extended version of this paper. The correctness relies on the fact that the predicate p is indeed monotonic, and that our propositional definitions need only be one-sided. If the monotonic definition is already in Horn theory, it can be used directly verify theory lemmas via RUP; otherwise, we proceed to Horn approximation, described next.

4.3 Instantiation-Based, Proof-Specific Horn Definition

We present the transformation from monotonic definitions into proof-specific Horn definitions. The transformation exploits the duality between predicates' positive and negative definitions.

Lemma 1 (Duality). *Let p be a monotonic predicate over Boolean arguments A. Suppose $\Sigma_{\mathbf{p}}$ and $\Sigma_{\bar{\mathbf{p}}}$ are positive and negative propositional definitions, respectively. For every assignment $\mathcal{M}$ to the variables in A:*

1. *$\Sigma_{\mathbf{p}} \models (\mathcal{M} \Rightarrow \mathbf{p})$ if and only if $\Sigma_{\bar{\mathbf{p}}} \wedge \mathcal{M} \wedge \mathbf{p}$ is satisfiable.*
2. *$\Sigma_{\bar{\mathbf{p}}} \models (\mathcal{M} \Rightarrow \bar{\mathbf{p}})$ if and only if $\Sigma_{\mathbf{p}} \wedge \mathcal{M} \wedge \bar{\mathbf{p}}$ is satisfiable.*

The proof of Lemma 1 is in the extended version of this paper. The duality of the positive ($\Sigma_{\mathbf{p}}$) and negative ($\Sigma_{\bar{\mathbf{p}}}$) definitions allows us to over-approximate positive (negative) definitions by instantiating the negative (positive) definitions.

Example 3. Returning to Ex. 1 and Fig. 2, consider the assignment $M = \{\mathbf{a}, \mathbf{b}, \overline{\mathbf{c}}, \overline{\mathbf{d}}, \mathbf{e}, \mathbf{f}, \mathbf{g}, \mathbf{h}\}$. Since s cannot reach t under this assignment, any propositional definition $\Sigma_{\overline{\mathbf{reach}^{\mathbf{t}}_{\mathbf{s}}}}$ must imply $M \Rightarrow \overline{\mathbf{reach}^{\mathbf{t}}_{\mathbf{s}}}$. Dually, $\Sigma^{h}_{\mathbf{reach}^{\mathbf{t}}_{\mathbf{s}}} \wedge M \wedge \overline{\mathbf{reach}^{\mathbf{t}}_{\mathbf{s}}}$ is satisfiable, e.g., $\{\mathbf{s}, \mathbf{v1}, \mathbf{v2}, \overline{\mathbf{v3}}, \overline{\mathbf{v4}}, \overline{\mathbf{t}}\}$.

Lemma 2 (Instantiation-Based Upper-Bound). *Let a predicate p over input A and a positive definition $\Sigma_{\mathbf{p}}$ be given. For any partial assignment M' over $var(\Sigma_{\mathbf{p}}) \setminus (var(p) \cup A)$, $\Sigma_{\mathbf{p}}|_{M' \cup \bar{\mathbf{p}}} \Rightarrow \bar{\mathbf{p}}$ is an over-approximation of $\Sigma_{\bar{\mathbf{p}}}$.* [6]

[6] Note that $\Sigma_{\mathbf{p}}|_{M'}$ is encoded in CNF, so to compactly (i.e., linear-size) encode $\Sigma_{\mathbf{p}}|_{M'} \Rightarrow \bar{\mathbf{p}}$ in CNF, we introduce a new literal l_i for each clause $C_i \in \Sigma_{\mathbf{p}}|_{M'}$, create clauses $\overline{c_{ij}} \vee l_i$ for each literal $c_{ij} \in C_i$, and add clause $\overline{l_1} \vee \overline{l_2} \vee \ldots \vee \overline{l_n} \vee \bar{\mathbf{p}}$.

The proof of Lemma 2 (in the extended paper) relies on the duality in Lemma 1. Lemma 2 enables upper-bound construction and paves the way for constructing an instantiation-based Horn upper-bound of a monotonic definition.

Lemma 3 (Instantiation-Based Horn Upper-Bound). *Given a monotonic predicate p over input A and a positive monotonic definition $\Sigma^+_{\mathbf{p}}$, let X represent the set of auxiliary variables: $var(\Sigma^+_{\mathbf{p}}) \setminus (A \cup var(p))$. For any complete satisfying assignment $M_{X \cup A}$ to $\Sigma^+_{\mathbf{p}}|_{\bar{\mathbf{p}}}$, the formula $(\Sigma^+_{\mathbf{p}}|_{\bar{\mathbf{p}} \cup \mathcal{M}_X}) \Rightarrow \bar{\mathbf{p}}$ serves as a Horn upper-bound for any propositional definition of $\bar{\mathbf{p}}$, where $\mathcal{M}_X$ is a partial assignment derived from $M_{X \cup A}$ for the auxiliary variables X.*

(Proof in the extended paper.) Note that the instantiation-based Horn upper-bound of a negative predicate atom $\bar{\mathbf{p}}$ is constructed from a monotonic definition of the positive predicate atom $\Sigma^+_{\mathbf{p}}$, and vice-versa.

For a given theory lemma, the instantiation-based Horn upper-bound construction (Lemma 3) enables the verification of the theory lemma if we can find a sufficient "witness" $\mathcal{M}_X$ for the instantiation. We now prove that a witness always exists for every valid theory lemma and does not exist otherwise.

Theorem 3 (Lemma-Specific Horn Upper-Bound). *Let a monotonic predicate p over input A, a monotonic definition $\Sigma^+_{\mathbf{p}}$ and a lemma in the form $\mathcal{M}_A \Rightarrow \overline{\mathbf{p}}$ be given. We denote X as the set of auxiliary variables: $var(\Sigma^+_{\mathbf{p}}) \setminus (A \cup var(p))$. The lemma $\mathcal{M}_A \Rightarrow \overline{\mathbf{p}}$ is in the theory of $\overline{\mathbf{p}}$ if and only if there exists an assignment $\mathcal{M}_X$ on X such that: (1) $\Sigma^+_{\mathbf{p}}|_{\bar{\mathbf{p}} \cup \mathcal{M}_X \cup \mathcal{M}_A}$ is satisfiable and (2) $(\Sigma^+_{\mathbf{p}}|_{\bar{\mathbf{p}} \cup \mathcal{M}_X} \Rightarrow \bar{\mathbf{p}}) \vdash_1 (\mathcal{M}_A \Rightarrow \bar{\mathbf{p}})$.*

(Proof in the extended paper.) Theorem 3 states that a lemma-specific Horn upper-bound for a theory lemma $\mathcal{M}_A \Rightarrow \overline{\mathbf{p}}$ can be constructed by instantiating the monotonic definition using a "witness" assignment $\mathcal{M}_X$. [7] The witness could be obtained by performing SAT solving on the formula $\Sigma^+_{\mathbf{p}}|_{\mathcal{M}^+_A \cup \overline{\mathbf{p}}}$, (where $\mathcal{M}^+_A$ is the extension of $\mathcal{M}_A$ by assigning unassigned input variables in A to $\top$ (Sec. 2.2)). However, in practice, a better approach is to modify the SMMT solver to produce the witness during the derivation of theory lemmas. In Section 5, we provide examples of witnesses for commonly used monotonic predicates.

Note that the witness is not part of the trusted foundation for the proof. An incorrect witness might not support verification of a theory lemma, but if a theory lemma is verified using a specific witness $\mathcal{M}_X$, Theorem 3 guarantees that the lemma is valid.

Example 4. Continuing the example, let a theory lemma $L := \mathbf{c} \vee \mathbf{d} \vee \overline{\mathbf{reach^t_s}}$ be given. To derive a lemma-specific Horn upper-bound for $\Sigma_{\overline{\mathbf{reach^t_s}}}$, we first obtain a witness $\mathcal{M}_X$ by finding a satisfying assignment to the formula $\Sigma^h_{\mathbf{reach^t_s}} \wedge M \wedge \overline{\mathbf{reach^t_s}}$, where $M := \{\mathbf{a}, \mathbf{b}, \overline{\mathbf{c}}, \overline{\mathbf{d}}, \mathbf{e}, \mathbf{f}, \mathbf{g}, \mathbf{h}\}$ (by assigning the unassigned

[7] Instead of instantiating a complete assignment on every auxiliary variable in X, a partial instantiation is sufficient so long as it determines the assignments on the other variables.

input variables in L to $\top$). Since M is a complete assignment to the edge variables, the graph is fully specified, and a suitable witness $\mathcal{M}_X$ can be efficiently computed using a standard graph-reachability algorithm, to compute the reachability status of each vertex. The witness $\mathcal{M}_X$ is $\{\mathbf{s}, \mathbf{v1}, \mathbf{v2}, \overline{\mathbf{v3}}, \overline{\mathbf{v4}}, \overline{\mathbf{t}}\}$. Following the construction in Theorem 3, the formula $\Sigma^h_{\mathbf{reach}^{\mathbf{t}}_{\mathbf{s}}}|_{\overline{\mathbf{reach}^{\mathbf{t}}_{\mathbf{s}}} \cup \mathcal{M}_X}$ simplifies to two (unit) clauses: $\overline{\mathbf{c}}$ and $\overline{\mathbf{d}}$ (from clauses (2) and (6) in Ex. 1), which can be visualized as the cut in Fig. 2 (right). The lemma-specific Horn upper bound $\Sigma^h_{\mathbf{reach}^{\mathbf{t}}_{\mathbf{s}}}|_{\overline{\mathbf{reach}^{\mathbf{t}}_{\mathbf{s}}} \cup \mathcal{M}_X} \Rightarrow \overline{\mathbf{reach}^{\mathbf{t}}_{\mathbf{s}}}$ is, therefore, $\overline{\mathbf{c}} \wedge \overline{\mathbf{d}} \Rightarrow \overline{\mathbf{reach}^{\mathbf{t}}_{\mathbf{s}}}$, which in this example is already CNF, but more generally, we would introduce two literals to encode the implication: $\{\mathbf{c} \vee \overline{\mathbf{l_1}},\ \mathbf{d} \vee \overline{\mathbf{l_2}},\ \mathbf{l_1} \vee \mathbf{l_2} \vee \overline{\mathbf{reach}^{\mathbf{t}}_{\mathbf{s}}}\}$. The lemma-specific Horn upper-bound is dual-Horn and implies the theory lemma L by unit propagation.

From the lemma-specific Horn upper-bounds, we construct the proof-specific Horn definition by combining the lemma-specific Horn upper-bounds for all lemmas in the proof obligations.

In summary, to efficiently verify SMMT theory lemmas, we propose the following approach: (1) define the propositional definitions (in CNF) for the atoms of theory predicates; (2) transform the definitions into monotonic definitions offline; (3) during proof checking, approximate a proof-specific Horn definition (if not already Horn) from the constructed monotonic definition using theory lemmas in the proof; (4) combine the proof-specific definition together and verify the proof via RUP. The only theory-specific, trusted foundation for the proof is the definition for the theory atoms. (The extended version of this paper contains a figure to help visualize this workflow.)

Example 5. Summarizing, the positive propositional definition $\Sigma_{\mathbf{reach}^{\mathbf{t}}_{\mathbf{s}}}$ in Ex. 1 is already Horn, so is sufficient for verifying via DRAT any SMMT lemmas that imply $\mathbf{reach}^{\mathbf{t}}_{\mathbf{s}}$. To verify lemmas that imply $\overline{\mathbf{reach}^{\mathbf{t}}_{\mathbf{s}}}$, we can compute a proof-specific definition of $\overline{\mathbf{reach}^{\mathbf{t}}_{\mathbf{s}}}$ from $\Sigma_{\mathbf{reach}^{\mathbf{t}}_{\mathbf{s}}}$ using Theorem 3.

Remark 1. The only trusted basis of our approach are the propositional definitions of theory atoms. For the monotonic theories in the section 5, we considered the definitions intuitively understandable, and therefore sufficiently trustworthy. But to further increase confidence, propositional definitions can be validated using techniques from hardware validation/verification, e.g., simulation to sanity-check general behavior, equivalence checking against known-good circuits, etc.

5 Example Propositional Definitions

In this section, we illustrate the monotonic definitions for the most commonly used monotonic predicates. Due to space constraints, we present only graph reachability here in detail, and only sketch bit-vector comparison and summation, and max-flow. Full definitions for those theories are in the extended version of this paper.

Graph Reachability: Given a graph $G = (V, E)$ where V and E are sets of vertices and edges, respectively, as discussed in Sect. 2, the graph reachability theory contains the reachability predicate $reach^v_u$ for $u, v \in V$ over input

$e_1, e_2 \dots e_n \in E$. For convenience, we refer to the positive edge atom for the edge from vertex i to vertex j as $\mathbf{e}_{i \to j}$. The predicate is positively monotonic for E, and the monotonic definition for the positive predicate atom $\mathbf{reach_u^v}$ contains the clauses:

1. $\overline{reach^i} \vee \overline{\mathbf{e}_{i\to j}} \vee reach^j$ for every edge $e_i^j \in E$ and the unit clause $reach^u$
2. $\overline{reach^v} \vee \mathbf{reach_u^v}$

The monotonic definition introduces a reachability atom $reach^i$ for every $i \in V$ and asserts the fact that u is reachable from itself. For every edge (i, j), if the edge (i, j) is enabled ($\mathbf{e}_{i\to j}$) and i is reachable ($reach^i$), then j must also be reachable ($reach^j$). The predicate atom $\mathbf{reach_u^v}$ is implied by the reachability of v ($reach^v$). The definition is monotonic since it only contains negative edge atoms. Moreover, the definition is already a Horn definition and can be used directly for proving theory lemmas in the theory of $\mathbf{reach_u^v}$ without the need for transformation into a proof-specific Horn definition. The size of the definition is $O(|E|)$.

Instead of defining the monotonic definition for the negative predicate atom $\overline{\mathbf{reach_u^v}}$, we construct its proof-specific definition from the monotonic definition of the positive predicate atom $\mathbf{reach_u^v}$. For each theory lemma in the proof, the witness for constructing the lemma-specific Horn upper-bound is the reachability status ($reach^i$) of every vertex $i \in V$, which is efficiently computed in the SMMT solver using standard graph-reachability algorithms.

Bit-Vector Comparison (sketch): The positive definition is just the Tseitin encoding of a typical bit-vector comparison circuit, with some simplification due to being one-sided: For each bit position i, we introduce auxiliary variables ge_i and gt_i, which indicate that the more-significant bits from this position have already determined vector $\vec{a}$ to be $\geq$ or $>$ $\vec{b}$, respectively. Simple clauses compute ge_{i-1} and gt_{i-1} from ge_i and gt_i and the bits at position $i-1$ of $\vec{a}$ and $\vec{b}$. The negative definition is similar. These are both Horn, so can be used without further transformation into proof-specific Horn definitions.

Bit-Vector Summation and Comparison (sketch): These are basically Tseitin encodings of ripple-carry adders, combined with the comparison theory above — using Def. 6 to handle the fact that the the Tseitin encodings of the XOR gates in the adders are non-monotonic with respect to the input bit-vectors. The resulting propositional definitions are not Horn, so we use witnesses to construct lemma-specific Horn definitions. The witnesses come from the SMMT solver maintaining lower and upper bounds on the possible values of the bit-vectors, e.g., a witness for $\sum \vec{A} \geq \sum \vec{B}$ are lower bounds for the vectors in $\vec{A}$ and upper bounds for the vectors in $\vec{B}$ such that their sums make the inequality true. (*Mutadis mutandis* for the negative witness.)

Max-Flow (sketch): For the positive definition (that the max-flow exceeds some value), we introduce auxiliary bit-vectors to capture the flow asisgned to each edge. We use the bit-vector theories to ensure that the flows do not exceed the edge capacities, that each node's (except the source) outgoing flows do not exceed the incoming flows (equality is unnecessary due to the one-sidedness), and that the flow to the sink exceeds the target value. For the negative definition, we exploit the famous max-flow/min-cut duality. We introduce an auxiliary variable $incut_e$ for each edge. We use the graph reachability theory to ensure that the edges in the cut separate the source from the sink, and the bit-vector summation theory to ensure that the capacity of the cut does not exceed the target max-flow value. Both the positive and negative definitions are not Horn, so require instantiation-based upper-bounds. The witnesses are the flow values or the cuts, and are easily computed by the SMMT solver.

6 Experimental Evaluation

To evaluate our proposed method, we implemented it as shown earlier in Fig. 1 (Sec. 3). We call our implementation *MonoProof* (available at `https://github.com/NickF0211/MonoProof`).

The two basic questions of any proof-generating SAT/SMT solver are: (1) how much overhead does the support for proofs add to the solving time, and (2) how efficiently can a proof be prepared from the proof log, and verified? For the first question, we compare the runtime of unmodified MonoSAT versus the MonoSAT that we have extended to produce proof certificates. For the second question, we need a baseline of comparison. MonoProof is the first proof-generating SMMT solver, so there is no obvious comparison. However, since SMMT theories are finite-domain, and bit-blasting (i.e., adding clauses that encode the theory predicates to the problem instance and solving via a propositional SAT solver) is a standard technique for finite-domain theories, we compare against bit-blasting. Arguably, this comparison is unfair, since MonoSAT outperforms bit-blasting when *solving* SMMT theories [9]. Thus, as an additional baseline, we propose an obvious hybrid of SMMT and bit-blasting, which we dub Lemma-Specific Bit-Blasting (LSBB): we run MonoProof until the core theory lemmas have been extracted, benefitting from MonoSAT's fast solving time, but then instead of using our techniques from Sec. 4, we bit-blast only the core theory lemmas.[8]

We ran experiments on 3GHZ AMD Epyc 7302 CPUs with 512GB of DDR4 RAM, with a timeout of 1 hour and memory limit of 64GB. For the bit-blasting SAT solver, we use the state-of-the-art SAT solver Kissat [13]. In all cases, the proof is verified with standard DRAT-trim [37].

6.1 Benchmarks

We wish to evaluate scalability on real, industrial problems arising in practice. MonoProof has successfully generated and verified industrial UNSAT proofs for

[8] We implemented this both via separate SAT calls per lemma; and also by providing all lemmas in a single SAT call (with auxiliary variables to encode the resulting DNF), to allow the solver to re-use learned clauses on different lemmas. The latter approach generally worked better, so we report those results, but (spoiler) neither worked well.

a set of hard, unsatisfiable Tiros [2,8] queries collected in production use at AWS over a multi-week period. However, these instances are proprietary and cannot be published, making them irreproducible by others. Instead, we evaluate on two sets of benchmarks that we can publicly release (also at `https://github.com/NickF0211/MonoProof`):

Network Reachability Benchmarks. These are synthetic benchmarks that mimic the real-world problems solved by Tiros, without disclosing any proprietary information. Network reachability is the problem of determining whether a given pair of network resources (source and destination) can communicate. The problem is challenging because network components can intercept, transform, and optionally re-transmit packets traveling through the network (e.g., a firewall or a NAT gateway). Network components come in various types, each with their own complex behaviors and user-configurable network controls. In these benchmarks, we abstract to two types of intermediate components: simple and transforming. Simple components relay an incoming packet as long as its destination address belongs to a certain domain, expressed in terms of a network CIDR (Classless Interdomain Routing), e.g., 10.0.0.0/24. Transforming network components intercept an incoming packet and rewrite the source address and ports to match their own before re-transmitting it. The simple network components are akin to subnets, VPCs, and peering connections; transforming network components are a highly abstracted version of load balancers, NAT gateways, firewalls, etc. The SMT encoding uses the theories of bit vectors and of graph reachability. The network packets are symbolically represented using bit vectors, and the network is modeled as a symbolic graph. Network behavior is modeled as logical relations between packets and elements in the network graph. Unsatisfiability of a query corresponds to unreachability in the network: for all possible packet headers that the source could generate, and for all possible paths connecting the source to the destination, the combined effect of packet transformations and network controls placed along the path cause the packet to be dropped from the network before it reaches its destination.

We generated 24 instances in total, varying the size and structure of the randomly generated network. Graph sizes ranged from 1513 to 15524 (average 5485) symbolic edges.

Escape Routing Benchmarks. Escape routing is the problem of routing all the signals from a component with extremely densely packed I/O connections (e.g., the solder bumps on a Ball-Grid Array (BGA)) to the periphery of the component, where other routing techniques can be used. For a single-layer printed circuit board (PCB), escape routing is optimally solvable via max-flow, but real chips typically require multiple layers. The multi-layer problem is difficult because the vias (connections between layers) are wider than the wires on a layer, disrupting what routes are possible on that layer. Bayless et al. [11] proposed a state-of-the-art solution using SMMT: max-flow predicates determine routability for each layer on symbolic graphs, whose edges are enabled/disabled by logical constraints capturing the design rules for vias.

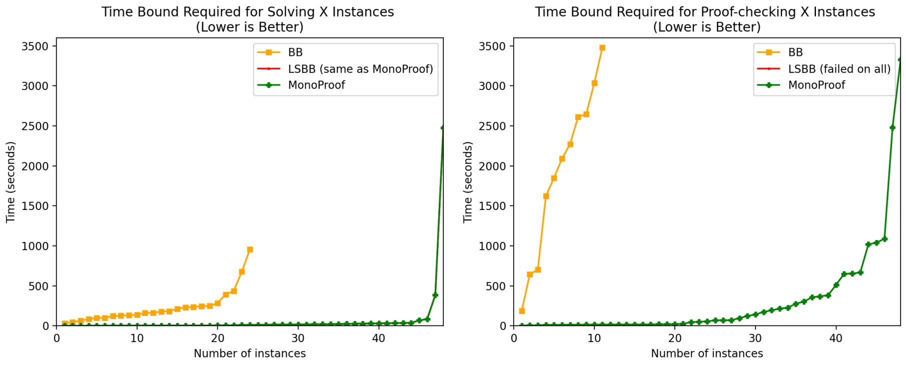

Fig. 3. Cactus Plots for Solving (left) and Proof Preparation&Checking (right). Each point is the runtime for one instance, so the plot shows the number of instances (x-axis) that ran in less than any time bound (y-axis). BB denotes standard bit-blasting; LSBB, lemma-specific bit-blasting; and MonoProof is our new method. The left graph shows that MonoProof (and LSBB, which uses MonoProof's solver) is vastly faster than bit-blasting for solving the instances. The right graph shows that MonoProof is also vastly faster than bit-blasting for proving the result; LSBB timed-out on all proofs.

In [11], 24 commercial BGAs were analyzed under two different via technologies and different numbers of layers. For our benchmark set, we select all configurations where the *provable* minimum number of layers were reported. This results in 24 unsatisfiable SMMT problems instances (routing with one fewer layer than the minimum), which exercise the bit-vector and max-flow theories. Graph sizes ranged from 193994 to 3084986 (average 717705) symbolic edges.

6.2 Results

Returning to the two questions for our evaluation:
1. *The solver overhead of our proof certificate generation is minimal.* On the network reachability benchmarks, the geometric mean (GM) runtime overhead was 14.10% (worst case 28.8%). On the escape routing benchmarks, the GM runtime overhead was only 1.11% (the worst case 5.71%). (The lower overhead is because MonoSAT spent more time learning theory lemmas vs. recording them in the proof.) The overall GM runtime overhead across all benchmarks was 7.41%. These overhead figures are comparable to state-of-the-art, proof-generating SAT solvers, which is not surprising, since our proof certificates are essentially the same as a DRAT proof certificate in SAT. This compares favorably with the solver overhead of heavier-weight, richer, and more expressive SMT proof certificates like LFSC [34].
2. *MonoProof's time to prepare and check a proof of unsatisfiability is markedly faster than standard bit-blasting or lemma-specific bit-blasting.* Fig. 3 summarizes our results. (A full table is in the extended version of this paper.) The left

graph shows solving times (with proof logging). Since the proof-logging overhead is so low for both bit-blasting (Kissat generating DRAT) and MonoProof, these results are consistent with prior work showing the superiority of the SMMT approach for *solving* [9]. Note that bit-blasting (BB) solved all 24 network reachability instances, but failed to solve any of the 24 escape routing instances in the 1hr timeout. Lemma-specific bit-blasting (LSBB) and MonoProof share the same solving and proof-logging steps. The right graph shows proof-checking times (including BackwardCheck and proof-specific Horn upper-bound construction for MonoProof). Here, BB could proof-check only 11/24 reachability instances that it had solved. Restricting to only the 11 instances that BB proof-checked, MonoProof was at least 3.7× and geometric mean (GM) 10.2× faster. LSBB timed out on all 48 instances. Summarizing, MonoProof solved and proved all 48 instances, whereas BB managed only 11 instances, and LSBB failed to prove any.

The above results were with our modified BackwardCheck enabled (*drat-trim-theory* in Fig. 1). Interestingly, with BackwardCheck disabled, MonoProof ran even faster on 37/48 benchmarks (min speedup 1.03×, max 6.6×, GM 1.7×). However, enabling BackwardCheck ran faster in 10/48 cases (min speedup 1.02×, max 7.9×, GM 1.6×), and proof-checked one additional instance (69 sec. vs. 1hr timeout). The modified BackwardCheck is a useful option to have available.

7 Conclusion

We have introduced the first efficient proof-generating method for SMMT. Our approach uses propositional definitions of the theory semantics and derives compact, proof-specific Horn-approximations sufficient to verify the theory lemmas via RUP. The resulting pure DRAT proofs are checkable via well-established (and even machine verified) tools. We give definitions for the most common SMMT theories, and experimental results on industrial-scale problems demonstrate that the solving overhead is minimal, and the proof preparation and checking times are vastly faster than the alternative of bit-blasting.

The immediate line of future work is to support additional finite domain monotonic theories, such as richer properties on pseudo-boolean reasoning. We also aim to apply our approach to support monotonic theories beyond finite domains. In addition, we plan to extend our proof support to emerging proof format such as LRAT [18] and FRAT [3] that enable faster proof checking.

Acknowledgments

Nick Feng was supported in part by an Amazon Research Award. Alan Hu was supported in part by a Discovery Grant from the Natural Sciences and Engineering Research Council of Canada (NSERC). The authors would like to thank Marijn Heule for insightful feedback and mentorship during Nick's internship at Amazon Web Services in 2021. We also thank Dan Dacosta, Nadia Labai, and Nate Launchbury for reviewing earlier drafts of this work.

References

1. Armand, M., Faure, G., Grégoire, B., Keller, C., Théry, L., Werner, B.: A Modular Integration of SAT/SMT Solvers to Coq through Proof Witnesses. In: Jouannaud, J., Shao, Z. (eds.) Certified Programs and Proofs - First International Conference, CPP 2011, Kenting, Taiwan, December 7-9, 2011. Proceedings. Lecture Notes in Computer Science, vol. 7086, pp. 135–150. Springer (2011). https://doi.org/10.1007/978-3-642-25379-9_12, `https://doi.org/10.1007/978-3-642-25379-9_12`
2. Backes, J., Bayless, S., Cook, B., Dodge, C., Gacek, A., Hu, A.J., Kahsai, T., Kocik, B., Kotelnikov, E., Kukovec, J., McLaughlin, S., Reed, J., Rungta, N., Sizemore, J., Stalzer, M., Srinivasan, P., Subotić, P., Varming, C., Whaley, B.: Reachability analysis for AWS-based networks. In: Dillig, I., Tasiran, S. (eds.) International Conference on Computer Aided Verification (CAV). pp. 231–241. Springer (2019)
3. Baek, S., Carneiro, M., Heule, M.J.H.: A Flexible Proof Format for SAT Solver-Elaborator Communication. In: Groote, J.F., Larsen, K.G. (eds.) Tools and Algorithms for the Construction and Analysis of Systems — 27th International Conference, TACAS 2021. Lecture Notes in Computer Science, vol. 12651, pp. 59–75. Springer (2021). https://doi.org/10.1007/978-3-030-72016-2_4, `https://doi.org/10.1007/978-3-030-72016-2_4`
4. Barbosa, H., Blanchette, J., Fleury, M., Fontaine, P., Schurr, H.J.: Better SMT proofs for easier reconstruction. In: AITP 2019-4th Conference on Artificial Intelligence and Theorem Proving (2019)
5. Barbosa, H., Blanchette, J.C., Fontaine, P.: Scalable Fine-Grained Proofs for Formula Processing. In: de Moura, L. (ed.) Automated Deduction - CADE 26 - 26th International Conference on Automated Deduction, Gothenburg, Sweden, August 6-11, 2017, Proceedings. Lecture Notes in Computer Science, vol. 10395, pp. 398–412. Springer (2017). https://doi.org/10.1007/978-3-319-63046-5_25, `https://doi.org/10.1007/978-3-319-63046-5_25`
6. Barrett, C., De Moura, L., Fontaine, P.: Proofs in satisfiability modulo theories. All about proofs, Proofs for all **55**(1), 23–44 (2015)
7. Barrett, C.W., Conway, C.L., Deters, M., Hadarean, L., Jovanovic, D., King, T., Reynolds, A., Tinelli, C.: CVC4. In: Gopalakrishnan, G., Qadeer, S. (eds.) Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings. Lecture Notes in Computer Science, vol. 6806, pp. 171–177. Springer (2011). https://doi.org/10.1007/978-3-642-22110-1_14, `https://doi.org/10.1007/978-3-642-22110-1_14`
8. Bayless, S., Backes, J., DaCosta, D., Jones, B., Launchbury, N., Trentin, P., Jewell, K., Joshi, S., Zeng, M., Mathews, N.: Debugging Network Reachability with Blocked Paths. In: International Conference on Computer Aided Verification (CAV). pp. 851–862. Springer (2021)
9. Bayless, S., Bayless, N., Hoos, H., Hu, A.: SAT modulo monotonic theories. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 29 (2015)
10. Bayless, S., Bayless, N., Hoos, H.H., Hu, A.J.: SAT Modulo Monotonic Theories. In: Bonet, B., Koenig, S. (eds.) Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA. pp. 3702–3709. AAAI Press (2015), `http://www.aaai.org/ocs/index.php/AAAI/AAAI15/paper/view/9951`
11. Bayless, S., Hoos, H.H., Hu, A.J.: Scalable, high-quality, SAT-based multi-layer escape routing. In: Liu, F. (ed.) Proceedings of the 35th International Conference

on Computer-Aided Design, ICCAD 2016, Austin, TX, USA, November 7-10, 2016. p. 22. ACM (2016). https://doi.org/10.1145/2966986.2967072, https://doi.org/10.1145/2966986.2967072

12. Bayless, S., Kodirov, N., Iqbal, S.M., Beschastnikh, I., Hoos, H.H., Hu, A.J.: Scalable Constraint-Based Virtual Data Center Allocation. Artif. Intell. **278**(C) (jan 2020). https://doi.org/10.1016/j.artint.2019.103196, https://doi.org/10.1016/j.artint.2019.103196
13. Biere, A., Fazekas, K., Fleury, M., Heisinger, M.: CaDiCaL, Kissat, Paracooba, Plingeling, and Treengeling Entering the SAT Competition 2020. In: Balyo, T., Froleyks, N., Heule, M., Iser, M., Järvisalo, M., Suda, M. (eds.) SAT Competition 2020 — Solver and Benchmark Descriptions. Department of Computer Science Report Series B, vol. B-2020-1, pp. 51–53. University of Helsinki (2020)
14. Böhme, S.: Proof reconstruction for Z3 in Isabelle/HOL. In: 7th International Workshop on Satisfiability Modulo Theories (SMT'09) (2009)
15. Böhme, S., Weber, T.: Fast LCF-Style Proof Reconstruction for Z3. In: Kaufmann, M., Paulson, L.C. (eds.) Interactive Theorem Proving, First International Conference, ITP 2010, Edinburgh, UK, July 11-14, 2010. Proceedings. Lecture Notes in Computer Science, vol. 6172, pp. 179–194. Springer (2010). https://doi.org/10.1007/978-3-642-14052-5_14, https://doi.org/10.1007/978-3-642-14052-5_14
16. Bouton, T., Oliveira, D.C.B.D., Déharbe, D., Fontaine, P.: veriT: An Open, Trustable and Efficient SMT-Solver. In: Schmidt, R.A. (ed.) Automated Deduction - CADE-22, 22nd International Conference on Automated Deduction, Montreal, Canada, August 2-7, 2009. Proceedings. Lecture Notes in Computer Science, vol. 5663, pp. 151–156. Springer (2009). https://doi.org/10.1007/978-3-642-02959-2_12, https://doi.org/10.1007/978-3-642-02959-2_12
17. Christ, J., Hoenicke, J., Nutz, A.: SMTInterpol: An Interpolating SMT Solver. In: Donaldson, A.F., Parker, D. (eds.) Model Checking Software - 19th International Workshop, SPIN 2012, Oxford, UK, July 23-24, 2012. Proceedings. Lecture Notes in Computer Science, vol. 7385, pp. 248–254. Springer (2012). https://doi.org/10.1007/978-3-642-31759-0_19, https://doi.org/10.1007/978-3-642-31759-0_19
18. Cruz-Filipe, L., Heule, M.J.H., Jr., W.A.H., Kaufmann, M., Schneider-Kamp, P.: Efficient Certified RAT Verification. In: de Moura, L. (ed.) Automated Deduction - CADE 26 - 26th International Conference on Automated Deduction, Gothenburg, Sweden, August 6-11, 2017, Proceedings. Lecture Notes in Computer Science, vol. 10395, pp. 220–236. Springer (2017). https://doi.org/10.1007/978-3-319-63046-5_14, https://doi.org/10.1007/978-3-319-63046-5_14
19. Cruz-Filipe, L., Marques-Silva, J., Schneider-Kamp, P.: Efficient Certified Resolution Proof Checking. In: Legay, A., Margaria, T. (eds.) Tools and Algorithms for the Construction and Analysis of Systems. pp. 118–135. Springer Berlin Heidelberg, Berlin, Heidelberg (2017)
20. Davis, M., Putnam, H.: A Computing Procedure for Quantification Theory. J. ACM **7**(3), 201–215 (1960). https://doi.org/10.1145/321033.321034, http://doi.acm.org/10.1145/321033.321034
21. Fleury, M., Schurr, H.: Reconstructing veriT Proofs in Isabelle/HOL. In: Reis, G., Barbosa, H. (eds.) Proceedings Sixth Workshop on Proof eXchange for Theorem Proving, PxTP 2019, Natal, Brazil, August 26, 2019. EPTCS, vol. 301, pp. 36–50 (2019). https://doi.org/10.4204/EPTCS.301.6, https://doi.org/10.4204/EPTCS.301.6

22. Ganzinger, H., Hagen, G., Nieuwenhuis, R., Oliveras, A., Tinelli, C.: DPLL (T): Fast decision procedures. In: International Conference on Computer Aided Verification. pp. 175–188. Springer (2004)
23. Giesl, J., Aschermann, C., Brockschmidt, M., Emmes, F., Frohn, F., Fuhs, C., Hensel, J., Otto, C., Plücker, M., Schneider-Kamp, P., Ströder, T., Swiderski, S., Thiemann, R.: Analyzing Program Termination and Complexity Automatically with AProVE. J. Autom. Reason. **58**(1), 3–31 (2017). https://doi.org/10.1007/s10817-016-9388-y, https://doi.org/10.1007/s10817-016-9388-y
24. Goldberg, E., Novikov, Y.: Verification of Proofs of Unsatisfiability for CNF Formulas. In: Proceedings of the Conference on Design, Automation and Test in Europe - Volume 1. p. 10886. DATE '03, IEEE Computer Society, USA (2003)
25. Gurfinkel, A., Vizel, Y.: DRUPing for interpolates. In: Formal Methods in Computer-Aided Design, FMCAD 2014, Lausanne, Switzerland, October 21-24, 2014. pp. 99–106. IEEE (2014). https://doi.org/10.1109/FMCAD.2014.6987601, https://doi.org/10.1109/FMCAD.2014.6987601
26. Heule, M.J.H., Kiesl, B., Biere, A.: Strong Extension-Free Proof Systems. J. Automated Reasoning **64**, 533–554 (2020). https://doi.org/10.1007/s10817-019-09516-0
27. Heule, M.J., Hunt, W.A., Wetzler, N.: Trimming while checking clausal proofs. In: 2013 Formal Methods in Computer-Aided Design. pp. 181–188. IEEE (2013)
28. Klenze, T., Bayless, S., Hu, A.J.: Fast, Flexible, and Minimal CTL Synthesis via SMT. In: Chaudhuri, S., Farzan, A. (eds.) Computer Aided Verification. pp. 136–156. Springer International Publishing, Cham (2016)
29. Luckow, K.S., Dimjasevic, M., Giannakopoulou, D., Howar, F., Isberner, M., Kahsai, T., Rakamaric, Z., Raman, V.: JDart: A Dynamic Symbolic Analysis Framework. In: Chechik, M., Raskin, J. (eds.) Tools and Algorithms for the Construction and Analysis of Systems - 22nd International Conference, TACAS 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings. Lecture Notes in Computer Science, vol. 9636, pp. 442–459. Springer (2016). https://doi.org/10.1007/978-3-662-49674-9_26, https://doi.org/10.1007/978-3-662-49674-9_26
30. McMillan, K.L.: Interpolation and SAT-Based Model Checking. In: Jr., W.A.H., Somenzi, F. (eds.) Computer Aided Verification, 15th International Conference, CAV 2003, Boulder, CO, USA, July 8-12, 2003, Proceedings. Lecture Notes in Computer Science, vol. 2725, pp. 1–13. Springer (2003). https://doi.org/10.1007/978-3-540-45069-6_1, https://doi.org/10.1007/978-3-540-45069-6_1
31. de Moura, L.M., Bjørner, N.: Proofs and Refutations, and Z3. In: Rudnicki, P., Sutcliffe, G., Konev, B., Schmidt, R.A., Schulz, S. (eds.) Proceedings of the LPAR 2008 Workshops, Knowledge Exchange: Automated Provers and Proof Assistants, and the 7th International Workshop on the Implementation of Logics, Doha, Qatar, November 22, 2008. CEUR Workshop Proceedings, vol. 418. CEUR-WS.org (2008), http://ceur-ws.org/Vol-418/paper10.pdf
32. de Moura, L.M., Bjørner, N.: Z3: An Efficient SMT Solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings. Lecture Notes in Computer Science, vol. 4963, pp. 337–340. Springer (2008). https://doi.org/10.1007/978-3-540-78800-3_24, https://doi.org/10.1007/978-3-540-78800-3_24

33. Otoni, R., Blicha, M., Eugster, P., Hyvärinen, A.E.J., Sharygina, N.: Theory-Specific Proof Steps Witnessing Correctness of SMT Executions. In: 58th ACM/IEEE Design Automation Conference, DAC 2021, San Francisco, CA, USA, December 5-9, 2021. pp. 541–546. IEEE (2021). https://doi.org/10.1109/DAC18074.2021.9586272, https://doi.org/10.1109/DAC18074.2021.9586272
34. Ozdemir, A., Niemetz, A., Preiner, M., Zohar, Y., Barrett, C.W.: DRAT-based Bit-Vector Proofs in CVC4. In: Janota, M., Lynce, I. (eds.) Theory and Applications of Satisfiability Testing - SAT 2019 - 22nd International Conference, SAT 2019, Lisbon, Portugal, July 9-12, 2019, Proceedings. Lecture Notes in Computer Science, vol. 11628, pp. 298–305. Springer (2019). https://doi.org/10.1007/978-3-030-24258-9_21, https://doi.org/10.1007/978-3-030-24258-9_21
35. Selman, B., Kautz, H.A.: Knowledge Compilation using Horn Approximations. In: Dean, T.L., McKeown, K.R. (eds.) Proceedings of the 9th National Conference on Artificial Intelligence, Anaheim, CA, USA, July 14-19, 1991, Volume 2. pp. 904–909. AAAI Press / The MIT Press (1991), http://www.aaai.org/Library/AAAI/1991/aaai91-140.php
36. Stump, A., Oe, D., Reynolds, A., Hadarean, L., Tinelli, C.: SMT proof checking using a logical framework. Formal Methods Syst. Des. **42**(1), 91–118 (2013). https://doi.org/10.1007/s10703-012-0163-3, https://doi.org/10.1007/s10703-012-0163-3
37. Wetzler, N., Heule, M., Jr., W.A.H.: DRAT-trim: Efficient Checking and Trimming Using Expressive Clausal Proofs. In: Sinz, C., Egly, U. (eds.) Theory and Applications of Satisfiability Testing - SAT 2014 - 17th International Conference, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8561, pp. 422–429. Springer (2014). https://doi.org/10.1007/978-3-319-09284-3_31, https://doi.org/10.1007/978-3-319-09284-3_31
38. Wetzler, N., Heule, M.J.H., Hunt, W.A.: Mechanical Verification of SAT Refutations with Extended Resolution. In: Blazy, S., Paulin-Mohring, C., Pichardie, D. (eds.) Interactive Theorem Proving. pp. 229–244. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)
39. Zhang, L., Malik, S.: Validating SAT Solvers Using an Independent Resolution-Based Checker: Practical Implementations and Other Applications. In: 2003 Design, Automation and Test in Europe Conference and Exposition (DATE 2003), 3-7 March 2003, Munich, Germany. pp. 10880–10885. IEEE Computer Society (2003). https://doi.org/10.1109/DATE.2003.10014, http://doi.ieeecomputersociety.org/10.1109/DATE.2003.10014

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.