Compass: Synthesis of Term Rewriting Systems with Bounded Completeness

MARCEL ULLRICH, Saarland University, Saarland Informatics Campus, Germany
TITOUAN LOZAC'H, Université Paris-Saclay, ENS Paris-Saclay, France
ALBERT GREACA, Saarland University, Saarland Informatics Campus, Germany
SEBASTIAN HACK, Saarland University, Saarland Informatics Campus, Germany

Term rewriting is an important part of many optimizing compilers: the compiler engineer specifies a set of rewrite rules, and the compiler applies them to simplify programs. A rule set is convergent (confluent and terminating) and complete if every term reaches, under any rule application order, a unique normal form that ideally is the smallest term among all semantically equivalent terms. Such rule sets are hard to construct manually, so compilers rely on rule sets without guarantees and heuristic application orders, or on equality saturation, which explores all rewrites in an e-graph but loses its guarantees when time or memory runs out before saturation, which is not uncommon in practice.

We present Compass, an algorithm that synthesizes rewrite rule sets that are convergent and complete for all ground terms up to a given size. Compass builds the rule set inductively over the term size and uses the system synthesized so far to prune the enumeration of candidate terms. We prove ground convergence and ground completeness and mechanize the proof in Lean. For theories with a finite universe, Compass provably terminates with a rule set that is convergent and complete for ground terms of *all* sizes.

In our evaluation on boolean, integer, and bitvector theories, greedy rewriting with Compass's rule sets simplifies terms better and faster than equality saturation. Compass also compares favorably against Ruler and unfailing Knuth-Bendix completion in synthesis time, rule coverage, and simplification power.

1 Introduction

Term rewriting is an important part of the optimizer of many compilers (e.g. [Lopes et al. 2015]). Moreover, rewriting plays a central role in optimizing programs written in domain-specific languages (e.g. [Hagedorn et al. 2020; Jia et al. 2019; Ragan-Kelley et al. 2013]). A rewrite rule is a pair of terms (ℓ, r), written $\ell \longrightarrow r$, expressing that the left term ℓ can be transformed to the equivalent right term r. Optimizing by term rewriting is attractive because it makes optimization *declarative*: the compiler writer merely has to specify a set of rewrite rules but does not have to care about when and how the rules have to be applied.

Rewrite rules must be *sound*, meaning that ℓ and r have to be semantically equivalent. A set of rules is *terminating* if for every term, each sequence of rule applications is finite. This means that each term t can be *reduced* by a finite set of rule applications to a term that cannot be reduced any further. This term is called a normal form of t. A rule set is called *convergent* if each sequence of rule applications applied to t always terminates in the *same unique* normal form. Convergence is desirable because it means that it does not matter in which order we apply the rules: we always get the same result. Termination is typically enforced by establishing that the rules are *compatible* with a well-founded reduction order $\succ$, i.e. $\ell \succ r$ for each rule $\ell \longrightarrow r$. Intuitively, this means that rewrite rules can only *simplify* terms.

In addition to convergence, it is also desirable that a rule set is *complete*. Complete here means that two semantically equivalent terms have the same normal form. Together with confluence, this gives a very strong property: every term can be *normalized* to its semantically equivalent normal

Authors' Contact Information: Marcel Ullrich, ullrich@cs.uni-saarland.de, Saarland University, Saarland Informatics Campus, Saarbrücken, Germany; Titouan Lozac'h, titouan.lozac_h@ens-paris-saclay.fr, Université Paris-Saclay, ENS Paris-Saclay, Gif-sur-Yvette, France; Albert Greaca, algr00003@uni-saarland.de, Saarland University, Saarland Informatics Campus, Saarbrücken, Germany; Sebastian Hack, hack@cs.uni-saarland.de, Saarland University, Saarland Informatics Campus, Saarbrücken, Germany.

form by applying rules in any order. This normal form is then also the *least* (or optimal) term with respect to the reduction order. Typically, complete and convergent rewrite systems are hard, or even impossible, to obtain in practice. Therefore, compilers resort to manually engineered rule sets and heuristics for rule application order, both of which may lead to suboptimal results.

One promising solution to the rule application order problem that has been studied extensively in the past is *equality saturation (EqSat)* [Tate et al. 2011]. EqSat gives up on convergence and applies all rules everywhere possible. It relies on an *e-graph* to represent equivalence classes of terms by hash-consing to efficiently store *all* terms that originate from rewriting. EqSat stops when the e-graph has saturated, i.e. no rule application yields a term that is not yet stored in the e-graph. After saturation, an *extraction* algorithm is used to extract the best version of the optimized program from the e-graph. In practice, however, EqSat rarely reaches saturation and has to be interrupted due to excessive use of time or memory. In this case, even optimal extraction algorithms are not guaranteed to extract the optimal program because not all possible rewrites have been carried out.

In this paper, we explore the automated generation of rule sets that are *complete and convergent* for terms *up to a given size* with the goal of replacing compile-time expensive EqSat with simple and efficient greedy rewriting.

Existing rewrite rule set generation approaches do not give such guarantees: Ruler [Nandi et al. 2021] focuses on generating rewrite rules for EqSat. Their focus is on generating small rule sets that are as general as possible. Ruler's rule sets specifically do not follow a term order as they aim to be as general as possible. Therefore, they are not *simplifying* on their own but rely on EqSat to rewrite to a "good" term.

Knuth-Bendix completion (KBC) [Knuth and Bendix 1970], and especially unfailing KBC [Bachmair et al. 1989], is a semi-decision procedure for the *word problem*, i.e. the question of deciding if two terms are equal given an axiomatization (a set of equalities). The idea of KBC is to turn the axiomatization into a rewrite rule set that is compatible with a reduction order so that equality can be decided simply by rewriting. Hence, *if* it terminates, KBC, like Compass, produces complete and convergent rule sets. It establishes convergence by making rules compatible with a reduction order and by creating new equalities based on so-called *critical pairs*: two terms ℓ, r for which there is another term u such that $\ell \leftarrow u \rightarrow r$. The new equality can then be turned into a rewrite rule if one of the two terms is smaller than the other in the reduction order.

In terms of guarantees, Compass is similar to KBC: Compass takes a positive integer n as a parameter and produces a rewrite system that is convergent and complete for all ground terms of size $\leq n$. KBC guarantees convergence and completeness only for the final system and not for intermediate systems up to a specific term size. Both approaches also share that they produce rewrite rule sets that are compatible with a reduction order, which guarantees termination. The algorithm of Compass itself is, however, significantly different from KBC. Compass enumerates, like Ruler, terms that are candidates for left-hand sides of rewrite rules. It enumerates these terms by size and uses several techniques to prune a significant portion of the enumeration space. It relies on a decision procedure (an SMT solver) for equality to identify adequate right-hand sides for left-hand side candidates, whereas KBC uses the critical pair construction to derive new rewrite rules. The rationale of Compass is that convergence up-to-size-n is "good enough" for optimizing large terms succinctly.

In summary, this paper makes the following contributions:

- We present Compass, an algorithm to automatically generate rewrite rule sets that are *convergent* and *complete* up to a given term size. The algorithm builds the rule set incrementally by enumerating left-hand sides in order of increasing size and uses previously generated

rules to reduce the number of left-hand sides for which we have to synthesize right-hand sides.
- We prove the algorithm correct (ground convergence and ground completeness of the produced rewrite system, Section 4) and accompany the paper-and-pencil proof in this paper with a mechanized proof in Lean.
- We show that for finite theories (e.g. bitvectors, booleans) there is always a term size k such that the rewrite system Compass produces for k is convergent and complete for *all* ground terms over this theory (Section 5). Therefore, for finite theories, Compass *terminates*. However, for many practically relevant theories, this k is unrealistically high. We further evaluate the complexity of Compass quantitatively and experimentally.
- In Section 6, we experimentally evaluate the simplification power of the rewrite systems Compass generates in the setting of greedy rewriting on randomly generated terms of various sizes (from 50 to 3000 nodes) on four different theories (boolean, integers, bitvectors of size 4 and 32). Our experiments show that greedy rewriting using the "convergent and complete up to size n" rewrite systems that Compass generates consistently simplifies terms better and faster than EqSat-based rewriting.

The remainder of this paper is structured as follows: Section 2 reviews the important notions from term rewriting that are necessary for the remainder of the paper. Section 3 gives a short informal overview of the basic ideas of Compass. Section 4 discusses Compass in detail and develops the algorithm based on several refinements to a simple base version. We accompany the treatment with a proof of ground convergence and completeness. Section 5 proves that Compass can produce ground convergent and complete rewrite systems for terms *of all sizes* for finite theories. It also provides a quantitative evaluation of a run of Compass on the boolean theory that produces a ground convergent and complete rewrite system *for terms of all sizes*. Section 6 contains the experimental evaluation and a comparison to other tools and rewriting strategies. Section 7 discusses related work. Section 8 and Section 9 conclude the paper.

2 Terms and Term Rewriting Systems

In this section, we review the basic notions of terms and term rewriting that are relevant for this paper. For further details, we refer to Baader and Nipkow [1998]. A *signature* Σ is a set of function symbols, each associated with an *arity*. We write $f/2 \in \Sigma$ to say that there is a function symbol f with arity 2 in Σ. If the arity does not matter in a particular context, we also write $f \in \Sigma$. Function symbols of arity 0 are called *constants*.

Given a set of *variables* V and a *signature* Σ, the set of Σ-terms is recursively defined as:

- each $X \in V$ is a term.
- if $t_1, \ldots, t_n$ are terms then $f(t_1, \ldots, t_n)$ is a term if $f/n \in \Sigma$.

A term that does not contain any variable is called a *ground term*. We denote signature elements with lower-case letters and variables with upper-case letters.

The size $|t|$ of a term t is defined to be the number of function symbols and variables of t:

$$|t| := \begin{cases} 1 + \sum_{i=1}^{n} |t_i| & \text{if } t = f(t_1, \ldots, t_n) \\ 1 & \text{otherwise} \end{cases}$$

For each element $\alpha \in \Sigma \cup V$, we write $|t|_\alpha$ for the number of occurrences of α in t.

Semantics. A ground term gets a semantics through a *model*. A model $\mathcal{A}$ for a signature Σ consists of a set A, called the *universe* of $\mathcal{A}$, and a mapping $[\![\cdot]\!]_{\mathcal{A}}$ that assigns each function symbol $f/n \in \Sigma$ a total function $[\![f]\!]_{\mathcal{A}} : A^n \to A$.

A *theory* $\mathcal{T}$ restricts the set of possible models by fixing a universe and *interpreting* some of the function symbols. It thereby splits the signature into an interpreted and an uninterpreted part: $\Sigma = \Sigma_I \uplus \Sigma_U$. For example, in the theory of integers, the formula $2 + x$ contains the interpreted constant 2, the uninterpreted constant (UC) x, and the interpreted function symbol +. Hence, a theory $\mathcal{T}$ can be described as a tuple $(A, \Sigma_I, [\![\cdot]\!]_{\mathcal{T}})$ of a universe, a set of interpreted function symbols, and an interpretation that gives each interpreted function symbol a semantics. Given a model and a theory, a ground term t is assigned a value $[\![t]\!]_{\mathcal{A},\mathcal{T}}$ by an *interpretation*:

$$[\![\tau(t_1, \ldots, t_n)]\!]_{\mathcal{A},\mathcal{T}} \coloneqq f([\![t_1]\!]_{\mathcal{A},\mathcal{T}}, \ldots, [\![t_n]\!]_{\mathcal{A},\mathcal{T}}) \qquad f = \begin{cases} [\![\tau]\!]_{\mathcal{A}} & \text{if } \tau \in \Sigma_U \\ [\![\tau]\!]_{\mathcal{T}} & \text{if } \tau \in \Sigma_I \end{cases}$$

We say that two ground terms s, t are semantically equivalent if the interpretations $[\![s]\!]_{\mathcal{A},\mathcal{T}}$ and $[\![t]\!]_{\mathcal{A},\mathcal{T}}$ are equal for every model $\mathcal{A}$ of Σ_U and write $s \equiv t$.

In a concrete application, for example a compiler, one uses uninterpreted function symbols, especially UCs, to represent terms that are parameterized. For example, the term $x + 1$ could be taken from a program in which x is a declared variable. Note that a UC is different from a variable in a term as defined above: a UC x is a nullary element of the signature Σ whereas a variable X is an element of V. Variables in the context of term rewriting are typically only used in left- or right-hand sides of rewrite rules (see below). When a rewrite rule is matched to a larger term, the variables of the left-hand side represent the subterms of the match. From now on, we will assume that the only uninterpreted function symbols in our signature are constants.

Rewriting. A rewrite rule $\ell \to r$ consists of two terms ℓ and r such that the variables in r are a subset of the set of variables of ℓ, and ℓ is not a variable. A *reducible expression (redex)* of a rule $\ell \to r$ in a term t is a pair (p, σ) of a *position* p and a substitution σ. The position p (which we do not define formally for the sake of brevity) identifies a subterm $[t]_p$ in t. The substitution σ maps the variables of ℓ to subterms of $[t]_p$ such that $\ell\sigma = [t]_p$. Rewriting t by a rule $R : \ell \to r$ means identifying a redex (p, σ) and replacing $[t]_p$ in t by $r\sigma$, yielding the new term t'. We then write $t \to_R t'$. The requirement that the set of variables in the rule's right-hand side must be a subset of the left-hand side's variables ensures that rewriting introduces no variables to the (ground) term t. The requirement that ℓ is not a variable avoids rules which are always applicable.

A set of rewrite rules is called a term rewriting system (TRS). Given a term rewriting system S and two terms t and t', we write $t \to t'$ if there exists a rule R in S such that $t \to_R t'$. We write $\to^*$ for the reflexive transitive closure of $\to$. A term t is called

- *reducible* if there exists a rule $R \in S$ and a term t' such that $t \to_R t'$.
- *in normal form* if it is not reducible.
- *a normal form* of some term t' if $t' \to^* t$ and t is in normal form.

Two terms t_1, t_2 are called *joinable* if there exists a term t such that $t_1 \to^* t \leftarrow^* t_2$. A TRS is called:

- *terminating* if there is no infinite chain $t_1 \to t_2 \to \cdots$.
- *locally confluent* if for all terms $t_1 \leftarrow t \to t_2$, t_1 and t_2 are joinable.
- *confluent* if for all terms $t_1 \leftarrow^* t \to^* t_2$, t_1 and t_2 are joinable.

It follows directly that in a confluent and terminating TRS, two terms $t_1 \leftrightarrow^* t_2$ have a unique normal form. We then say that the TRS is *convergent*. Furthermore, Newman's lemma states that if a TRS is locally confluent and terminating, it is also confluent.

Definition 2.1 (Completeness). A convergent TRS is called *complete* if and only if for every pair of semantically equivalent terms (t, s), t and s have the same normal form.

Reduction Orders. A *reduction order* $\succ$ is an irreflexive, transitive, and well-founded order on terms that is closed under substitutions (for all substitutions σ: $s \succ t$ implies $s\sigma \succ t\sigma$) and contexts (for all terms u: $s \succ t$ implies $u(\dots, s, \dots) \succ u(\dots, t, \dots)$).

An important well-known fact is that if all rules in a TRS are compatible ($\ell \to r$ implies $\ell \succ r$) with a given reduction order, then all sequences of rule applications of the TRS always *terminate*, and each term has a (not necessarily unique) normal form.

Definition 2.2 (Knuth-Bendix order). One prominent reduction order is the Knuth-Bendix order (KBO). For the sake of brevity, we do not go into the full detail of how a KBO is defined (all details can be found for example in [Baader and Nipkow 1998]). Important for this paper is that a KBO is parameterized by a weight function $w : \Sigma \cup V \to \mathbb{R}_0^+$ that assigns a non-negative weight to each function symbol, and a total order $>_\Sigma$ on the function symbols, called the *precedence*. Then, two terms ℓ, r satisfy $\ell \succ r$ if the following conditions are both satisfied:

(1) the number of occurrences of each variable in r is less than or equal to that in ℓ, i.e. $|\ell|_X \geq |r|_X$ for each variable $X \in V$
(2) the *weight function* w lifted to terms is strictly greater on ℓ than on r (i.e. $w(\ell) > w(r)$), where $w(t)$ is defined as:

$$w(t) \coloneqq \sum_{f \in \Sigma} w(f) \cdot |t|_f + \sum_{X \in V} w_0 \cdot |t|_X ,$$

where $w_0 \in \mathbb{R}^+$ is some positive constant that acts as the weight of a variable. If two terms have the same weight, $>_\Sigma$ is used to try to break ties lexicographically. If that is not possible (because some subterms contain variables), the terms are incomparable.

Each KBO is *total* on ground terms. To see this, consider two ground terms s and t: the first requirement (variable occurrences of the right-hand side) is trivially fulfilled because ground terms have no variables. The second requirement also applies: if $w(s) = w(t)$, ties can always be broken lexicographically due to the total order on the function symbols. Only terms with variables can be unordered, for example $X + Y \not\succ Y + X$ and $Y + X \not\succ X + Y$. The reason is that, because there is no order on variable symbols, ties cannot be broken lexicographically.

Definition 2.3 (Size-KBO). A KBO for which $w_0 = 1$ and $w(f) = 1$ for each $f \in \Sigma$ is called a size-KBO.

Note that in a size-KBO, the term weight function $w(t)$ is equal to the size of the term $|t|$. When using the symbol $\succ$, from now on, we mean some size-KBO. The rewrite rules Compass synthesizes will always be compatible with such a size-KBO for a given precedence $>_\Sigma$ and therefore terminate.

3 Overview

Given the following ingredients

- a theory $(A, \Sigma_I, [\![\cdot]\!]_{\mathcal{T}})$ with decidable equality,
- a finite set of k UCs Σ_U that together with Σ_I forms the signature Σ,
- a precedence $>_\Sigma$ over the symbols in $\Sigma = \Sigma_I \uplus \Sigma_U$,
- a finite set of variables V,
- some size $n \in \mathbb{N}$,

Property 1. Compass builds a TRS $\mathbb{T}_n$ that is complete (and thereby terminating and confluent) for *ground* terms up to a given size n. In $\mathbb{T}_n$, each ground term t has one unique normal

form which is, among all the (ground) terms that are semantically equivalent to t, the one that is *least* with respect to the size-KBO induced by $>_\Sigma$.

Compass follows the idea of related work [Nandi et al. 2021] of enumerating terms and constructing rules between enumerated terms based on semantic equivalence. However, in contrast to Ruler, Compass uses the structure of the TRS it generates to avoid enumerating a significant number of terms. Moreover, Compass is guaranteed to terminate on finite theories even though the number of terms is infinite.

The fundamental idea of Compass is to build $\mathbb{T}_n$ inductively: assume we have a ground complete TRS $\mathbb{T}_{n-1}$. How can we extend $\mathbb{T}_{n-1}$ to a ground complete system $\mathbb{T}_n$?

To keep things simple and to gain some intuition first, let us focus on ground terms only and ignore variables (the elements of V) for now. To establish Property 1 in the box above for $\mathbb{T}_n$, we need to ensure that each term t of size n has exactly one normal form that is the $\succ$-least term that is semantically equivalent to t. *Conceptually,* we investigate *every* term t of size n and reduce it using the rules of $\mathbb{T}_{n-1}$. This will give us a set N_t of normal forms of t with respect to $\mathbb{T}_{n-1}$. This set is well-defined because rewriting with $\mathbb{T}_{n-1}$ always terminates: as we will see, every rule $\ell \to r \in \mathbb{T}_{n-1}$ is $\succ$-compatible (i.e. $\ell \succ r$), and $\succ$ is well-founded. Note that $\mathbb{T}_{n-1}$ is not necessarily confluent on terms of sizes larger than or equal to n, only on terms of size less than n, by assumption. Therefore, we might get, for each term t of size n, multiple normal forms or even no normal form at all in the case when t is not reducible by $\mathbb{T}_{n-1}$. Let us now examine the set N_t more closely:

(1) N_t is empty. In this case, none of the rules in $\mathbb{T}_{n-1}$ was applicable to t. Hence, t is a normal form with respect to $\mathbb{T}_{n-1}$.
(2) N_t is not empty. Let us consider each $t' \in N_t$. Note that t' cannot be larger than n because each rule is $\succ$-compatible and therefore terms can never increase in size by rewriting. If the size of t' is less than n, then t' is the $\succ$-least term that is semantically equivalent to t by induction. If the size of t' is equal to n, it is a normal form with respect to $\mathbb{T}_{n-1}$.

The question now is if a term t' of size n that is a normal form with respect to $\mathbb{T}_{n-1}$ is also a normal form in $\mathbb{T}_n$: since $\mathbb{T}_n$ is supposed to be ground complete up to size n, no two normal forms can be semantically equivalent. If they were, they would be—by the nature of $\mathbb{T}_n$—reducible to a common unique semantically equivalent normal form and therefore could not be normal forms themselves. There are now two cases:

(1) There is another term $t'' \neq t'$ of size less than or equal to n that is semantically equivalent to t' and a normal form with respect to $\mathbb{T}_{n-1}$. Because $\succ$ is total on ground terms, t' and t'' are ordered by $\succ$. Without loss of generality, let $t' \succ t''$. To preserve Property 1, we add a rule $t' \to t''$ to $\mathbb{T}_n$.
(2) There is no such term. In this case t' will be a normal form in $\mathbb{T}_n$ and in fact a normal form in $\mathbb{T}_k$ for all k.

These considerations give each rule $\ell \to r$ we add to $\mathbb{T}_n$ the following properties:

(1) r is the $\succ$-least term that is semantically equivalent to ℓ. This implies $\ell \succ r$.
(2) r is a normal form of $\mathbb{T}_k$ for all k.
(3) ℓ is a normal form of $\mathbb{T}_{n-1}$.

This gives us an important criterion to focus our enumeration of potential left-hand sides when we build $\mathbb{T}_n$ from $\mathbb{T}_{n-1}$:

Because the left-hand side ℓ of each rule is not reducible by $\mathbb{T}_{n-1}$, the subterms of ℓ must be normal forms in $\mathbb{T}_{n-1}$. Hence, instead of enumerating all terms of size n, we only enumerate all

terms of size n that can be entirely constructed from the normal forms of $\mathbb{T}_{n-1}$, i.e. that have the shape $\tau(t_1, \ldots, t_m)$ where the t_i are normal forms of $\mathbb{T}_{n-1}$ of size less than n and $\tau \in \Sigma$. This cuts down the number of terms to be enumerated significantly.

Rules with Variables. Up to now, we have only considered rewrite rules in which the left- and right-hand sides were ground terms. Such rules are only suitable for rewriting a host term *bottom up*, always matching at the leaves. The reason is that a ground-term left-hand side only contains uninterpreted or interpreted constants at its leaves and therefore cannot match a term in the context of another one, e.g. $x + 0 \to x$ cannot match $(y + z) + 0$ whereas the rule $X + 0 \to X$ would reduce $(y + z) + 0$ to $y + z$.

It is important to understand that variables in rewrite rules basically stand for arbitrary subterms in the context of matching. Analogously, a rule with variables can also be seen as an infinite set of rules in which each rule corresponds to a specific instantiation of the variables with ground terms. Hence, rules with variables are more "powerful" because they subsume an infinite set of ground term rules.

Compass can enumerate left-hand sides with variables as well. However, because some terms with variables might not be ordered by $\succ$, we cannot always guarantee that there is a least normal form for a term with variables. In that case, we have to resort to ground instantiations of that left-hand side. For example, $X + Y$ and $Y + X$ are semantically equivalent, but they do not have a unique normal form. Hence, there is no rewrite rule with either $X + Y$ or $Y + X$ as its left-hand side. However, assuming $\Sigma_U = \{x, y, z\}$ and $z >_\Sigma y >_\Sigma x$, we have the *ground* rules $y + x \to x + y$, $z + x \to x + z$, and $z + y \to y + z$.

4 Compass

In this section, we present the Compass algorithm and investigate its properties. For the rest of the paper, we assume the ingredients specified at the beginning of Section 3. We will develop the full Compass algorithm step by step, adding a series of refinements that each make it more efficient. We establish the correctness of each refinement at the end of the respective subsection. In these proofs it is helpful to not prove Property 1 directly but to prove an invariant (Property 2) that implies Property 1. This invariant hinges on the concept of a *pre-normal form:*

Definition 4.1 (Pre-normal form). Given a term t, a pre-normal form t^* of t is a term that is semantically equivalent to t ($t \equiv t^*$), less than or equal to t in the KBO ($t \succeq t^*$), and $\succ$-minimal: there is no term t' that is semantically equivalent to t for which $t^* \succ t'$.

Note that a term can also be a pre-normal form of itself. Eventually, some pre-normal forms will become the normal forms of the TRS $\mathbb{T}_n$. Note that the distinction between a pre-normal form and a normal form is subtle: the notion of a normal form is based on an underlying TRS (a normal form cannot be reduced with any of the rules in the TRS) whereas a pre-normal form is defined solely with respect to a given term.

Note that a non-ground term can in general have multiple $\succ$-incomparable pre-normal forms: for example, $X + Y + 0$ has two pre-normal forms: $X + Y$ and $Y + X$. Since X, Y are variables, both pre-normal forms are unordered with respect to $\succ$. However, for a ground term t, the pre-normal form is unique because all terms that are semantically equivalent to t are ground as well and all ground terms are totally ordered by $\succ$.

The general idea of $\mathbb{T}_n$ is that each term t of size at most n can be reduced to a *pre-normal form* t^*:

Property 2 (Invariant of Compass). Given some $n \in \mathbb{N}$, each rule $\ell \to r \in \mathbb{T}_n$ is KBO-compatible ($\ell \succ r$), semantics-preserving ($l \equiv r$), and each normal form of a term of size at most n is one of its pre-normal forms.

Let us now establish that each TRS $\mathbb{T}_n$ that satisfies Property 2 also satisfies the central Compass property (Property 1).

Theorem 4.2. *Given some $n \in \mathbb{N}$, each TRS $\mathbb{T}_n$ that satisfies Property 2 is terminating for all terms of size at most n and convergent for all* ground terms *of size at most n.*

Proof. *Termination.* $\mathbb{T}_n$ is terminating because all rules are KBO-compatible and a KBO is well-founded.

Confluence on ground terms. Let t be a ground term of size $|t| \leq n$. By Newman's lemma (see Section 2), we know that a terminating TRS is (ground) confluent if and only if it is locally (ground) confluent. We therefore show that $\mathbb{T}_n$ is locally ground confluent.

Suppose that $v \xleftarrow{\mathbb{T}_n} t \xrightarrow{\mathbb{T}_n} u$. Since $\mathbb{T}_n$ is terminating, u and v both have normal forms u' and v'. Since u' is a normal form of u, it is also a normal form of t. As t is ground with $|t| \leq n$, by Property 2, u' is a pre-normal form of t. This implies that u' is $\succ$-minimal. Similarly, v' is $\succ$-minimal.

Because each rewrite rule in $\mathbb{T}_n$ is KBO-compatible, no rewrite step can introduce a new variable (cf. first criterion in Definition 2.2). Therefore, since t is ground, u' and v' are ground as well. As $\succ$ is total on ground terms, either $u' \succ v'$, $v' \succ u'$ or $u' = v'$. Because both u' and v' are semantically equivalent to t, they are also semantically equivalent to one another. Hence, $u' \succ v'$ or $v' \succ u'$ would contradict the $\succ$-minimality of u' and v', which are both pre-normal forms of t. Therefore, $u' = v'$ which proves confluence.

Completeness on ground terms. Consider two ground terms s and t such that $|s| \leq n$ and $|t| \leq n$. Assume $s \equiv t$. By ground confluence and termination of $\mathbb{T}_n$, s (resp. t) has a unique normal form s' (resp. t'). Let us show that $s' = t'$: since s' is the normal form of s, by Property 2, s' is a pre-normal form of s. In particular, s' is $\succ$-minimal. t' is $\succ$-minimal as well by the same argument. As s and t are ground terms, s' and t' are ground terms as well. Since $\succ$ is total on ground terms, it follows that either $s' \succ t'$, $t' \succ s'$ or $s' = t'$. Moreover, $s' \equiv t'$ because $s' \equiv s$ and $t' \equiv t$ by preservation of the semantics, and $s \equiv t$ by hypothesis. $s' \succ t'$ would contradict the $\succ$-minimality of s' and $t' \succ s'$ would contradict the $\succ$-minimality of t'; therefore, $s' = t'$. s and t have the same unique normal form. □

4.1 The Basic Algorithm

We will start with a very simple and inefficient algorithm to establish the core invariant of Compass. The basic version of Compass relies entirely on a *(presumably costly)* oracle that can tell us some pre-normal form for a given term t. For the sake of presentation, we require this oracle to be deterministic: given a term t, the oracle will always answer with the same term t^*. Note that this requirement is not strictly necessary to prove Property 1 for $\mathbb{T}_n$. Our Lean development does not require a deterministic oracle.

Basic Algorithm. Enumerate all terms of sizes 1 to n. For each term t we enumerate, query the oracle and get a pre-normal form t^* of t. Then add a rule $t \to t^*$ to $\mathbb{T}_n$ if $t^* \neq t$.

Theorem 4.3. *The basic algorithm satisfies Property 2.*

Proof. By the nature of the oracle, we get a pre-normal form t^* for each term t we enumerate. By the definition of a pre-normal form (Theorem 4.1), t^* is less than or equal to t and semantically equivalent. Since we add the rule only if $t \neq t^*$, the rule is $\succ$-compatible and semantics-preserving.

Each right-hand side of a rule is also a normal form in $\mathbb{T}_n$: if there were a term t' to which a right-hand side t^* could reduce in $\mathbb{T}_n$, then t^* would not be KBO-minimal, which contradicts the fact that t^* is a pre-normal form. □

We will now present multiple refinements of this simple algorithm to reduce the number of rules that have to be inserted and to reduce the number of terms that have to be enumerated.

4.2 Reducing the Number of Rules (Refinement 1)

The first extension to the simple algorithm of the previous subsection reduces invocations of the oracle significantly by using induction:

Assume we have $\mathbb{T}_{n-1}$ that satisfies Property 2. Enumerate each term t of size n. Before asking the oracle, use the rules in $\mathbb{T}_{n-1}$ to simplify t. This gives us a set N_t of all normal forms of t with respect to $\mathbb{T}_{n-1}$. As indicated in Section 3, there are the following cases:

(1) N_t is empty. Therefore, none of the rules in $\mathbb{T}_{n-1}$ was applicable to t. Hence, t is a normal form with respect to $\mathbb{T}_{n-1}$.
(2) N_t is not empty. Let us consider each $t' \in N_t$. Note that t' cannot be larger than n because each rule is $\succ$-compatible and therefore terms can never increase in size by rewriting. If the size of t' is less than n, then t' is the $\succ$-least term that is semantically equivalent to t by induction ($\mathbb{T}_{n-1}$ satisfies Property 2, i.e. the normal form of a term with size less than n is one of its pre-normal forms). If the size of t' is equal to n, it is a normal form with respect to $\mathbb{T}_{n-1}$.

The case distinction shows that either we manage to reduce t to a term of size less than n, in which case $\mathbb{T}_{n-1}$ allows us to reduce it to one of its pre-normal forms (because $\mathbb{T}_{n-1}$ satisfies Property 2) or we get a set of terms *of size n* that are normal forms with respect to $\mathbb{T}_{n-1}$. Note that the "or" in the previous sentence is *inclusive*: both can be true. The reason is that multiple rules of $\mathbb{T}_{n-1}$ might match on t and that $\mathbb{T}_{n-1}$ is not necessarily confluent on terms of size n. Also note that each of these terms of size n will itself appear in our iteration at some point.

Algorithmically, this means that for each enumerated term t we check if some rule of $\mathbb{T}_{n-1}$ matches. If yes, we do not have to do anything (because $\mathbb{T}_{n-1}$ gets us to the desired normal form already). If no, we ask the oracle for a pre-normal form t^* of t. If $t \neq t^*$, we add a rule $t \to t^*$ to $\mathbb{T}_n$; if not, t will be a normal form in $\mathbb{T}_n$. Note again that this means that the left-hand side of each inserted rule is a normal form in $\mathbb{T}_{n-1}$ and the right-hand side is a normal form in $\mathbb{T}_k$ for any k.

The following theorem formalizes this reasoning.

Theorem 4.4. *Given a TRS $\mathbb{T}_{n-1}$ that fulfills Property 2, the TRS $\mathbb{T}_n$ that Refinement 1 builds from $\mathbb{T}_{n-1}$ fulfills Property 2.*

Proof. Consider an n and $\mathbb{T}_n$ that has been built by Refinement 1 and assume $\mathbb{T}_{n-1}$ fulfills Property 2. Consider a rule $\ell \to r \in \mathbb{T}_n$. If $\ell \to r \in \mathbb{T}_{n-1}$, then, as $\mathbb{T}_{n-1}$ satisfies Property 2, this rule is KBO-compatible ($\ell \succ r$) and semantics-preserving ($\ell \equiv r$). Otherwise, $\ell \to r \in (\mathbb{T}_n \setminus \mathbb{T}_{n-1})$. The only case where a rule is added is case (1), when the pre-normal form ℓ^* of ℓ is not ℓ. The rule $\ell \to \ell^*$ has been added; by identification $\ell^* = r$. This means that r is the pre-normal form of ℓ. By definition of a pre-normal form, $\ell \succ r$ and $\ell \equiv r$.

Now consider a term t of size $|t| \leq n$.

- If $|t| < n$, then no rule of $(\mathbb{T}_n \setminus \mathbb{T}_{n-1})$ can be applied to t, as the left-hand side of any such rule has size n. This implies that all normal forms of t (with respect to $\mathbb{T}_n$) are also normal forms of t with respect to $\mathbb{T}_{n-1}$. As $\mathbb{T}_{n-1}$ fulfills Property 2, any normal form of t is one of its pre-normal forms.

- Otherwise, $|t| = n$. For all terms $u \preceq t$, call $H(u)$ the following property: "if u^* is a normal form of u, then u^* is a pre-normal form of u". We will show by induction on $\succ$ (which is well-founded) that $H(u)$ holds for any $u \preceq t$.
  - *Base case:* Let $u \preceq t$ be $\succ$-minimal. As u is $\succ$-minimal and rules are KBO-compatible, u is its only normal form. u is a term of size at most n. If $|u| < n$, as $\mathbb{T}_{n-1}$ fulfills Property 2, then u is a pre-normal form of u. Otherwise, $|u| = n$. In that case, N_u is necessarily empty, and the oracle returns u^*, a pre-normal form of u. u^* is semantically equivalent to u, and $u \succeq u^*$. As u is $\succ$-minimal by hypothesis, $u = u^*$. Therefore, u is a pre-normal form.
  - *Induction case:* Let $u \preceq t$ be a term such that for all terms v such that $u \succ v$, $H(v)$ holds. Let us show that $H(u)$ holds. Let s be a normal form of u with respect to $\mathbb{T}_n$. If $|s| < n$, then as $\mathbb{T}_{n-1}$ fulfills Property 2, s is a pre-normal form of itself. As $u \to^* s$, s is also a pre-normal form of u ($u \equiv s$, $u \succeq s$ and s is $\succ$-minimal). Otherwise, $|s| = n$.
    * If $u = s$, then N_u is empty, and the oracle was queried and returned some pre-normal form of u called u^*. Since $u = s$ is a normal form, no rule has been added. As N_u is empty, case (1) applied, and as no rule was added, $u = u^* = s$. Therefore, s is a pre-normal form of u.
    * Otherwise, $u \neq s$, and since $u \succeq s$ and the rewrites are KBO-compatible, $u \succ s$. Since $u \succ s$, we can use the induction hypothesis on s: any normal form of s is a pre-normal form of s. As s is a normal form, it is its own normal form. Hence s is its own pre-normal form. As $u \to^+ s$, s is a pre-normal form of u.

  In conclusion, $H(u)$ holds for any $u \preceq t$, and in particular it holds for t.

  Any normal form of t is one of its pre-normal forms.

□

As $\mathbb{T}_0 = \emptyset$ fulfills Property 2 (since there are no terms of size 0), we directly obtain that for any $n \in \mathbb{N}$, the TRS $\mathbb{T}_n$ that Refinement 1 builds fulfills Property 2.

4.3 Reducing the Number of Terms to Enumerate (Refinement 2)

This extension builds on the previous one but reduces the number of terms we have to enumerate significantly. The considerations of Refinement 1 showed that whenever we add a rule to $\mathbb{T}_n$, the left-hand side of the rule is a normal form with respect to $\mathbb{T}_{n-1}$. Hence, instead of enumerating every term of size n, we can limit ourselves to terms that are not reducible under $\mathbb{T}_{n-1}$. How can we do that? The important observation is that for a term t of size n to not be reducible under $\mathbb{T}_{n-1}$, the subterms of t must be normal forms of $\mathbb{T}_{n-1}$. If t had a subterm that was not a normal form in $\mathbb{T}_{n-1}$, that subterm would constitute a redex for a rule of $\mathbb{T}_{n-1}$. Therefore, we can limit the enumeration for $\mathbb{T}_n$ to terms t that have the following properties:

$$\tau(t_1, \ldots, t_m) \quad \text{with} \quad |\tau(t_1, \ldots, t_m)| = n, \tau/m \in \Sigma, \text{and each } t_i \text{ is a normal form in } \mathbb{T}_{n-1}$$

Note that such a term can, however, have one potential redex for a rule of $\mathbb{T}_{n-1}$ at the root. Therefore, we need to check for each such term if there is a rule of $\mathbb{T}_{n-1}$ that matches the root of the term. If so, we skip the term; otherwise, we get a pre-normal form from the oracle and insert a rule if the pre-normal form is different from the term. The following theorem justifies this refinement.

Theorem 4.5. *Given some $n \in \mathbb{N}$, denote by $\mathbb{T}_n^1$ the TRS that Refinement 1 builds and $\mathbb{T}_n^2$ the TRS that Refinement 2 builds. Then $\mathbb{T}_n^1 = \mathbb{T}_n^2$.*

Proof. We proceed by induction on n. The case $n = 0$ is immediate, as $\mathbb{T}_0^1 = \mathbb{T}_0^2 = \emptyset$. For the induction step, assume that $\mathbb{T}_{n-1}^1 = \mathbb{T}_{n-1}^2$ holds for some $n \in \mathbb{N}^+$. We show both directions of the set equality:

$\mathbb{T}^1_n \subseteq \mathbb{T}^2_n$: Take $\ell_1 \to r_1 \in \mathbb{T}^1_n$. If $\ell_1 \to r_1 \in \mathbb{T}^1_{n-1}$, then $\ell_1 \to r_1 \in \mathbb{T}^2_{n-1}$ and finally $\ell_1 \to r_1 \in \mathbb{T}^2_n$. Otherwise, $\ell_1 \to r_1$ was added by case (1), with $r_1 \neq \ell_1$. This implies that ℓ_1 is not reducible by $\mathbb{T}^1_{n-1} = \mathbb{T}^2_{n-1}$, and r_1 is a pre-normal form of ℓ_1. Since ℓ_1 is not reducible by $\mathbb{T}^2_{n-1}$, all of its subterms must be irreducible with respect to $\mathbb{T}^2_{n-1}$. All of the subterms of ℓ_1 are therefore normal forms with respect to $\mathbb{T}^2_{n-1}$. When those subterms were enumerated, they had to be their own pre-normal forms, and case (1) applied, but no rules were added. Therefore, ℓ_1 is a term of the form $\tau(t_1, \ldots, t_m)$ with $|\ell_1| = n$, $\tau \in \Sigma$ and each t_i is a normal form in $\mathbb{T}_{n-1}$. ℓ_1 is thus enumerated in Refinement 2. When enumerating ℓ_1, the oracle gives back the same pre-normal form of ℓ_1 that it did in Refinement 1 (by determinism of the oracle), that is r_1. By case (1) with $\ell_1 \neq r_1$, the rule $\ell_1 \to r_1$ belongs to $\mathbb{T}^2_n$.

$\mathbb{T}^2_n \subseteq \mathbb{T}^1_n$: Take $\ell_2 \to r_2 \in \mathbb{T}^2_n$. If this rule is in $\mathbb{T}^2_{n-1}$, then by the induction hypothesis it is in $\mathbb{T}^1_{n-1}$ and then in $\mathbb{T}^1_n$. Otherwise, $\ell_2 \to r_2$ was added to $\mathbb{T}^2_n$ by case (1) with $\ell_2 \neq r_2$, with $\ell_2 = \tau(t_1, \ldots, t_m)$ for $\tau \in \Sigma$, t_i normal forms with respect to $\mathbb{T}_{n-1}$, and $|\ell_2| = n$. ℓ_2 is enumerated by Refinement 1, and since N_{ℓ_2} is empty, the oracle is prompted. It returns the same pre-normal form of ℓ_2 that it did in Refinement 2 (by determinism of the oracle), that is r_2. The rule $\ell_2 \to r_2$ is then added to $\mathbb{T}^1_n$ by case (1) with $\ell_2 \neq r_2$. □

4.4 Breaking Symmetries during Enumeration (Refinement 3)

When enumerating terms with variables, many terms are equivalent up to renaming, such as $X + X + Y$ and $Z + Z + X$. We break these symmetries by enumerating only over the representatives of the equivalence classes of terms up to renaming.

Similar symmetry breaking can be done for the UCs in Σ_U. For example, a rule similar to $y + x \to x + y$ basically exists for every pair of elements $x, y \in \Sigma_U$ with $y >_\Sigma x$. Instead of enumerating them all, we factor the set of terms by equivalence up to *order-preserving* renaming. This necessitates a modification to the way we match the rules. For example, consider the rule $R : X + (b + a) \to X + (a + b)$, where $X \in V$ and $a <_\Sigma b <_\Sigma c <_\Sigma d \in \Sigma_U$. The rule matches the term $f(f(0)) + (d + b)$ with the substitution $[f(f(0))/X]$ and the (order-preserving) renaming $[b \mapsto d, a \mapsto b]$. Applying this rule to the term, we obtain the term $f(f(0)) + (b + d)$.

These two sentences changed More formally, we call a map $\rho : \Sigma_U \to \Sigma_U$ a *renaming* on $U \subseteq \Sigma_U$ if it is order-preserving, i.e. for all $a, b \in U$, $a <_\Sigma b$ implies $\rho(a) <_\Sigma \rho(b)$. With our slightly modified matching policy, a *redex* of a left-hand side ℓ in a term t is now a triple (p, σ, ρ) such that $[t]_p = \ell\sigma\rho$, where σ is a substitution and ρ a renaming of the UCs of ℓ. All other definitions of the usual term rewriting are modified accordingly. Note that this is strictly equivalent to performing the usual term rewriting on a larger term rewriting system where there is one rule per possible (ordered) tuple of UCs. For example, with this new definition, the TRS only containing the rule R has the same rewriting power as the TRS with rules (applied in the traditional sense):

$$\begin{aligned} X + (b + a) &\to X + (a + b), \quad & X + (c + a) &\to X + (a + c), \\ X + (d + a) &\to X + (a + d), \quad & X + (c + b) &\to X + (b + c), \\ X + (d + b) &\to X + (b + d), \quad & X + (d + c) &\to X + (c + d) \end{aligned}$$

We further assume that our oracle is *renaming compatible*: for any variable renaming σ, renaming ρ, and term t, $(t\sigma\rho)^* = t^*\sigma\rho$ where t^* is the pre-normal form that the oracle provides for t.

Theorem 4.6 (Subsumption). *Given some $n \in \mathbb{N}$, denote by $\mathbb{T}^2_n$ the TRS that Refinement 2 builds and $\mathbb{T}^3_n$ the TRS that Refinement 3 builds. Then for any rule $\ell \to r \in \mathbb{T}^2_n$, there exist a variable renaming σ, a renaming ρ, and a rule $\ell' \to r' \in \mathbb{T}^3_n$ such that $\ell = \ell'\sigma\rho$ and $r = r'\sigma\rho$.*

Proof. Consider $\ell \to r \in \mathbb{T}^2_n$. Let $\Sigma_U = \{a_1 < a_2 < \ldots < a_k\}$ and $U_\ell := \{a_{i_1} < a_{i_2} < \ldots < a_{i_m}\}$ be its subset of UCs appearing in ℓ. Let σ be the substitution mapping ℓ to its representative, and ρ the renaming such that $\rho(a_{i_j}) = a_j$ for all $1 \leq j \leq m$, and $\rho(a) = a$ if $a \in (\Sigma_U \setminus U_\ell)$. Then the

term $\ell' := \ell\sigma\rho$ is enumerated by Refinement 3. The query to the oracle with ℓ' returns a term r'. By the renaming compatibility of the oracle, $r' = r\sigma\rho$. As $\ell \neq r$, we have $\ell' \neq r'$; therefore, $\ell' \to r'$ is a rule in $\mathbb{T}_n^3$. Call σ^{-1} the variable renaming such that for all $X \in \mathrm{dom}(\sigma), \sigma^{-1}(\sigma(X)) = X$, and $\sigma^{-1}(X) = X$ otherwise. Similarly, call ρ^{-1} the renaming such that for all $a \in U_\ell, \rho^{-1}(\rho(a)) = a$, and $\rho^{-1}(a) = a$ otherwise. ρ^{-1} is indeed order-preserving: take $a, b \in \mathrm{dom}(\rho^{-1}) = \mathrm{codom}(\rho)$ such that $a < b$. Then there exist $c, d \in U_\ell$ such that $c = \rho^{-1}(a)$ and $d = \rho^{-1}(b)$, i.e. $\rho(c) = a$ and $\rho(d) = b$. Since ρ is order-preserving, $c < d$ as otherwise $a < b$ would be violated. Then, $\ell = \ell'\sigma^{-1}\rho^{-1}$ and $r = r'\sigma^{-1}\rho^{-1}$. □

Theorem 4.7. *Given $n \in \mathbb{N}$, the TRS $\mathbb{T}_n$ that Refinement 3 builds fulfills Property 2.*

Proof. We first prove that $\mathbb{T}_n^3 \subseteq \mathbb{T}_n^2$. Consider a rule $\ell \to r \in \mathbb{T}_n^3$. Like all terms enumerated in Refinement 3, ℓ is also enumerated in Refinement 2. By determinism of the oracle, in Refinement 2 the oracle answers r on the input ℓ. As $\ell \neq r$, the rule is added to $\mathbb{T}_n^2$.

As $\mathbb{T}_n^3 \subseteq \mathbb{T}_n^2$, any normal form with respect to $\mathbb{T}_n^2$ is a normal form with respect to $\mathbb{T}_n^3$. If a term t of size at most n is not a normal form with respect to $\mathbb{T}_n^2$, then there exist a rule $\ell \to r \in \mathbb{T}_n^2$, a subterm s of t, and a substitution μ such that $s = \ell\mu$. By subsumption (Theorem 4.6), there exist a rule $\ell' \to r' \in \mathbb{T}_n^3$, a variable renaming σ, and a renaming ρ such that $\ell = \ell'\sigma\rho$ and $r = r'\sigma\rho$. Then $s = \ell'\sigma\rho\mu = \ell'(\sigma \circ \mu)\rho$ (as ρ operates on UCs and μ on variables, which are disjoint sets). s is, with our new definitions, therefore an instance of $\ell' \to r'$, and therefore t is not a normal form with respect to $\mathbb{T}_n^3$. In conclusion, the normal forms of terms of size at most n of $\mathbb{T}_n^2$ and $\mathbb{T}_n^3$ are the same.

As $\mathbb{T}_n^2$ fulfills Property 2, any normal form of a term of size at most n is one of its pre-normal forms. As $\mathbb{T}_n^2$ and $\mathbb{T}_n^3$ have the same set of normal forms of terms of size at most n, it follows that $\mathbb{T}_n^3$ fulfills Property 2. □

4.5 Implementing the Oracle

Assume we are currently enumerating terms of size n. The oracle returns a pre-normal form t^* for some term t. This pre-normal form can be of size n or smaller. By Property 2, for each term t, there is a normal form t' in $\mathbb{T}_n$ that is a pre-normal form of t. Hence, the pre-normal form we are looking for is either a normal form of size less than n that we have already encountered when we enumerated terms of size less than n or it is one of the size-n terms we are currently enumerating.

The idea that Compass pursues to implement the oracle is to record all terms it classifies as normal forms in a set, during term enumeration. Hence, when enumerating terms of size n, we already have a set N_{n-1} of all normal forms of size less than n. To get the pre-normal form for some term t we then check if there is a semantically equivalent term in N_{n-1}. (The function getLeastEqual in Algorithm 1 implements exactly that and *deterministically* returns a $\succ$-least such term if it exists.) If that is not the case, the pre-normal form might be another term of size n that we already have or have not yet enumerated. Therefore, we table the term t for later and revisit all tabled terms after we have completed the enumeration.

Both for factoring the tabled terms and for quickly finding a semantically equivalent term in N_{n-1}, we use a hash table that uses the values of a term evaluated on a given set of input values as keys. In general, the equality test must be done by a decision procedure for equality of the theory. In our implementation, we use the Z3 SMT solver [De Moura and Bjørner 2008] to decide the equality between two terms.

Because we also enumerate terms with variables, there are usually multiple normal forms that are semantically equivalent but not $\succ$-comparable, e.g. $X + Y$, $Y + X$, $x + y$. Note that there is still only one $\succ$-least *ground* term in each set of semantically equivalent normal forms because $\mathbb{T}_n$ is ground convergent.

Algorithm 1 shows the final Compass algorithm including the refinements and the implementation of the oracle that we have discussed above.

Algorithm 1 The Compass algorithm

```
1:  procedure Compass(n: ℕ, Σ, V, ≡)
2:      if n = 1 then
3:          return (∅, {t | t/0 ∈ Σ} ∪ V)      ▷ All constants and variables are normal forms of size 1
4:      else
5:          (𝕋_{n−1}, N_{n−1}) ← Compass(n − 1, Σ, V)
6:          (𝕋_n, N_n) ← (𝕋_{n−1}, N_{n−1})
7:          C ← {}
8:          ▷ We enumerate over terms filtered by the techniques we discussed in Refinements 2
              and 3. ≡_V is syntactic term equivalence up to renaming of variables. ≡_{Σ_U} is syntactic
              term equivalence up to order-preserving renaming of UCs. ◁
9:          for t ∈ {τ(t_1, ..., t_m) | ∑_{i=1}^m |t_i| = n − 1, τ ∈ Σ, t_i ∈ N_{n−1}}/_{≡_V}/_{≡_{Σ_U}} do
10:             if rule in 𝕋_{n−1} matches the root of t then
11:                 continue
12:             if t* ← getLeastEqual(N_{n−1}, t, ≡) and t ≠ t* then
13:                 ▷ If there is a semantically equivalent normal form of size less than n, add a rule. ◁
14:                 𝕋_n ← 𝕋_n ∪ {t → t*}
15:             else
16:                 ▷ Otherwise, we table the current term for later ◁
17:                 C ← C ∪ {t}
18:         ▷ Now, we consider all tabled terms, factored by semantic equivalence. For each such
              term, there was no semantically equivalent normal form of size less than n. ◁
19:         for E ∈ C/_≡ do
20:             ▷ Put all normal forms in this class into N ◁
21:             N ← {t ∈ E | exists no t′ ∈ E such that t ≻ t′}
22:             for t ∈ E \ N do                    ▷ For all non-normal forms
23:                 t* ← getLeastEqual(N, t, ≡)     ▷ t* is such that t* ∈ N and t ≻ t*
24:                 𝕋_n ← 𝕋_n ∪ {t → t*}
25:             N_n ← N_n ∪ N
26:         return (𝕋_n, N_n)
```

4.6 The Mechanized Proof

The mechanized correctness proof proves Property 1 directly for Algorithm 1. We combine confluence, uniqueness, and completeness into one statement. The proof is *monolithic* and does not follow the refinement structure of the paper, which we find more suitable for presentation. We do not make use of determinism or renaming compatibility in the mechanized proof. The only substantial difference is that we do not implement the oracle as described in Section 4.5 but rather assume the oracle as an axiom.

4.7 A Comprehensive Example

To provide more intuition on how Compass works, we demonstrate the first rounds of Compass on the minimal boolean theory $\mathbb{B} \ni s, t := \top \mid \bot \mid \sim s \mid s \,\&\, t$ with variables A, B and UCs $a <_\Sigma b$. Each box shows one iteration n: the terms built from smaller normal forms (*enumerated*, line 9), those

that no rule matches at the root (*candidates*, line 10), and their outcome: new normal forms (green), size-reducing rules (blue), and rules that keep the size but decrease in the KBO (orange).

Size 1 — enumerated 4 · candidates 4

New irreducible terms 4

A	a	$\bot$	$\top$

At size 1, every term is a normal form (line 3): the variable A, the UC a, and the constants $\top$ and $\bot$. We only enumerate a and A, as B and b are just renamings (Refinement 3). As we know nothing about the general UC a and variable A, both are irreducible.

Size 2 — enumerated 4 · candidates 4

New irreducible terms 2

$\sim A$	$\sim a$

Size-reducing rules 2

$\sim\bot \to \top$	$\sim\top \to \bot$

At size 2, we build larger terms from the normal forms found at size 1 (line 9). This yields four negated terms: $\sim A$, $\sim a$, $\sim\top$, and $\sim\bot$. For $\sim\bot$, the oracle finds the smaller equivalent normal form $\top$ (line 12), so we add the size-reducing rule $\sim\bot \to \top$ (line 14), and likewise $\sim\top \to \bot$. $\sim A$ and $\sim a$ have no smaller equivalent: they are tabled (line 17). Each forms an equivalence class of its own (line 19) and becomes a normal form (line 21).

Size 3 — enumerated 21 · candidates 21

New irreducible terms 4

$A \,\&\, B$	$A \,\&\, a$	$a \,\&\, A$	$a \,\&\, b$

Size-reducing rules 6

$\sim\sim A \to A$	$A \,\&\, A \to A$	$A \,\&\, \bot \to \bot$	$A \,\&\, \top \to A$	$\bot \,\&\, A \to \bot$
$\top \,\&\, A \to A$				

KBO-simplifying rules 1

$b \,\&\, a \to a \,\&\, b$

Size 3 builds $\sim\sim A$, $\sim\sim a$, and 19 conjunctions of size-1 normal forms; no existing rule matches at their roots yet. The oracle finds double negation ($\sim\sim A = A$), idempotence ($A \,\&\, A = A$), identity ($A \,\&\, \top = A$), and domination ($A \,\&\, \bot = \bot$) with variables, e.g. $\sim\sim A \to A$. Their UC instances, such as $\sim\sim a$ or $a \,\&\, \top$, need no rule of their own because the variable rule $\sim\sim A \to A$ is more general.

Commutativity cannot be expressed at the variable level because the KBO cannot orient $A \,\&\, B$ and $B \,\&\, A$. For UCs, however, $b \,\&\, a \succ a \,\&\, b$: the equivalence class $\{a \,\&\, b, b \,\&\, a\}$ (line 19) yields the normal form $a \,\&\, b$ (line 21) and the rule $b \,\&\, a \to a \,\&\, b$ (line 24). $A \,\&\, a$ and $a \,\&\, A$ are both normal forms, as neither is larger than the other according to $\succ$.

The remaining 9 terms are subsumed by other rules discovered at this size (line 21): $\sim\sim a$ is subsumed by $\sim\sim A$. Similarly, $a \,\&\, \bot$, $\bot \,\&\, A$, $a \,\&\, \top$, $\top \,\&\, a$, $\bot \,\&\, \bot$, $\bot \,\&\, \top$, $\top \,\&\, \bot$, and $\top \,\&\, \top$ can be simplified by other rules at size 3.

Size 4 — enumerated 26 · candidates 18

New irreducible terms 11

$\sim(A \,\&\, B)$	$\sim(A \,\&\, a)$	$\sim(a \,\&\, A)$	$\sim(a \,\&\, b)$	$A \,\&\, \sim B$	$A \,\&\, \sim a$
$a \,\&\, \sim A$	$a \,\&\, \sim b$	$b \,\&\, \sim a$	$\sim A \,\&\, B$	$\sim a \,\&\, A$	

Size-reducing rules 2

$A \,\&\, \sim A \to \bot$	$\sim A \,\&\, A \to \bot$

KBO-simplifying rules 1

$\sim A \,\&\, a \to a \,\&\, \sim A$

Size 4 generates the negations of the size-3 normal forms and conjoins size-1 with size-2 normal forms in both orders, resulting in 26 terms. Now the root check prunes for the first time (line 10): the eight conjunctions with $\top$ or $\bot$, such as $\sim A \,\&\, \bot$, are instances of size-3 rules, which leaves 18 candidates. The oracle finds the complement rules $A \,\&\, \sim A \to \bot$ and $\sim A \,\&\, A \to \bot$. The pair $\sim A \,\&\, a$ and $a \,\&\, \sim A$ can be oriented by the KBO, resulting in $\sim A \,\&\, a \to a \,\&\, \sim A$. $\sim A$ has size at least 2 and is therefore larger than any UC. Of the remaining 15 candidates, four are instances of the new rules; the other 11 become normal forms.

The algorithm continues until size 61, resulting in 586,388 size-reducing rules, 1173 KBO-simplifying rules, and 4445 normal forms. The largest normal forms are found at size 30. This minimal boolean theory takes more iterations to saturate than our other boolean theory with | (disjunction) and ^ (xor), as normal forms are expressed more verbosely.

5 Complexity

In this section, we quantitatively investigate the complexity of the TRSs that Compass generates. We start with the result that Compass terminates for theories with finite universes and therefore produces a TRS of finite size.

Lemma 5.1. *Every $\succ$-antichain of a size-KBO over a finite Σ and a finite $V = \{X_1, \ldots, X_p\}$ is finite.*

Proof. Consider a $\succ$-antichain T and assume it is infinite. Since only finitely many terms exist per size, choose an infinite subsequence $t_1, t_2, \ldots$ with strictly increasing sizes. Map each t_i to its variable-count vector $(|t_i|_{X_1}, \ldots, |t_i|_{X_p}) \in \mathbb{N}^p$. By Dickson's lemma [Dickson 1913] there exist $i < j$ such that the variable-count vector of t_i is (component-wise) less than or equal to that of t_j. Then, t_j satisfies both KBO conditions over t_i (see Section 2): the variable condition, and $w(t_j) = |t_j| > |t_i| = w(t_i)$, so $t_j \succ t_i$ which contradicts the antichain assumption. □

Theorem 5.2. *If the universe A of the theory is finite, there is a k such that* Compass$(k', \ldots) =$ Compass$(k, \ldots)$ *for each $k' \geq k$.*

Proof. Compass does not add new rules or normal forms if the set that it loops over in line 9 is empty. This is the case when there are no more normal forms of sufficient size in N_{n-1} such that terms of size n can be constructed. Assume that the number of normal forms Compass adds is finite and let m be the iteration of Compass in which the last normal form is added, i.e. $N_{m'} = N_m$ for each $m' \geq m$. If the signature's highest arity is q, the set of line 9 will be empty for $n > q \cdot m + 1$ at the latest because we simply cannot construct terms of size n from normal forms anymore.

Now, each normal form we add corresponds to a function in $A^{|V \cup \Sigma_U|} \to A$. If A is finite, the number of functions in this set is also finite. If our rule set has variables, we might have multiple semantically equivalent normal forms that are not totally ordered by $\succ$. But all semantically equivalent normal forms constitute a $\succ$-antichain, because getLeastEqual will always deterministically return a least semantically equivalent term. Therefore, by Lemma 5.1, we can add only finitely many normal forms per function. Hence, Compass adds only finitely many normal forms if A is finite. □

Note that as a consequence we get a finite TRS that is convergent and complete for *all* ground terms over Σ, if the universe of the theory is finite. We experimentally confirm this by letting Compass derive a ground convergent TRS for terms over the theory of booleans with up to three UCs in the next subsection.

5.1 Quantitative Investigation

To gauge the effectiveness of the refinements that we presented in Section 4, let us get an understanding of the total number of terms that there are for a given n. Given c_0 nullary, c_1 unary, and c_2

binary symbols in our signature, the total number $T(n)$ of terms of size n is given by the following recursive formula:

$$T(n) = c_1 \cdot T(n-1) + c_2 \cdot \sum_{i=1}^{n-2} (T(i) \cdot T(n-1-i)) \qquad T(1) = c_0$$

which closely resembles the Catalan[1] numbers. Using basic tools from analytic combinatorics [see Flajolet and Sedgewick 2009], we derive a function to describe the number of terms asymptotically:

$$T(n) \in O\left(\frac{\left(c_1 + 2\sqrt{c_0 c_2}\right)^n}{\sqrt{n^3}}\right)$$

The number of terms grows exponentially, and the base of the exponential is given by the number of function symbols in each category.

Let us consider the boolean signature $\mathbb{B}$ of Fig. 2a with three UCs and no variables. According to the formula above, the number of terms grows as $\widetilde{O}(7^n)$. Hence, for term sizes beyond 10, a complete enumeration of all terms is not efficient anymore. Yet, Compass generates a TRS that is complete and convergent for *all* ground terms over $\mathbb{B}$.

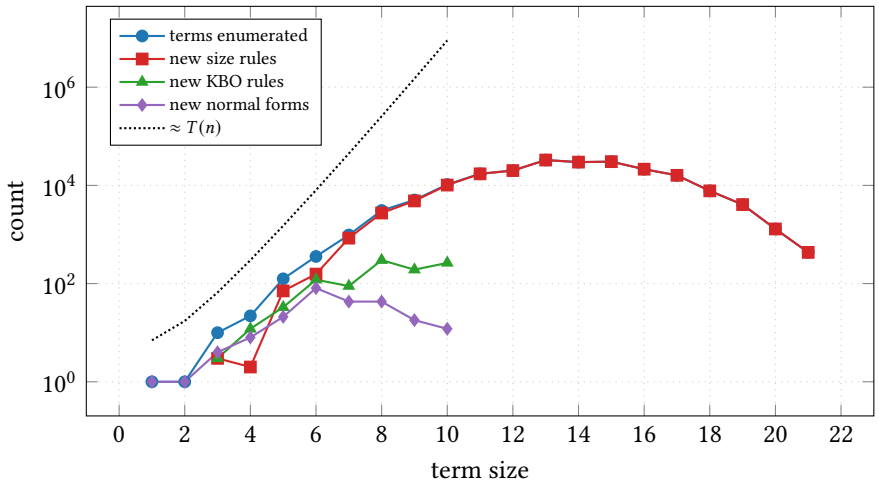
Fig. 1 shows the number of terms enumerated and rules generated. The dotted line shows, as a reference, the number of terms according to the above asymptotic approximation. The horizontal axis indicates the term size or, equivalently, the iteration of Compass. The curve "terms enumerated" gives the number of terms enumerated in each iteration of Compass. The curve "new KBO rules" shows the number of inserted rules in which the right-hand side has the same size as the left-hand side. The curve "new size rules" shows the number of added rules in which the size of the right-hand side is strictly smaller than the size of the left-hand side. The two different rule sets correspond to the two different locations in Algorithm 1 in which rules are added. The curve "new normal forms" shows the number of normal forms added in the respective iteration. In total, there are 232 normal forms. Note that there are $2^{2^3} = 256$ different boolean functions with three variables. The difference of $256 - 232 = 24$ comes from symmetry breaking, e.g. the three functions x, y, z appear as only one normal form because they are equivalent modulo renaming.

One can see that the largest normal form has size 10 and, accordingly, we enumerate terms up to size 21 which exactly corresponds to a term with a binary function symbol at its root and two size-10 subtrees. According to the results we have established in Section 4, there cannot be any rule with a larger left-hand side. After the iteration in which we find the last normal form, we also do not generate any "KBO rules" anymore. That is consistent with our theoretical results as well because the right-hand side of each rewrite rule is always a normal form in any $\mathbb{T}_{\geq n}$ where n is the iteration in which the rule was added. Hence, if the last normal form we add has size k, all rules we add in an iteration $> k$ must have right-hand sides of sizes $\leq k$ and therefore are "size rules".

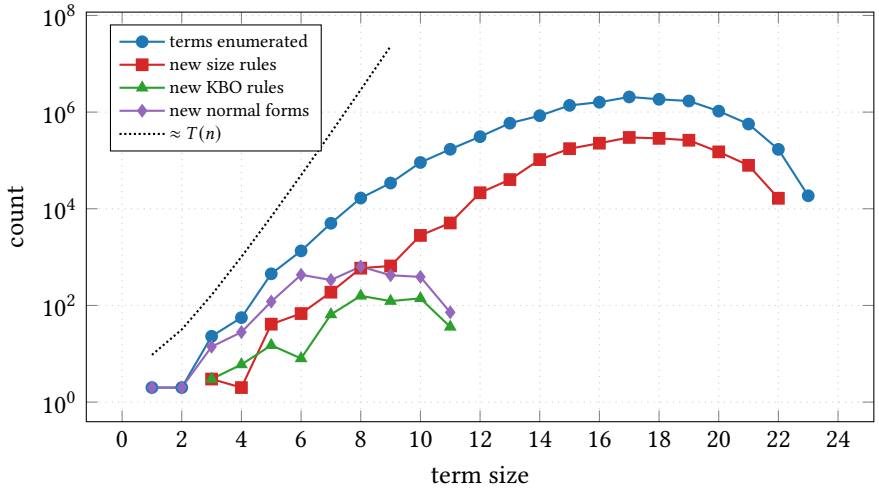
The resulting term rewriting system has $|\mathbb{T}_{20}| = 200{,}953$ rules of which 1013 are "KBO rules" and the rest are "size rules". Our algorithm enumerates $201{,}185 \approx 2 \times 10^5$ terms in total. According to the approximate total number of terms $T(n)$, there are roughly $7^{21} \approx 3 \times 10^{17}$ terms of size ≤ 21. Hence, Compass enumerates only a fraction of $\approx 10^{-12}$ of all possible terms and therefore avoids enumerating a significant portion of terms. None of the enumerated terms is skipped in line 10: a UC-only rule matches at the root only terms of its own size, and $\mathbb{T}_{n-1}$ contains only smaller rules. Hence, each enumerated term becomes either a rule or a normal form ($200{,}953 + 232 = 201{,}185$).

If we add (three) variables, the graph looks very similar. However, in the beginning, we generate fewer rules because left-hand sides with variables subsume UC-only rules. As the term size gets larger, the number of terms that have to be enumerated is one to two orders of magnitude larger so

[1] OEIS A000108



(a) Boolean theory with three uninterpreted constants.



(b) Boolean theory with three uninterpreted constants and three variables.

Fig. 1. Terminating runs of Compass.

that we eventually also add more rules (1,670,537 in total). Here, rules with variables also match at the root of larger terms: 61.5% of the 12,433,926 enumerated terms are skipped in line 10. This increase comes from rules in which variables and uninterpreted constants are mixed. The number of normal forms is also higher because there are now different semantically equivalent but unorderable normal forms such as $x \,\&\, X$ and $X \,\&\, x$. Since terms with variables that cannot be oriented by the KBO remain normal forms (Section 3), the largest normal form has size 11, so the system is complete after size 23 with 2452 normal forms.

6 Experimental Evaluation

We structure our evaluation around six research questions:

RQ1 How does Compass compare to Ruler in terms of rule coverage and synthesis time with respect to rule size?

RQ2 How do UC-only rewrite systems compare to rewrite systems with variables in terms of simplification power and TRS synthesis time?

RQ3 How does greedy rewriting compare to e-graph rewriting for term simplification when comparing Compass's against Ruler's TRS?

RQ4 How do TRSs produced by unfailing Knuth-Bendix solvers compare against Compass's with respect to synthesis time and simplification power?

RQ5 How do greedy rewriting and e-graph rewriting scale with larger terms and rule size in terms of simplification power and time and memory consumption?

RQ6 How do these results transfer to other theories?

In our experiments, we use the theories from Nandi et al. [2021] and an additional integer theory. Fig. 2 shows the signatures of all theories we experiment on. We will discuss RQ1–RQ5 on the boolean theory. On this theory, Compass terminates and produces a convergent and complete rewrite system for all terms which allows for an in-depth discussion of Compass's properties. RQ6 then discusses how well Compass's simplification power and scaling transfer to other theories.

In our experiments, we use a simple greedy rewriter. Our greedy rewriter initially fixes an arbitrary order of rules and always applies the first rule that matches. To make rule matching efficient, we use a discrimination tree [Sekar et al. 2001], a trie-like data structure. With a discrimination tree, the rule matching time complexity depends only on the size of the largest rule and the size of the signature but *not* on the number of rules. The greedy rewriter implements the augmented UC-matching discussed in Section 4.4.

All experiments were run on an AMD Ryzen 7 3700X (Zen 2, x86_64) with 8 cores at 3.6 GHz and 64 GB of RAM.

6.1 Synthesis Coverage and Scaling (RQ1)

We want to assess how strong the rewrite systems of Compass and Ruler are in terms of proving power. That is, given two TRSs $\mathbb{T}$ and $\mathbb{T}'$, we count how many rules of $\mathbb{T}'$ one can prove by reducing the left-hand side and the right-hand side of the rule to the same term using the rules in $\mathbb{T}$. The original Ruler paper [Nandi et al. 2021] used this experiment to assess Ruler's derivability. We replicate the experimental setup of Nandi et al. [2021] and use egg [Willsey et al. 2021] with 5 iterations of equality saturation to check if the left- and right-hand sides of a rule are equal under the respective rewrite system.

We compare synthesis time and the rule count of the rewrite systems Compass synthesizes against Ruler's on the boolean theory in Table 1. Ruler synthesizes by term tree depth, while Compass enumerates by number of term tree nodes. To make the results comparable, we determine

$$\mathbb{B} \ni s, t ::= \sim t \mid s \,\&\, t \mid s \mid t \mid s \,\hat{}\, t$$

(a) Boolean theory with Not, And, Or, and Xor.

$$\mathbb{I} \ni s, t ::= -t \mid s + t \mid s - t \mid s \times t$$

(b) Integer theory with Negation, Plus, Minus, and Multiplication.

$$\mathrm{BV} \ni s, t ::= \sim t \mid -t \mid s + t \mid s - t \mid s \times t \mid s \ll t \mid s \gg t \mid s \,\&\, t \mid s \mid t$$

(c) Bitvector theory with Not, Negation, Plus, Minus, Multiplication, Shift Left, Shift Right, And, and Or.

Fig. 2. Theories for our terms.

the size n of the largest left- or right-hand side of all rules Ruler generates and synthesize the corresponding size-n system with Compass.

We derive Ruler's rules with three variables and our rules with three UCs and no variables. Since the e-graph does not require a KBO, we treat each uninterpreted constant in our rules as a variable.

	Ruler			Compass			
Depth	# Rules	Time	Size	# Rules	Time	Ruler → Compass	Compass → Ruler
2	18	0.02 s	5	124	<0.01 s	100.00%	100.00%
3	27	0.08 s	7	1332	<0.01 s	99.85%	100.00%
4	33	25 s	9	9385	0.11 s	98.06%	100.00%
5	33	132 s	9	9385	0.13 s	98.06%	100.00%
6	35	740 s	12	57,022	0.74 s	—	—
—	—	—	21	200,953	3.04 s	—	—

Table 1. Coverage of Ruler's and our rewrite systems

Compass covers all of Ruler's rules at every depth. Ruler covers all of our size-5 rules, misses 2 of our 1332 size-7 rules (99.85%), and misses 182 of our 9385 size-9 rules (98.06%). From size 12 on, Ruler's coverage check cannot handle our number of rules.

Synthesis also scales better with Compass in both time and memory. Small systems up to size 7 are essentially instantaneous for both tools. Ruler's depth-6 system requires 740 s for 35 rules, whereas Compass generates 57,022 rules at size 12 in 0.7 s. Compass derives the complete ground-convergent boolean system of 200,953 rules at size 21 in 3.0 s. Note that this system has the full proving power: it can decide equivalence of boolean formulas with up to three variables.

6.2 Adding Variables (RQ2)

Variable rules are desirable because they can match entire subterms. UC-only rewrite systems, however, only match ground terms. Hence, a single variable rule subsumes many ground rules, e.g. $A \,\&\, A \longrightarrow A$ subsumes every instance, such as $(a \,\hat{}\, b) \,\&\, (a \,\hat{}\, b) \longrightarrow a \,\hat{}\, b$. We measure the effect of including variables in our TRS on synthesis time and memory, and simplification power. We run Compass once with three UCs alone and once with three variables added.

	UC-only		w/ Variables	
Size	#Rules	Time	#Rules	Time
3	6	0.0 s	6	0.0 s
4	20	0.0 s	14	0.0 s
5	124	0.0 s	70	0.0 s
6	400	0.0 s	146	0.0 s
7	1332	0.0 s	399	0.1 s
8	4368	0.0 s	1147	0.2 s
9	9385	0.1 s	1926	0.4 s
10	19,795	0.2 s	4874	1.1 s
11	36,988	0.4 s	9992	2.4 s
12	57,022	0.7 s	31,417	6.9 s

	UC-only		w/ Variables	
Size	#Rules	Time	#Rules	Time
13	89,827	1.1 s	71,547	16.6 s
14	119,599	1.6 s	175,599	40.8 s
15	150,040	2.1 s	350,487	1.5 min
16	171,394	2.5 s	577,677	2.9 min
17	187,417	2.7 s	876,605	5.1 min
18	195,157	2.9 s	1,162,845	8.2 min
19	199,225	3.0 s	1,424,581	11.7 min
20	200,521	3.0 s	1,574,849	14.8 min
all	200,953	3.0 s	1,670,537	17.3 min

Table 2. Number of rules and cumulative synthesis time of Compass on the boolean theory, for the UC-only and the variable setting. Sizes 3–12 (left) and 13–20 (right); "all" denotes the complete system, which is reached at size 21 without and at size 23 with variables.

Table 2 reports the number of rules Compass finds together with the synthesis time for the variable and the UC-only setting. With variables, we enumerate considerably more terms, 62× more in total in the boolean theory. For smaller term sizes, we create fewer rules compared to the UC-only system because one variable rule can subsume multiple UC-only rules. At some point, however,

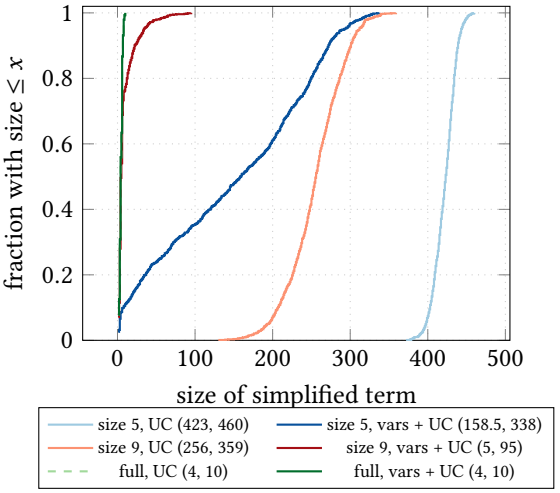


Fig. 3. Comparison of greedy simplification on 1000 terms of size 500 for rewrite systems of different sizes. Median simplified term size and maximal simplified term size are given in parentheses.

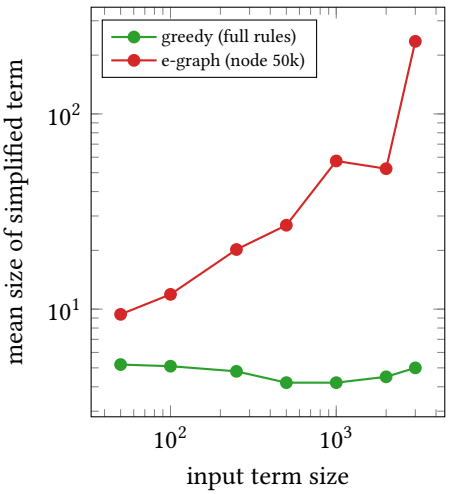


Fig. 4. Comparison of greedy and e-graph rewriting on large terms.

this shifts because the number of terms with variables that have to be enumerated is significantly larger than the number of UC-only terms. Therefore, synthesis takes longer, for example 17.3 min (w/ variables) versus 3.0 s (w/o variables) for the full boolean system.

The system with variables also has more normal forms: In a system with variables, there can be multiple semantically equivalent normal forms (e.g. $x \,\&\, X$ and $X \,\&\, x$), which is not possible in a UC-only system.

We compare the simplification power of UC-only systems and systems with variables by greedily rewriting 1000 random terms of size 500 using a small (size 5), a medium (size 9), and the complete rewrite system. The random terms use up to three random constants and the operations from the boolean signature. Figure 3 shows a cumulative plot of the term sizes after simplification.

Without variables, a larger TRS clearly has better simplification power. The median size of the optimized terms goes from 423 at size 5 to 256 at size 9. For a given n, the size of the terms does not vary a lot (the left and right ends of the curve are close together).

TRSs with variables have significantly better simplification power. The size-5 system with variables (median 158.5) already surpasses the size-9 system without variables (median 256). The size-9 system with variables reaches median 5 and reduces about 70% of the terms to near-minimal form, though with greater variance across terms.

The complete system reaches the minimal form for every term (median 4, maximum 10); with and without variables, the two curves coincide. The simplification time remains essentially independent of the rule-set size. Adding variables is therefore advantageous for simplification, especially with small rule sets, at the cost of higher synthesis time.

6.3 Rewriting Strategies (RQ3)

Greedy rewriting is much less expensive than e-graph saturation because it destructively updates the subject term. However, if the TRS is not confluent, it is prone to getting stuck in a local minimum. E-graph saturation does not have that problem because no intermediate term is ever

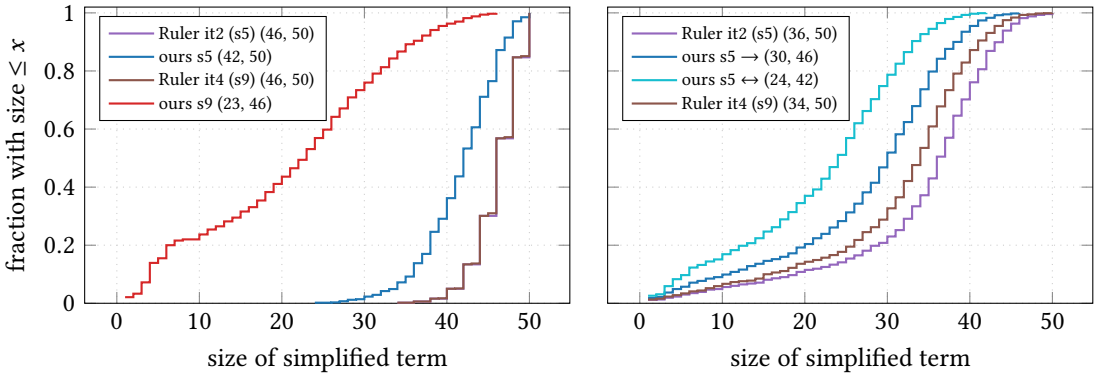


Fig. 5. Comparison with Ruler on greedy (left) and e-graph (right) simplification of 1000 terms of size 50. Median simplified term size and maximum term size are given in parentheses.

deleted. However, e-graph saturation is prone to hitting a memory wall because of just that. Ruler's goal is to generate a small rule set to avoid e-graph explosion, whereas Compass's goal is to provide a large rule set for greedy rewriting. In this section, we compare both rule synthesis techniques in both rewriting settings: greedy rewriting and e-graph saturation.

Greedy rewriting. Figure 5 compares greedy rewriting on 1000 terms of size 50. Our uninterpreted-constant system at size 5 (median 42) outperforms both Ruler systems (median 46). Our size-9 system is considerably stronger (median 23), and its distribution resembles that of variable rules on larger terms. Ruler's two systems basically give the same results. Their steep curves show that only a few rules apply, each removing a large portion of the term.

Equality saturation. We build one e-graph per term and saturate for up to two iterations with egglog [Zhang et al. 2023]; more iterations exhaust memory because of the associativity and commutativity rules. We also apply our rules bidirectionally, orienting every admissible rule both ways. Figure 5 shows that our rules improve over greedy rewriting (size-5 median 42 $\longrightarrow$ 30). Our size-5 system is stronger than both Ruler systems (median 36 and 34), and the bidirectional version improves the result further (median 24). The improvement from bidirectional orientation is consistent with our system being confluent only up to size 5. It is noteworthy that Compass's up-to-n rule set performs better in an e-graph than with greedy rewriting. We attribute this to two reasons: first, the e-graph does not forget intermediate terms and can therefore compensate for some of the "non-confluence" of the up-to-n systems. Second, because e-graphs put all terms that originate from a term with rewriting into the same e-class, one basically gets the rewrites among all of these terms for free, which could compensate for rules that are missing in the up-to-n system. This way, the e-graph provides a relevant critical pair.

We omit the size-9 system, since egglog cannot handle that many rules.

Summary. In this experiment we used relatively small terms of size 50 because larger terms regularly lead to memory exhaustion during e-graph saturation. On these small terms, the e-graph gives better simplification power than greedy rewriting. Our size-5 system surpasses both Ruler systems, which would require a deeper and more expensive saturation to match simplification performance, which the memory explosion prohibits. Greedy rewriting is directed, matches rules in near-constant time and applies to terms of any size. Choosing a larger rule set (here size-9 is sufficient) and using greedy rewriting matches or outperforms e-graph simplification.

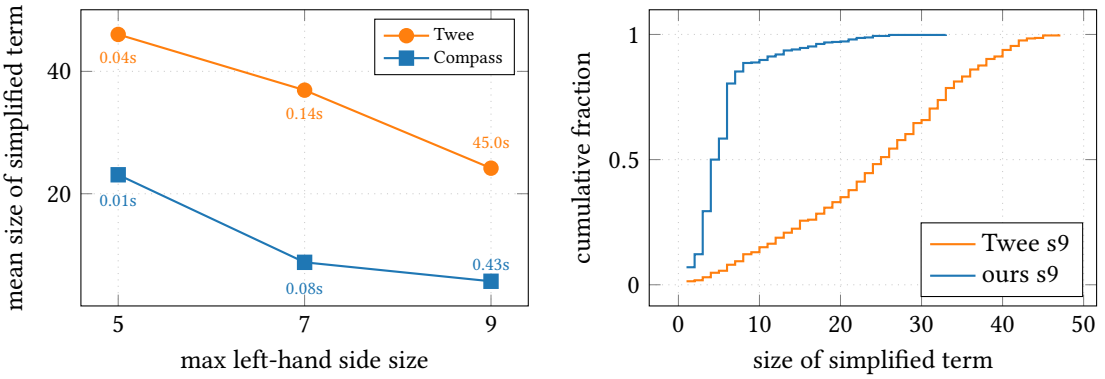


Fig. 6. Simplification power of Compass compared to Twee on 500 random terms of size 50 with differently sized rule systems. The synthesis time of the rule set is shown next to the nodes.

6.4 Comparison against Unfailing KBC (RQ4)

Beyond Ruler, we compare against Twee [Smallbone 2021], which synthesizes a ground-convergent rewrite system by unfailing Knuth-Bendix completion based on an axiomatization of the boolean theory we use in our experiments. We generate rules with Twee and, as for our own rules, apply them greedily to 500 random terms of size 50 (Fig. 6). Compass both synthesizes faster and simplifies more: at size 9, its rules reduce the terms to a mean size of 5.7 against Twee's 24.2, while Compass's synthesis takes 0.43 s against Twee's 45 s.

6.5 Scaling to Larger Terms and Rule Sets (RQ5)

Practical workloads involve large terms and large rule sets. We test both rewriting strategies, greedy rewriting and e-graph saturation, in this regime.

Large terms. We measure how each strategy scales as the term size grows. We consider terms of sizes 50, 100, 250, 500, 1000, 2000, 3000. For each size, we randomly generate 50 different terms and simplify each term with both strategies. Greedy rewriting uses Compass's full system, and the e-graph uses Ruler's depth-2 system. We limit the e-graph to 50,000 nodes and run until reaching the node or memory limit. Figure 4 shows the results.

Greedy rewriting reduces the term to its minimal form for every size. The e-graph returns results at every size but stays well above the optimum. The gap widens the larger the term is.

Greedy rewriting is also considerably less time-consuming: 50 terms of size 1000 take 0.12 s with greedy rewriting versus 42 s with the e-graph. Two effects degrade the e-graph on large terms: the associativity and commutativity rules enlarge it rapidly, and its undirected search reduces simplification quality.

Rule-set size. Our complete systems contain hundreds of thousands of rules. For greedy rewriting, however, the number of rules is irrelevant because of the discrimination tree. Even our largest systems apply without difficulty to very large terms. Egglog and Ruler, in contrast, cannot handle that many rules. An e-graph approach needs a small rule set, which limits simplification power.

Different rule sizes. Synthesis cost grows steeply with the maximal rule size, so we determine how large rules must be for effective simplification. We evaluate systems with rules up to size 14 and plot the median normal-form size against the rule size on random terms. We compare greedy rewriting

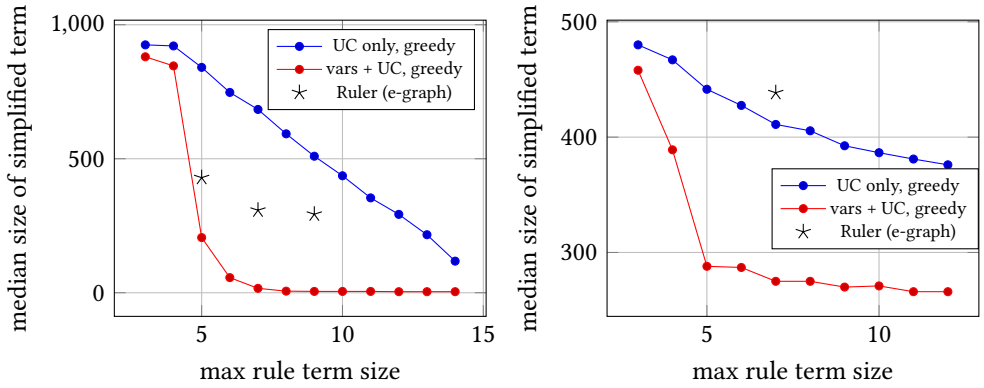


Fig. 7. Comparison of UC-only, variable, and e-graph rewrite systems on the boolean (left) and integer (right) theories.

with Compass's UC-only systems, greedy rewriting with Compass's systems with variables, and Ruler in egglog. Figure 7 shows the curves.

Uninterpreted-constant systems improve approximately linearly with rule size. Systems with variables improve considerably faster and converge near size 7, with size 8 already close to optimal.

Ruler surpasses the uninterpreted-constant system of the same size but is considerably slower and more memory-intensive. Ruler lies between the two greedy variants and stagnates near size 7. A small variable system therefore suffices to optimize terms well.

6.6 Other Theories (RQ6)

The boolean theory is finite and synthesis completes within a reasonable time frame. In this section, we check whether the findings generalize to larger and infinite theories by repeating the experiment of RQ5 with the integer, BV4, and BV32 theories. Figures 7 and 8 show the boolean, integer, and bitvector results.

For the greedy systems, the trends carry over to all theories: variable systems decrease the size of the simplified term quickly, and uninterpreted-constant systems decrease it steadily. Ruler's e-graph lies between the two greedy variants on the bitvector theories, but on the integer theory it is weaker than both. Unlike the boolean theory, the bitvector and integer theories are not exhausted. We let each theory run for a budget of 10 min. Their normal-form sizes therefore continue to decrease with rule size instead of reaching a fixed optimum.

6.7 Restrictions and Threats to Validity

Even if the number of terms Compass has to enumerate is finite for finite theories, the set of terms is still too vast to be enumerated completely for practically relevant theories. Therefore, for practical scenarios, Compass cannot guarantee convergence and completeness for *all* ground terms. However, our evaluation shows that even up-to-size-n systems are helpful for term simplification.

One could argue that performing benchmarks on randomly generated terms lacks practical relevance. However, it is difficult to derive a theoretical characterization of terms that occur "in practice", and existing approaches, such as Ruler [Nandi et al. 2021] or Twee [Smallbone 2021], also do not make such assumptions. Hence, if any of the compared approaches performed better or worse on "practically relevant" terms, it would be either because of some property that has not been

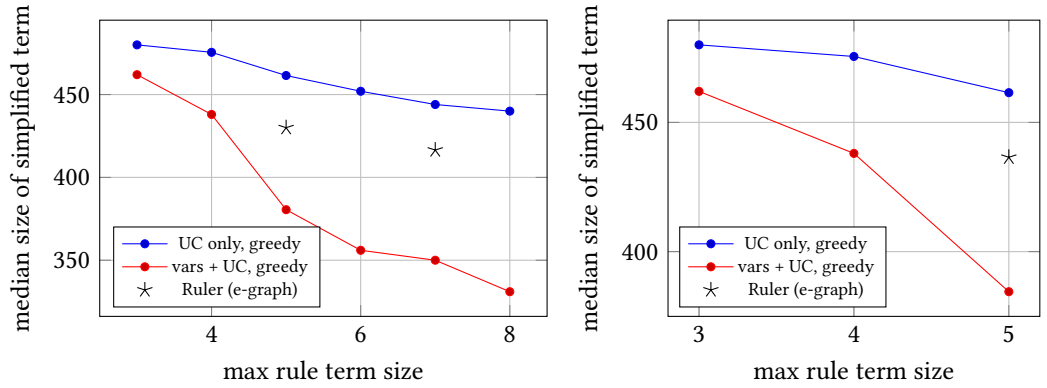


Fig. 8. Comparison of UC-only, variable, and e-graph rewrite systems on the 4-bit (left) and 32-bit (right) bitvector theories.

explored yet in the literature or because of mere coincidence. Investigating properties of terms that occur in practice is a potentially interesting direction of research but not the focus of this paper.

Our greedy rewriter can apply rewrite rule sets that have been generated for k UCs to subject terms with more than k UCs because of the renaming technique introduced in Section 4.4. However, a significant restriction of Compass is that the guarantees hold only for the number of UCs in the signature. For subject terms that require more than three UCs, the rewrite systems Compass generates with three UCs do not have any guarantees, not even for ground terms up to size n. Future work could investigate how the simplification power of Compass's three-UC rule sets scales with more UCs in the subject term.

7 Related Work

Ruler. Ruler [Nandi et al. 2021] uses equality saturation to *infer* rewrite rules. E-graphs compactly represent all candidate pairs of equivalent terms. Equality saturation collapses equivalence classes, yielding smaller and more general rule sets.

Like Ruler, we enumerate terms to generate our rules. However, Ruler and Compass have different goals: Ruler synthesizes small rule sets to drive equality saturation, while we synthesize large ground complete and convergent rule sets for greedy rewriting. Furthermore, Compass terminates on all finite-universe theories, a guarantee Ruler does not provide.

Enumo. Enumo [Pal et al. 2023] addresses the scalability limitations of Ruler by introducing a DSL for programmable theory exploration. Terms are enumerated using *workloads*, which are sets of s-expressions. Enumo provides operators to strategically compose workloads. By expressing targeted subsets of the term space, Enumo scales to grammars and term depths that an exhaustive enumeration cannot reach. While Enumo is crucially dependent on user-guided strategies to explore the term space, Compass automatically enforces the completeness of the resulting TRS. As with Ruler, Enumo's resulting rule sets are much smaller, in order to be used in an equality saturation engine rather than in a greedy rewriting tool. Finally, Enumo introduces a strategy to select desirable rules in order to *fast-forward* equality saturation. Enumo's fast-forwarding strategy provides support for theories where equality is undecidable. Compass must be used on a theory with decidable equality: extending this work to handle theories with undecidable equality is an interesting direction for future work.

Halide. Newcomb et al. [2020] verify and extend the large hand-written TRS inside the Halide compiler [Ragan-Kelley et al. 2013]. They certify existing rules and guarantee termination via formal verification, then use program synthesis to derive new provably correct rules from cases where the compiler fails to prove properties.

Both works make calls to an SMT solver in order to produce sound rules by construction: the SMT solver verifies that both sides of a rule are semantically equal. Like the term rewriting systems generated by Compass, Halide's term rewriting system is terminating. Their proof of termination uses a 13-component lexicographic ordering, while we use a size-KBO. In both Compass and the Halide work [Newcomb et al. 2020], candidate left-hand sides are iteratively generated from normal forms of the current TRS.

Their approach is able to handle conditional rewrite rules by first generating a sound rule on actual constants, and then trying to generalize the rule on more constants. A guard condition is synthesized iteratively by finding counterexamples. When no counterexample is found by Z3, the conditional rule is sound. However, they do not give any guarantee on confluence or completeness of the TRS. Maintaining our theoretical guarantees with support for conditional rules is important future work. Compass also assumes decidable equivalence, whereas the Halide work is specifically engineered for undecidable integer arithmetic.

(Unfailing) Knuth-Bendix completion. The KBC procedure [Knuth and Bendix 1970] transforms a set of equations into a convergent TRS by orienting equations under a reduction order and resolving critical pairs, failing when an equation is unorientable. Bachmair et al. [1989] removed this restriction: unorientable equations are retained and only their orientable ground instances are applied as rewrite rules when trying to prove an equality. This yields a TRS that is ground convergent with respect to the chosen ordering.

Unfailing KBC shares ground completeness and KBO-compatibility of rules as goals with Compass, but the approaches differ in their inputs and their termination behavior. KBC starts from a user-supplied equational axiomatization (that is, a set of equalities) and does not terminate for several theories (e.g. Peano arithmetic). KBC is a semi-decision procedure for the word problem and will not terminate on theories with an undecidable word problem. Compass requires no axiomatization; instead, it depends on a solver for the word problem of the given theory, and is guaranteed to terminate with a *finite* ground-complete TRS for all finite-universe theories. If the theory does not have a finite universe, then Compass will not find every possible rule. However, given a target size, Compass ensures ground convergence and completeness up to that size.

There exist theories for which a finite axiomatization does not exist, but for which the word problem is decidable (for example, the theory of positive numbers with exponentiation [Gurevič 1985]). In that case, KBC is not applicable, while Compass is.

Conversely, many theories for which the word problem is undecidable admit an axiomatization, like Peano arithmetic. Our tool cannot be complete in that case, as no full decision procedure for the word problem exists. However, automated theorem provers based on unfailing KBC (such as Twee [Smallbone 2021]) add the negated goal as an equation, and try to find a refutation. As unfailing completion is refutationally complete, this gives a semi-decision procedure for the word problem. This is not a result that we can achieve with our method.

Modern reformulations of KBC drop the requirement of fixing the reduction order in advance, with techniques like maximal completion [Klein and Hirokawa 2011; Winkler 2019], coupling with a termination checker [Wehrman et al. 2006], or multi-completion [Winkler et al. 2010]. In comparison, Compass commits to a fixed size-KBO because we want normal forms to be the smallest semantically equivalent terms.

CVC4. Nötzli et al. [2019] use syntax-guided synthesis (SyGuS) to enumerate and propose new rewrite rules for the SMT solver's simplifier (here, CVC4 [Barrett et al. 2011]). The process is semi-automated: their approach proposes a set of new rewrite rules from a base set. A human can then select some of the proposed rules to be added to the set. This process is repeated several times. In order to filter out candidate pairs of terms, they check three properties: (1) If the candidate rule is an instantiation of an existing rule, then it is not added. This is similar to Refinement 1, where a left-hand side of a rule must be a normal form of $\mathbb{T}_{n-1}$. (2) As in Compass, rewrite rules with variables are only enumerated up to a canonical renaming. (3) Like Ruler [Nandi et al. 2021] and Enumo [Pal et al. 2023], they use an e-graph to filter out rewrite rules that are implied by other rules up to congruence closure. In contrast to Compass, their objective is not to obtain a convergent or complete term rewriting system.

Theory exploration. Theory exploration [Buchberger et al. 2006] deals with the automatic discovery of lemmas in equational theories. QuickSpec [Claessen et al. 2010; Smallbone et al. 2017] enumerates terms over a given signature and partitions them into equivalence classes by random testing. All equations of the form $t \approx r$ where r is the representative of the class of t are candidate equations. The set of discovered equations is grown incrementally, and congruence closure on the previously discovered equations filters potential candidates. Compass enumerates and partitions terms into equivalence classes as well. However, our approach differs in three aspects: instead of random testing, we use an equivalence oracle; we do not have rule candidates (none of our rules is redundant); and we ensure ground convergence and ground completeness. QuickSpec enumerates terms in order of increasing depth. Terms that have non-canonical subterms are not enumerated: this is very similar to our Refinement 2.

Speculate [Braquehais and Runciman 2017] extends theory exploration to *conditional* equations and inequalities about black-box functions. Expressions are separated into equivalence classes by testing, and term rewriting is used to discard redundant equations. It supports conditional equations, which Compass does not. Similar to QuickSpec, however, the equations are conjectured from tests rather than proved, and no attempt is made to produce a convergent or complete system.

8 Future Work and Open Questions

Compass synthesizes *complete* rule sets, which restricts it to theories with decidable equality and to sizes within the enumeration budget. Approximate synthesis would trade these guarantees for scaling. Under-approximation, which tests on random inputs only and skips expensive equivalence checks, lowers the synthesis cost. Over-approximation, which admits extra candidate rules and filters them afterwards, enlarges the rule set and opens it to heuristic search. The same relaxation would let Compass target theories with undecidable or expensive equality. At the variable level, one could conflate equivalent normal forms into a single representative to minimize enumeration at the cost of overlooking some rules.

Associativity and commutativity enlarge the rule set and force Compass to retain UCs for symmetric terms. Matching modulo associativity and commutativity would make most rules redundant, allowing for fewer normal forms and a smaller rule set. An alternative is to enumerate only variable terms, and to enumerate the UC instances of a variable term only if that term cannot be reduced.

With variables, the test in line 10 of Algorithm 1 is true for 61.5% of the enumerated terms in our experiments. It would be interesting to investigate efficient methods that filter out more of these terms before enumeration.

Another open question concerns the minimality of the rule set. Are there smaller ground complete rule sets than Compass's for UC-only rules? We do not believe that Compass's rule sets are minimal for rules with variables. What could smaller systems look like?

Synthesis for rules with variables and UC-only rules could potentially be decoupled. Variable rules are cheap and general at small sizes, whereas UC rules scale to large rule sets. This suggests synthesizing with variables first and UCs later, then combining the results.

Finally, it would be of practical interest to also support the synthesis of conditional rewrite rules.

9 Conclusions

We presented Compass, an algorithm that synthesizes rewrite rule sets that are convergent and complete for ground terms up to a given size. With such a rule set, every ground term of size at most n reaches, under any rule application order, the smallest term that is semantically equivalent to it. We proved both convergence and completeness and mechanized the proof in Lean. Compass enumerates rewrite rules of increasing size and uses several techniques to significantly prune the enumeration space. For finite theories, Compass terminates with a rule set that covers ground terms of all sizes. We demonstrated this by generating a rule set that is convergent and complete for boolean terms *of any size* with up to three uninterpreted constants.

Our experiments show that these guarantees are useful in practice. With a rule set that is complete up to a sufficient size, greedy rewriting simplified terms further than equality saturation at a small fraction of the runtime and memory consumption. Compass's rule sets cover all of Ruler's rules, and Compass outperforms unfailing Knuth-Bendix completion, as implemented in Twee, in both synthesis time and simplification power. We conclude that convergent and complete rule sets are obtainable automatically at practically relevant sizes, and that simple greedy rewriting can be a feasible alternative to equality saturation for term simplification.

10 Disclosure of Use of AI in Research

Research. After finalizing the paper, we asked an AI agent to perform a critical review of the paper. This made two critical suggestions which we implemented: First, we added a comparison to Twee to compare Compass against KBC solvers. Second, our initial proof of Theorem 5.2 (the only proof of the paper that is not mechanized in Lean) was lacking the argument from Lemma 5.1. The AI agent suggested to use Dickson's lemma to close that gap.

Code. We used AI auto-completion for writing the code of the artifact.

For the proofs, an initial version was generated by an AI agent based on our paper-and-pencil proofs. As the proofs were not closing/not helpful, we rewrote most of them by hand. But the initial skeleton the AI agent drafted remains in place.

We created some helper scripts for the evaluation with an AI agent (random term generation, conversion between our rule formats, and the bash script combining all our evaluation runs).

11 Data Availability Statement

We provide an artifact under 

Our artifact includes the Lean proof under `proof/`. Executing `lake build` checks the proof. The proof checking takes about five minutes, including building the libraries.

The other folders contain the code to reproduce our evaluation. Detailed instructions are located in `eval/ReadMe.md` with an explanation of the experiments (and timings) in `eval/RESULTS.md`. The whole evaluation takes around two hours.

The Docker container for the tool can be built (10 min) with:

```
docker build -f eval/docker/Dockerfile -t rule-enum-eval .
```

To run the experiments use:

```
mkdir -p eval/out-docker
docker run --rm -u "$(id␣-u):$(id␣-g)" -v "$PWD/eval/out-docker:/eval/out" rule
    ↪ -enum-eval
docker run --rm -u "$(id␣-u):$(id␣-g)" -v "$PWD/eval/out-docker:/eval/out" --
    ↪ entrypoint python3 rule-enum-eval report.py
```

These commands run `all` experiments with the output in `eval/out-docker`. `report.py` creates a report in `eval/out-docker/results/report.md` and figures in `eval/out-docker/figures`.

References

Franz Baader and Tobias Nipkow. 1998. *Term Rewriting and All That.* Cambridge University Press.

Leo Bachmair, Nachum Dershowitz, and David A. Plaisted. 1989. Completion Without Failure. In *Rewriting Techniques*, Hassan Aït-Kaci and Maurice Nivat (Eds.). Academic Press, 1–30. doi:10.1016/B978-0-12-046371-8.50007-9

Clark W. Barrett, Christopher L. Conway, Morgan Deters, Liana Hadarean, Dejan Jovanovic, Tim King, Andrew Reynolds, and Cesare Tinelli. 2011. CVC4. In *Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings (Lecture Notes in Computer Science, Vol. 6806)*, Ganesh Gopalakrishnan and Shaz Qadeer (Eds.). Springer, 171–177. doi:10.1007/978-3-642-22110-1_14

Rudy Braquehais and Colin Runciman. 2017. Speculate: Discovering Conditional Equations and Inequalities about Black-Box Functions by Reasoning from Test Results. In *Proceedings of the 10th ACM SIGPLAN International Symposium on Haskell*. ACM, 40–51. doi:10.1145/3122955.3122961

Bruno Buchberger, Adrian Craciun, Tudor Jebelean, Laura Kovács, Temur Kutsia, Koji Nakagawa, Florina Piroi, Nikolaj Popov, Judit Robu, Markus Rosenkranz, and Wolfgang Windsteiger. 2006. Theorema: Towards computer-aided mathematical theory exploration. *J. Appl. Log.* 4, 4 (2006), 470–504. doi:10.1016/J.JAL.2005.10.006

Koen Claessen, Nicholas Smallbone, and John Hughes. 2010. QuickSpec: Guessing Formal Specifications Using Testing. In *Tests and Proofs (TAP 2010) (LNCS, Vol. 6143)*. Springer, 6–21. doi:10.1007/978-3-642-13977-2_3

Leonardo De Moura and Nikolaj Bjørner. 2008. Z3: An efficient SMT solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 337–340.

Leonard Eugene Dickson. 1913. Finiteness of the Odd Perfect and Primitive Abundant Numbers with *n* Distinct Prime Factors. *American Journal of Mathematics* 35, 4 (1913), 413–422. doi:10.2307/2370405

Philippe Flajolet and Robert Sedgewick. 2009. *Analytic combinatorics.* Cambridge University Press.

R. Gurevič. 1985. Equational Theory of Positive Numbers with Exponentiation. *Proc. Amer. Math. Soc.* 94, 1 (1985), 135–141. http://www.jstor.org/stable/2044966

Bastian Hagedorn, Johannes Lenfers, Thomas Koehler, Xueying Qin, Sergei Gorlatch, and Michel Steuwer. 2020. Achieving high-performance the functional way: a functional pearl on expressing high-performance optimizations as rewrite strategies. *Proc. ACM Program. Lang.* 4, ICFP (2020), 92:1–92:29. doi:10.1145/3408974

Zhihao Jia, Oded Padon, James Thomas, Todd Warszawski, Matei Zaharia, and Alex Aiken. 2019. TASO: Optimizing Deep Learning Computation with Automatic Generation of Graph Substitutions. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles (SOSP)*. ACM, 47–62. doi:10.1145/3341301.3359630

Dominik Klein and Nao Hirokawa. 2011. Maximal Completion. In *22nd International Conference on Rewriting Techniques and Applications (RTA 2011) (LIPIcs, Vol. 10)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 71–80. doi:10.4230/LIPIcs.RTA.2011.71

Donald E. Knuth and Peter B. Bendix. 1970. Simple Word Problems in Universal Algebra. In *Computational Problems in Abstract Algebra*, John Leech (Ed.). Pergamon Press, Oxford, 263–297.

Nuno P. Lopes, David Menendez, Santosh Nagarakatte, and John Regehr. 2015. Provably Correct Peephole Optimizations with Alive. In *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*. ACM, 22–32. doi:10.1145/2737924.2737965

Chandrakana Nandi, Max Willsey, Amy Zhu, Yisu Remy Wang, Brett Saiki, Adam Anderson, Adriana Schulz, Dan Grossman, and Zachary Tatlock. 2021. Rewrite rule inference using equality saturation. *Proc. ACM Program. Lang.* 5, OOPSLA, Article 119 (Oct. 2021), 28 pages. doi:10.1145/3485496

Julie L. Newcomb, Andrew Adams, Steven Johnson, Rastislav Bodik, and Shoaib Kamil. 2020. Verifying and improving Halide's term rewriting system with program synthesis. *Proc. ACM Program. Lang.* 4, OOPSLA, Article 166 (Nov. 2020), 28 pages. doi:10.1145/3428234

Andres Nötzli, Andrew Reynolds, Haniel Barbosa, Aina Niemetz, Mathias Preiner, Clark Barrett, and Cesare Tinelli. 2019. Syntax-Guided Rewrite Rule Enumeration for SMT Solvers. In *Theory and Applications of Satisfiability Testing – SAT 2019*, Mikoláš Janota and Inês Lynce (Eds.). Springer International Publishing, Cham, 279–297.

Anjali Pal, Brett Saiki, Ryan Tjoa, Cynthia Richey, Amy Zhu, Oliver Flatt, Max Willsey, Zachary Tatlock, and Chandrakana Nandi. 2023. Equality Saturation Theory Exploration à la Carte. *Proc. ACM Program. Lang.* 7, OOPSLA2, Article 258 (Oct. 2023), 29 pages. doi:10.1145/3622834

Jonathan Ragan-Kelley, Connelly Barnes, Andrew Adams, Sylvain Paris, Frédo Durand, and Saman Amarasinghe. 2013. Halide: a language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines. *Acm Sigplan Notices* 48, 6 (2013), 519–530.

R. Sekar, I. V. Ramakrishnan, and Andrei Voronkov. 2001. Term Indexing. In *Handbook of Automated Reasoning*, Alan Robinson and Andrei Voronkov (Eds.). Vol. II. Elsevier Science and MIT Press, Chapter 26, 1853–1964.

Nicholas Smallbone. 2021. Twee: An Equational Theorem Prover. In *Automated Deduction – CADE 28: 28th International Conference on Automated Deduction, Virtual Event, July 12–15, 2021, Proceedings*. Springer-Verlag, Berlin, Heidelberg, 602–613. doi:10.1007/978-3-030-79876-5_35

Nicholas Smallbone, Moa Johansson, Koen Claessen, and Maximilian Algehed. 2017. Quick Specifications for the Busy Programmer. *Journal of Functional Programming* 27 (2017), e18. doi:10.1017/S0956796817000090

Ross Tate, Michael Stepp, Zachary Tatlock, and Sorin Lerner. 2011. Equality Saturation: A New Approach to Optimization. *Logical Methods in Computer Science* Volume 7, Issue 1 (March 2011). doi:10.2168/lmcs-7(1:10)2011

Ian Wehrman, Aaron Stump, and Edwin Westbrook. 2006. Slothrop: Knuth-Bendix Completion with a Modern Termination Checker. In *17th International Conference on Rewriting Techniques and Applications (RTA 2006) (LNCS, Vol. 4098)*. Springer, 287–296. doi:10.1007/11805618_22

Max Willsey, Chandrakana Nandi, Yisu Remy Wang, Oliver Flatt, Zachary Tatlock, and Pavel Panchekha. 2021. egg: Fast and extensible equality saturation. *Proceedings of the ACM on Programming Languages* 5, POPL (Jan. 2021), 1–29. doi:10.1145/3434304

Sarah Winkler. 2019. Extending Maximal Completion. In *4th International Conference on Formal Structures for Computation and Deduction (FSCD 2019) (LIPIcs, Vol. 131)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 3:1–3:15. doi:10.4230/LIPIcs.FSCD.2019.3

Sarah Winkler, Haruhiko Sato, Aart Middeldorp, and Masahito Kurihara. 2010. Multi-Completion with Termination Tools (System Description). In *Automated Reasoning (IJCAR 2010) (LNCS, Vol. 6173)*. Springer, 306–312. doi:10.1007/978-3-642-14203-1_26

Yihong Zhang, Yisu Remy Wang, Oliver Flatt, David Cao, Philip Zucker, Eli Rosenthal, Zachary Tatlock, and Max Willsey. 2023. Better Together: Unifying Datalog and Equality Saturation. *Proc. ACM Program. Lang.* 7, PLDI, Article 125 (jun 2023), 25 pages. doi:10.1145/3591239