# An Inference Rule for the Acyclicity Property of Term Algebras

Simon Robillard

Chalmers University of Technology, Sweden

**Abstract**

Term algebras are important structures in many areas of mathematics and computer science. Reasoning about their theories in superposition-based first-order theorem provers is made difficult by the acyclicity property of terms, which is not finitely axiomatizable. We present an inference rule that extends the superposition calculus and allows reasoning about term algebras without axioms to describe the acyclicity property. We detail an indexing technique to efficiently apply this rule in problems containing a large number of clauses. Finally we experimentally evaluate an implementation of this extended calculus in the first-order theorem prover Vampire. The results show that this technique is able to find proofs for difficult problems that existing SMT solvers and first-order theorem provers are unable to solve.

## 1 Introduction

Term algebras are central to many areas of mathematics and computer science. In logic, they are closely related to the concept of Herbrand structures, and in other areas of mathematics, they can be used to formalize inductively defined structures. They are also useful in the study of programming languages, particularly those that manipulate inductive data types. The ability to reason about term algebras efficiently in an automatic prover is therefore of great importance.

The main difficulty in reasoning about their theory is caused by the *acyclicity property* of terms, which states that a term cannot be equal to one of its own subterms. This property cannot be described by a finite number of axioms in first-order logic, making it troublesome for first-order theorem provers based on superposition. Such provers find refutation proofs by saturating a set of clauses, and theory reasoning is accomplished by explicitly adding the theory axioms to the set of clauses to be saturated. Term algebra reasoning has therefore often been carried out by using dedicated decision procedures [15,17,18] which typically cannot be used on problems containing other theories. More recently, support for term algebra reasoning has been added to some SMT solvers [2,12], but these are usually not as efficient as superposition-based provers on problems that contain heavily quantified formulas.

We previously tackled the problem of reasoning about term algebras in a superposition prover [6] by introducing a conservative extension of their theories, in which an additional predicate symbol is used to represent the *subterm* relation over terms. The predicate is defined by additional axioms; in particular it has the property of being irreflexive, which corresponds to the restriction that terms cannot be equal to their own subterms. This technique provides an easy way to perform complete reasoning in the theory of term algebras using any first-order theorem prover. However the subterm relation is transitive, which means that provers may generate a large number of clauses containing the subterm predicate, most of which will not be used in the proof.

In this paper we present an alternative solution to reason about the acyclicity property of term algebras. Instead of relying on axioms, we extend the superposition calculus with a new inference rule. The inferences that result from it are sound in all interpretations that satisfy

the acyclicity property. This enables the prover to generate useful new consequences while minimizing the number of generated clauses, thus improving the efficiency of the proof search.

This paper is organized as follows. We define term algebras and their first-order theory in Section 2 and recall some notions about superposition in Section 3. In Section 4, we describe the new inference rule. Technical details allowing the rule to be efficiently implemented in a first-order theorem prover are given in Section 5. Lastly in Section 6, we evaluate this approach, as implemented in the first order-theorem prover Vampire, and compare it to other tools and techniques.

# 2 Term algebras

In this section, we define term algebras and their first-order theories, and describe some of their properties. The context of this presentation is unsorted first-order logic, but the results can be extended to a many-sorted logic in a straightforward manner. Equality is part of the logic, the notation $\approx$ stands for the equality predicate in first-order logic, and $\not\approx$ for its negation.

## 2.1 First-order theory

Let $\Sigma$ be a finite collection of function symbols containing at least one constant. We call these symbols *term constructors* and denote them by the letters $\mathsf{e}$, $\mathsf{f}$, $\mathsf{g}$, ... We denote by $\mathcal{T}(\Sigma)$ the set of ground terms constructed from $\Sigma$.

The $\Sigma$-*term algebra* is the algebraic structure whose carrier set is $\mathcal{T}(\Sigma)$ and in which terms are interpreted as themselves:

- every constant symbol $\mathsf{e}$ is interpreted as the corresponding constant in $\mathcal{T}(\Sigma)$;
- every $n$-ary function symbol $\mathsf{f}$ is interpreted as the function from $\mathcal{T}(\Sigma)^n$ to $\mathcal{T}(\Sigma)$ that maps the tuple $(t_1, \ldots, t_n)$ to the element $\mathsf{f}(t_1, \ldots, t_n)$.

**Definition 1.** We now define the first-order theory $\mathcal{T}_{FT}$ as the set of formulas that are consequences of the following axioms:

$$\forall x, \bigvee_{\mathsf{f} \in \Sigma} \exists \overline{y}\, (x \approx \mathsf{f}(\overline{y})) \tag{A1}$$

$$\forall \overline{xy}, \mathsf{f}(\overline{x}) \not\approx \mathsf{g}(\overline{y}) \tag{A2}$$

for every $\mathsf{f}, \mathsf{g} \in \Sigma$ such that $\mathsf{f} \neq \mathsf{g}$;

$$\forall \overline{xy}, \mathsf{f}(\overline{x}) \approx \mathsf{f}(\overline{y}) \rightarrow \overline{x} \approx \overline{y} \tag{A3}$$

for every $\mathsf{f} \in \Sigma$ of arity $\geq 1$;

$$\forall x, x \not\approx t[x] \tag{A4}$$

for every term $t[x] \neq x$ in which $x$ appears.

Axiom (A1), sometimes called the domain closure axiom, asserts that every element in $\Sigma$ is obtained by applying a term constructor to other elements. Axiom (A2) ensures that terms constructed from different constructors are distinct, while axiom (A3) describes the injectivity of term constructors. (A4) is an axiom schema: the resulting formulas assert that no term can

be equal to one of its own subterms, i.e., terms are acyclic. The $\Sigma$-term algebra is a model of every formula in (A1)–(A4), and therefore of every formula in $\mathcal{T}_{FT}$.

We say that a formula is a *pure term algebra formula* if its language only contains function symbols from $\Sigma$ and the equality predicate symbol. Consider for example a term algebra signature consisting of a constant $\mathsf{z}$ and a unary symbol $\mathsf{s}$. The sentence

$$(\forall xy,\ f(x,\mathsf{z}) \approx x \wedge f(x,\mathsf{s}(y)) \approx \mathsf{s}(f(x,y))) \rightarrow \forall xy,\ f(x,y) \approx f(y,x) \tag{S}$$

is not a pure term algebra formula because it contains the function symbol $f$, which is not part of the set of term constructors $\Sigma$.

The theory $\mathcal{T}_{FT}$ is complete on pure term algebra formulas [8]: for any such formula $\varphi$, either $\varphi \in \mathcal{T}_{FT}$ or $\neg\varphi \in \mathcal{T}_{FT}$.

## 2.2 Acyclicity and induction

The acyclicity property is closely related to the notion of induction. In order to illustrate this, we consider a theory $\mathcal{T}_{FT^{\text{Ind}}}$, in which the acyclicity axiom schema (A4) is replaced by an induction axiom schema.

**Definition 2.** $\mathcal{T}_{FT^{\text{Ind}}}$ is the set of formulas that are consequences of the axioms (A1)–(A3) and of the formulas that instantiate the following axiom schema:

$$\bigwedge_{\mathsf{f}\in\Sigma} \Big(\forall x_1 \dots x_n,\ \varphi(x_1) \wedge \dots \wedge \varphi(x_n) \rightarrow \varphi\big(\mathsf{f}(x_1,\dots,x_n)\big)\Big) \rightarrow \forall x,\ \varphi(x) \tag{A5}$$

for every sentence $\varphi(x)$ in the language in which $x$ is the only free variable.

**Lemma 1** (Properties of $\mathcal{T}_{FT^{\text{Ind}}}$)**.** *The following properties hold:*

1. *$\mathcal{T}_{FT^{Ind}}$ is consistent;*
2. *for every formula $\psi$, if $\psi \in \mathcal{T}_{FT}$, then $\psi \in \mathcal{T}_{FT^{Ind}}$;*
3. *for every pure term algebra formula $\psi$, either $\psi \in \mathcal{T}_{FT^{Ind}}$ or $\neg\psi \in \mathcal{T}_{FT^{Ind}}$;*
4. *for every pure term algebra formula $\psi$, $\psi \in \mathcal{T}_{FT^{Ind}}$ if and only if $\psi \in \mathcal{T}_{FT}$.*

*Proof.* (1) holds because $\mathcal{T}(\Sigma)$ is a model of $\mathcal{T}_{FT^{\text{Ind}}}$. (2) holds because every formula of the axiom schema (A4) is a consequence of the axioms of $\mathcal{T}_{FT^{\text{Ind}}}$ (the acyclicity axioms can be proven by induction). (3) is a consequence of the completeness of $\mathcal{T}_{FT}$ and of (2). Finally for (4), one direction of the equivalence if is given by (2). For the other direction, we must show that $\psi \in \mathcal{T}_{FT^{\text{Ind}}}$ implies $\psi \in \mathcal{T}_{FT}$. By contradiction, assume $\psi \in \mathcal{T}_{FT^{\text{Ind}}}$ and $\psi \notin \mathcal{T}_{FT}$. Then by completeness of $\mathcal{T}_{FT}$, we have that $\neg\psi \in \mathcal{T}_{FT}$, and by (2) it follows that $\neg\psi \in \mathcal{T}_{FT^{\text{Ind}}}$, which contradicts the consistency of $\mathcal{T}_{FT^{\text{Ind}}}$. □

Naturally, if we consider languages that include arbitrary predicate and function symbols, $\mathcal{T}_{FT}$ is a strict subset of $\mathcal{T}_{FT^{\text{Ind}}}$. Consider again the sentence (S) given above: intuitively, the term algebra generated by the signature $\Sigma = \{\mathsf{z},\mathsf{s}\}$ is isomorphic to the algebra of natural numbers, and $f$ can be interpreted as the addition on these numbers. The sentence, which expresses the commutatitvity of $f$, belongs to $\mathcal{T}_{FT^{\text{Ind}}}$ but cannot be proven without induction, and therefore does not belong to $\mathcal{T}_{FT}$.

From the previous results we gather that the acyclicity property is strictly weaker than the principle of induction, while at the same time being sufficiently strong to ensure the completeness of $\mathcal{T}_{FT}$ on pure term algebra formulas.

# 3 First-order logic and superposition

We now recall some definitions related to first-order logic and the superposition calculus. A more complete overview of these topics can be found in [7].

First-order theorem provers work by applying inferences to a set of clauses and adding the conclusions to that set. The typical application is to find refutation proofs, by deriving the empty clause from a set that initially contains hypotheses and a negated conjecture. Satisfiability results are also possible if the empty clause is not found and the set is saturated – no new inferences are possible among its clauses. The calculi used by these provers are based on the superposition calculus, which has the property of being refutationally complete: for any unsatisfiable set of clauses, there exists a refutation proof in the calculus.

Terms appearing in clauses may contain variables (denoted $x$, $y$, $z$, ...) which are implicitly universally quantified. To perform inferences between such quantified clauses, rules in the superposition calculus require the detection of unifiable terms and the computation of a *substitution* under which those terms are equal. A substitution is a function from variables to terms. Given a term $t$ and a substitution $\theta$, we denote by $\theta(t)$ the application of $\theta$ to $t$, in which all occurrences of variables $x_1, \ldots, x_n$ in $t$ have been simultaneously replaced by $\theta(x_1), \ldots, \theta(x_n)$. Application of a substitution can be extended to literals and clauses. The composition of two substitutions $\sigma$ and $\tau$ is the function that takes every variable $x$ to $\tau(\sigma(x))$. It is itself a substitution and is denoted $\tau\sigma$. An *equation* is a pair of terms, denoted $s \stackrel{?}{=} t$, and a substitution $\theta$ is a *unifier* of $s$ and $t$, or a solution of the equation, if $\theta(s) = \theta(t)$. Moreover, if for every unifier $\tau$ of an equation $E$, there exists a substitution $\delta$ such that $\tau = \delta\sigma$, then $\sigma$ is a *most general unifier* (mgu) of $E$. The notions of unifier and mgu can be applied to a finite set of equations, if the substitution is a solution of every equation in the set.

To direct the proof search, first-order theorem provers use a *selection function*, a function that selects a non-empty subset of literals in any non-empty clause. The selection function is used to restrict the number of possible inferences (resolution is performed only on selected literals, for example) while preserving refutational completeness of the system, provided that the function satisfies certain properties. Different selection functions may be used, leading to variations of the calculus. To indicate that a literal is among the selected literals in a clause, we show it over a gray background, e.g., $s \approx t \vee A$.

The calculus is also parametrized by an ordering on terms. This is another important component to limit possible inferences and ensure the efficiency of the proof search. However the rule described in the following does not use order restrictions, therefore we do not describe the notion further.

# 4 An inference rule for acyclicity

In this section we describe an inference rule that extends the superposition calculus and allows reasoning about the acyclicity property without adding the corresponding axioms to the set of clauses to saturate.

In the remainder of this text, it is important to distinguish symbols belonging to $\Sigma$ (*term constructors*) from other symbols. This distinction is necessary because the rule must be applicable and sound even for problems that contain uninterpreted symbols or other theory symbols. Indeed, even clauses resulting from the clausification of a pure term algebra formula may contain non-constructor symbols, in particular those introduced by Skolemization. Constructors are denoted by the letters $\mathsf{e}$, $\mathsf{f}$, $\mathsf{g}$, ... while we use $s$, $t$, $u$, ... to denote arbitrary terms.

**Definition 3.** We say that a term $s$ *occurs under term constructors* in a term $t$, if $t$ is of the form $\mathsf{f}(u_1, \ldots, u_n)$ and there exists $u_i$ with $1 \leq i \leq n$ such that either:

1. $s = u_i$
2. $s$ occurs under term constructors in $u_i$.

We will use the notation $p[s]_\Sigma$ to denote a term in which $s$ occurs under constructors. For any ground terms $s$ and $p[s]_\Sigma$ and any interpretation $\mathcal{I}$ that satisfies the instances of axiom schema (A4), it must be the case that $\mathcal{I}(p[s]_\Sigma) \neq \mathcal{I}(s)$.

**Definition 4.** The inference rule $\mathsf{Acycl}^+$, which takes an arbitrary number $n$ of premises, is defined as follows:

$$\frac{t'_1 \approx p[t_2]_\Sigma \vee C_1 \qquad t'_2 \approx q[t_3]_\Sigma \vee C_2 \qquad \ldots \qquad t'_n \approx r[t'_1]_\Sigma \vee C_n}{\theta(C_1 \vee C_2 \vee \cdots \vee C_n)}\ \mathsf{Acycl}^+$$

where $\theta$ is an mgu of the set of equations $\{t_1 =^? t'_1, \ldots, t_n =^? t'_n\}$.

In essence, this rule finds a set of equalities that contradict the acyclicity property under a certain substitution. Since these equalities cannot all be true under the substitution, the conclusion indicates that at least one $C_i$ must be true. We first illustrate this principle with some examples of concrete applications, then demonstrate the soundness of the inference rule with Lemma 2.

**Example 1.** In this simple example, the rule has only one premise and the unifier is the empty substitution:

$$\frac{s \approx \mathsf{g}(\mathsf{e}, \mathsf{f}(s)) \vee A}{A}$$

**Example 2.** Here is a more complex example of application of the rule, with three premises:

$$\frac{s \approx \mathsf{f}(t) \vee A \qquad t \approx \mathsf{g}(u(x), \mathsf{e}) \vee B[x] \qquad u(s) \approx \mathsf{f}(s) \vee C}{A \vee B[s] \vee C}$$

The substitution $\{x \mapsto s\}$ is used as unifier and applied to the conclusion.

**Example 3.** Consider the clause

$$s \approx \mathsf{f}(u(s)) \vee A$$

The rule may not be applied to this clause without additional premises, as $s$ does not occur under term constructors in the right-hand side of the selected literal. Since $u$ is not a term algebra constructor, there exist interpretations that associate $u(s)$ with a term (in the domain of discourse) not featuring $s$ as a subterm, and in which $s \approx \mathsf{f}(u(s))$ holds.

**Lemma 2** (Soundness of $\mathsf{Acycl}^+$)**.** *For any interpretation $\mathcal{I}$ that satisfies the instances of axiom schema (A4), if the premises of the rule hold in $\mathcal{I}$, then the conclusion holds in $\mathcal{I}$ as well.*

*Proof.* Let $\theta$ be a unifier of the equations $\{t_1 =^? t'_1, \ldots, t_n =^? t'_n\}$ (in particular, $\theta$ may be an mgu, although this is not required for soundness).

By contradiction, assume that $\theta(C_1 \vee C_2 \vee \cdots \vee C_n)$ does not hold in $\mathcal{I}$. Instances of the premises $\theta(t'_1 \approx p[t_2]_\Sigma \vee C_1)$, $\theta(t'_2 \approx q[t_3]_\Sigma \vee C_2)$, $\ldots$, $\theta(t'_n \approx r[t_1]_\Sigma \vee C_n)$ hold in $\mathcal{I}$, therefore it must the case that $\theta(t'_1 \approx p[t_2]_\Sigma)$, $\theta(t'_2 \approx q[t_3]_\Sigma)$, $\ldots$, $\theta(t'_n \approx r[t_1]_\Sigma)$ also hold. Since $\theta(t_i) = \theta(t'_i)$ for $1 \leq i \leq n$, there exists at least one ground term $p'[\theta(t_1)]_\Sigma$ such that $\mathcal{I}(p'[\theta(t_1)]_\Sigma) = \mathcal{I}(\theta(t_1))$, which contradicts the hypothesis on $\mathcal{I}$. □

In addition to $\mathsf{Acycl}^+$, we can also add a rule to deal with disequalities:

$$\frac{t \not\approx p[t]_\Sigma \vee A}{\emptyset}\ \mathsf{Acycl}^-$$

As denoted by the double line, this is a *simplification rule*, i.e., a rule that deletes its premise after application. Here the rule generates no conclusion, but merely deletes a clause that is always true in the theory. Such rules do not add to the deductive power of the calculus, but by deleting useless clauses they lighten the load of the prover and thus play an important practical role. Their application is also very inexpensive and should be carried out eagerly: every time a clause is generated it may be tested against this rule and discarded when applicable.

# 5 Implementation

There exist different saturation algorithms, as described in [13], but in general a prover will maintain a set of *active clauses* such that all possible inferences between these clauses have been performed. Every time a new clause $C$ is selected for inference, all active clauses must be tested to check whether they can participate in an inference with $C$. In the case of the rule $\mathsf{Acycl}^+$, this test is particularly difficult:

1. An arbitrary number of clauses can participate in the inference. Assuming that the number of active clauses is $k$, an exhaustive test of the $2^k$ possible combinations would be very impractical, given that $k$ is often large. In contrast, all other rules of the standard superposition calculus are either unary or binary.

2. The rule requires computing an mgu over a set of equations, rather than over a single equation like other rules of the calculus. While this problem is well-known and can be solved efficiently [9], first-order theorem provers typically avoid solving it directly and instead rely on *term indexing* [16] to retrieve, among a set of indexed terms, all of those that are unifiable with a given term $t$.

In order to achieve a practical implementation of the rule, it is important to develop an indexing strategy enabling the prover to retrieve sets of clauses to be used as premises. In this section we describe an algorithm that accomplishes this.

## 5.1 Data structures

The indexing strategy relies on the use of two auxiliary data structures to retrieve terms, literal and clauses that appear among the set of active clauses.

**Subterm index.** This index must support queries over terms with the following invariant: given a term $s$, the query `subtermClause`$(s)$ must return all pairs $(t, C)$ such that $t$ is a term and $C$ is an active clause of the form $s \approx p[t]_\Sigma \vee A$. This index can be implemented straightforwardly as a map, and must be updated every time a clause is added to or removed from the set of active clauses.

**Unification index.** This index supports a query over terms: given a term $s$, the query `unifiable`$(s)$ returns all pairs $(t, \sigma)$ such that there exists an active clause $s' \approx t \vee A$ and $s'$ is unifiable with $s$ under an mgu $\sigma$. Like the other one, this index must be updated every time the set of active clauses is modified. The efficient implementation of such an index is not trivial: unlike the subterm index, a query upon term $s$ may return results even if $s$ does not appear in

any of the active clauses. Efficient retrieval of unifiable terms is central to the implementation of first-order theorem provers, and any state-of-the-art prover should offer data structures that can be used to implement such an index.

## 5.2 Retrieving premises

Every time a new clause $C$ is selected for inference, the procedure `performInferences`$(C)$ must perform all possible inference between $C$ and active clauses. For this, the subterm and unification indexes are first updated to include $C$, then a search is conducted to find the sets of premises.

We give now a procedure to detect all sets of premises among active clauses – more precisely, all minimal sets with respect to subsumption – and perform the corresponding applications of $\mathsf{Acycl}^+$ among the active clauses (Algorithm 1).

```
Procedure performInferences(C_1) is
    for t s.t. C_1 = t_1 ≈ p[t]_Σ ∨ A do
        enumerate(t_1, t, ϵ, {C_1})
    end
end

Procedure enumerate(t_1, t, θ, P) is
    for (t', σ) ∈ unifiable(θ(t)) do
        if t' = t_1 then
            apply Acycl^+ to P under σθ
        else
            for (t_i, C_i) ∈ subtermClause(t') do
                if C_i ∉ P then
                    enumerate(t_1, t_i, σθ, P ∪ {C_i})
                end
            end
        end
    end
end
```

**Algorithm 1:** A procedure to apply $\mathsf{Acycl}^+$ among active clauses

Given two terms $t_1$ and $t$ and a substitution $\theta$, the procedure `enumerate`$(t_1, t, \theta, P)$ applies $\mathsf{Acycl}^+$ to all minimal sets of premises that include $P$. Every call to `enumerate`$(t_1, t, \theta, P)$ verifies the following invariant: the selected literals in $P$ imply an equality between $\theta(t_1)$ and some term $p[\theta(t)]_\Sigma$. The parameter $P$ is used to collect the premises. The substitution is computed by composing the mgus of each individual equation, starting with the empty substitution $\epsilon$. The correctness of this construction is proven in Lemma 3: the side condition of this lemma is satisfied because variables from distinct clauses are always distinct themselves.

**Lemma 3** (Correctness of the mgu)**.** *Let $\theta$ be an mgu of a set of equations $E$ and $\sigma$ an mgu of an equation $\theta(s) =^? t$ such that $t$ does not contain variables appearing in $E$, then:*

1. *$\sigma\theta$ is a unifier of $E' = E \cup \{s =^? t\}$*

*2. $\sigma\theta$ is a most general unifier of $E'$*

*Proof.* Let us first note that since the variables of $t$ do not appear in $E$, and $\theta$ is an mgu of $E$, the variables of $t$ do not belong to the domain of $\theta$, from which we have that $\theta(t) = t$.

For (1), we have that $\theta$ is a unifier of $E$, therefore for any substitution $\delta$, $\delta\theta$ is a unifier of E, so in particular this holds for $\sigma\tau$. In addition, as $\theta(t) = t$, $\sigma\theta$ is also a unifier of $\theta(s) =^? t$.

For (2), we must show that for any $\alpha$ that is a unifier of $E'$, there exists a substitution $\delta$ such that $\delta\sigma\theta = \alpha$. As $E$ is a subset of $E'$, $\alpha$ must also be a unifier of $E$. As $\theta$ is an mgu of $E$, there exists $\delta_1$ such that $\delta_1\theta = \alpha$. $\delta_1$ is a unifier of $\{\theta(s) =^? t\}$, or equivalently of $\{\theta(s) =^? \theta(t)\}$, and consequently $\delta_1\theta$ is a unifier of $\{\theta(s) =^? t\}$. As $\sigma$ is an mgu for that set, there exists $\delta_2$ such that $\delta_2\sigma = \delta_1$. The substitution $\delta_2$ satisfies $\delta_2\sigma\theta = \alpha$, showing that $\sigma\theta$ is most general. □

**Lemma 4** (Termination)**.** *The procedure* `performInferences` *terminates.*

*Proof.* Termination of the procedure follows from the following facts:

1. Queries to the subterm index and the unification index always return a finite number of results, so that each call to `enumerate`$(t_1, t, \theta, P)$ only makes a finite number of recursive calls.

2. The condition $C_i \notin P$ ensures that the depth of the recursion is bounded by the number of active clauses, as an element is added to $P$ on every recursive call.

□

# 6 Experiments

We implemented the new inference rule in the first-order theorem prover Vampire. As described in [6], Vampire already provides support for term algebra reasoning, relying on a conservative extension of the theory to enforce the acyclicity property. Our new rule can be used instead of this mechanism, and we compare the two approaches below.

Among currently available benchmarks, few problems rely on the acyclicity property: many of them are theorems that hold on term algebras as well as on similar structures that do not satisfy the acyclicity property (such as algebras of rational trees, described in [8]). For example in [12], the authors find that only 6 problems (among 4170) are solved exclusively when the acyclicity rule of CVC4 is activated, a result confirmed by experiments with Vampire. Moreover, those 6 problems are very simple, and any prover implementing some form of reasoning about the acyclicity property should be able to solve them. In order to provide a meaningful evaluation of the performance of the different techniques for handling the acyclicity property, we generated 200 new problems[1] of various size and complexity. The problems were constructed by generating DNF formulas in which each disjunct contains literals that imply a cyclic equality. The formulas generated in this manner contain between 1 and 20 disjuncts, each containing between 1 and 20 literals. The signature of the algebra was also varied across the different problems. We separated the problems in two sets: the first set contains 100 problems without universal quantifiers, so that their clausified forms feature only ground terms; the remaining 100 problems include quantifiers, and therefore variables are present after clausification. No other theories were involved, and the only uninterpreted symbols are constants which can be seen as Skolem symbols, so that the problems belong to the decidable fragment of the theory of term algebras.

[1] These benchmarks are available at `http://www.cse.chalmers.se/~simrob/tools.html`

While these problems do not correspond to real applications of theorem provers, they allowed us to specifically evaluate reasoning about acyclicity. In contrast to other available benchmarks, none of these problems can be solved without some non-trivial way to enforce this property of term algebras.

We compare four solvers/configurations:

1. VAMPIRE$^{\text{INF}}$ uses the extended calculus described in this paper. Axioms (A1)–(A3) are included in the set of clauses to be saturated (together with the problem hypotheses and negated conjecture) but no axioms describing the acyclicity property are used.

2. VAMPIRE$^{\text{EXT}}$ uses the standard superposition calculus, and instead relies on a conservative extension of the theory of term algebras to reason about the acyclicity property. An additional predicate symbol is added to the signature, which corresponds to the subterm relation over terms. In addition to axioms (A1)–(A3), the initial set of clauses also contains formulas that define the subterm relation.

3. CVC4 is an SMT solver that includes a theory solver for the theory of finite term algebras [12]. This solver constructs equivalence classes for terms present in the problem, checking that none of them correspond to infeasible values (cyclic terms).

4. Z3 [5] is another SMT solver with support for this theory.

Apart from the difference highlighted above, VAMPIRE$^{\text{INF}}$ and VAMPIRE$^{\text{EXT}}$ share identical parameters. Notably, simplification rules related to term algebra constructors (as described in [6]) are activated, as well as AVATAR [20]. All experiments were carried out on a cluster on which each node is equipped with two quad core Intel processors running at 2.4 GHz and 24 GiB of memory. The different solvers were run on each problem with a time limit of 60 seconds.

Initial tests with these problems led to some minor optimizations in the implementation of the inference rule. After communication with its developer, the theory solver of CVC4 was also improved on the basis of these benchmarks. The results that we give here take these improvements into account. They are presented in Figure 1.

Among the 100 problems containing only ground hypothesis clauses, VAMPIRE$^{\text{EXT}}$ was able to solve 90 of the problems and exceeded the time limit for the remaining 10 problems. The total time required to solve the 90 problems was nearly 800 seconds. VAMPIRE$^{\text{INF}}$ however was able to solve all of the problems, and took less than 14 seconds to do so. Each individual problem was solved in at most 0.6 second. CVC4 was able to solve all the problems within the time limit, the combined time to solve these problems was 45 seconds. Z3 was the most efficient, solving all the problems in less than 2 seconds.

On the problems containing variables, the results are generally similar for the two approaches based on superposition: VAMPIRE$^{\text{EXT}}$ solved 93 of the problems before the time limit, taking nearly 700 seconds to do it, while VAMPIRE$^{\text{INF}}$ solved all of the problems in less than 12 seconds. The performance of SMT solvers is however noticeably different on this set of problems: Z3 solved 76 of the problems fairly quickly (103 seconds) but exceeded the time limit for the remaining 24 problems, while CVC4 could only solve 12 problems, in 14 seconds.

The new inference rule clearly proves useful for solving difficult problems based on the acyclicity property. In particular, it is the only approach that succeeded in finding proofs for all 200 problems. This approach was also very fast on all benchmarks: in the worst case it took 2.5 seconds to find a proof, while all other problems were solved in less than 1 second.

The presence of non-ground terms in the hypotheses does not affect the performance of VAMPIRE$^{\text{EXT}}$, nor that of VAMPIRE$^{\text{INF}}$, despite the added complexity of finding relevant premises

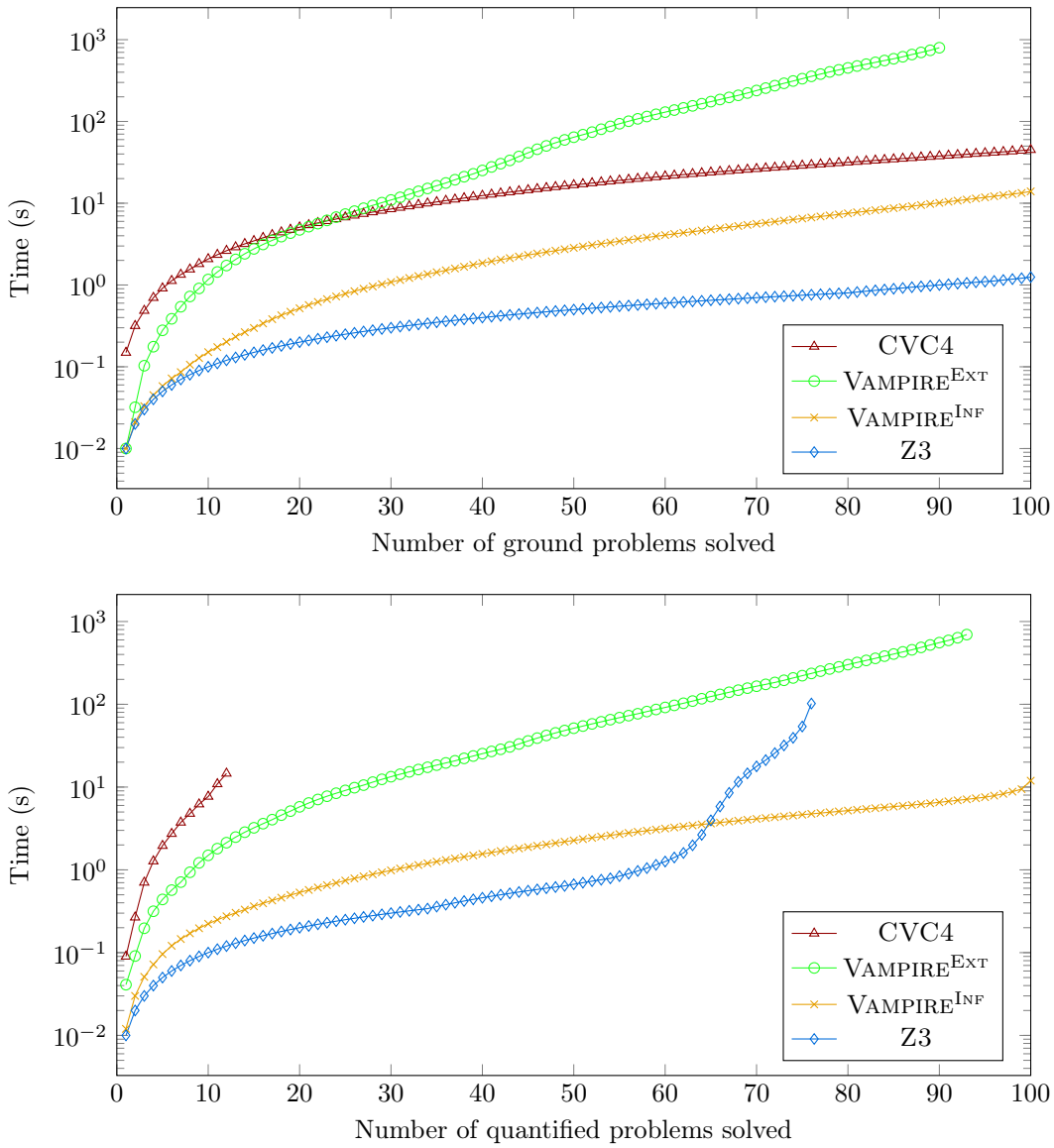


Figure 1: Time required to solve a number of problems among both sets. Where no number is given, the solver was unable to solve some of the problems within the 60-second limit imposed on each run.

among active non-ground clauses. This is in contrast with the performance of SMT solvers, which is negatively affected by the presence of variables in the clauses.

# 7 Related work

The idea of replacing axioms by an inference rule is central to paramodulation [22], and consequently to the advent of useful first-order theorem provers. Paramodulation is a rule that replace the axioms of the equality predicate. While finite, the axiomatization of the equality predicate is potentially large, as one additional formula is required for each symbol in the signature. It is theoretically possible to reason about equality without paramodulation, but in practice only very simple problems can be solved that way.

In its form, the rule presented here shares some similarity with the hyper-resolution rule introduced by Robinson [14], as it finds a resolvent among an arbitrary number of clauses. Efficient implementation of hyper-resolution is notoriously difficult, solutions have been proposed by Overbeek [10]. However, hyper-resolution seems to have fallen out of favor among modern provers, perhaps because it yields only small benefits compared to binary resolution.

The acyclicity property is of some importance in Prolog, where it corresponds to the *occur-check* of the unification algorithm. For performance reasons, many Prolog implementations do not enforce this check. This means that such implementations actually solve equations over algebras of infinite trees [4]. This change in semantics has been formalized by van Emdem et al. [19], while Plaisted [11] and Apt [1] establish criteria to determine whether the omission of the occur-check modifies the semantics of a given program.

Decisions procedures based on quantifier elimination have been used to show the completeness of the theory of term algebras, for example by Maher [8], or by Rybiana et al. in the context of a theory extended with queues [15]. More practical decision procedures have been described and evaluated [17, 18]. Barrett et al. have introduced a theory solver for inductive data types in the SMT solver CVC4 [2] and the SMT solver Z3 uses a comparable theory solver (unpublished work by de Moura). These developments allow reasoning about problems that contain uninterpreted symbols, as well as mixed theories. Reynolds et al. have provided a theory solver that can also reason about co-inductive data types [12], while Bjørner has included a decision procedure for both inductive and co-inductive data types in STeP, the Stanford Temporal Prover [3]. In his PhD thesis, Wand proposes an extension of the superposition calculus with support for inductive data types and inductive reasoning over these types [21].

# 8 Conclusion

We have presented an inference rule aimed at replacing the infinitely many axioms needed to describe the acyclicity property of term algebras. Thanks to the indexing strategy described, an efficient implementation of the rule can be achieved in theorem provers. In comparison to other techniques applicable in saturation-based prover (in particular, the theory extension described in our previous work [6]), this rule generates fewer consequences and does not needlessly expand the search space, leading to better performance. The rule is not proven to be complete with respect to the axioms of acyclicity, but it is empirically shown to outperform other approaches for reasoning about the acyclicity property of term algebras.

Similar approaches could be used to reason about other theories without finite axiomatizations. In particular the theory of infinite trees [4] is a good candidate. This theory provides a first-order semantics for co-inductive data types [12] and shares many similarities with the the-

ory of term algebras. Notably, its uniqueness property – which asserts the existence of unique cyclic elements – is not finitely axiomatizable. A better characterization of that theory and its properties, in particular from the point of view of automated theorem proving, would be helpful for program verification and interactive theorem proving.

## Acknowledgments

We acknowledge funding from the ERC Starting Grant 2014 SYMCAR 639270, the Wallenberg Academy Fellowship 2014 TheProSE, the Swedish VR grant GenPro D0497701, and the Austrian FWF research project RiSE S11409-N23.

## References

[1] Krzysztof R. Apt and Alessandro Pellegrini. Why the occur-check is not a problem. In *Programming Language Implementation and Logic Programming*, volume 631 of *LNCS*, pages 69–86. Springer, 1992.

[2] Clark Barrett, Igor Shikanian, and Cesare Tinelli. An abstract decision procedure for a theory of inductive data types. *Journal on Satisfiability, Boolean Modeling and Computation*, 3:21–46, 2007.

[3] Nikolaj Bjørner. *Integrating Decision Procedures for Temporal Verification*. PhD thesis, Stanford University, 1999.

[4] Alain Colmerauer. Prolog and infinite trees. *Logic Programming*, 16:231–251, 1982.

[5] Leonardo de Moura and Nikolaj Bjørner. Z3: An efficient SMT solver. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 4963 of *LNCS*, pages 337–340. Springer, 2008.

[6] Laura Kovács, Simon Robillard, and Andrei Voronkov. Coming to terms with quantified reasoning. In *Symposium on Principles of Programming Languages*, pages 260–270. ACM, 2017.

[7] Laura Kovács and Andrei Voronkov. First-order theorem proving and Vampire. In *International Conference on Computer Aided Verification*, volume 8044 of *LNCS*, pages 1–35. Springer, 2013.

[8] Michael J. Maher. Complete axiomatizations of the algebras of finite, rational and infinite trees. In *Symposium on Logic in Computer Science*, pages 348–357. IEEE, 1988.

[9] Alberto Martelli and Ugo Montanari. An efficient unification algorithm. *ACM Transactions on Programming Languages and Systems*, 4(2):258–282, 1982.

[10] Ross A. Overbeek. An implementation of hyper-resolution. *Computers & Mathematics with Applications*, 1(2):201–214, 1975.

[11] David A. Plaisted. The occur-check problem in Prolog. *New Generation Computing*, 2(4):309–322, 1984.

[12] Andrew Reynolds and Jasmin Christian Blanchette. A decision procedure for (co)datatypes in SMT solvers. *Journal of Automated Reasoning*, 58(3):341–362, 2017.

[13] Alexandre Riazanov and Andrei Voronkov. Limited resource strategy in resolution theorem proving. *Journal of Symbolic Computation*, 36(1):101–115, 2003.

[14] John Alan Robinson. Automatic deduction with hyper-resolution. *International Journal of Computing and Mathematics*, 1:227–234, 1965.

[15] Tatiana Rybina and Andrei Voronkov. A decision procedure for term algebras with queues. *ACM Transactions on Computational Logic*, 2(2):155–181, 2001.

[16] R. Sekar, I.V. Ramakrishnan, and Andrei Voronkov. Term indexing. In *Handbook of Automated Reasoning*, pages 1853–1964. Elsevier, 2001.

[17] Philippe Suter, Mirco Dotta, and Viktor Kuncak. Decision procedures for algebraic data types with abstractions. *Acm Sigplan Notices*, 45(1):199–210, 2010.

[18] Dao Thi-Bich-Hanh. *Résolution de Contraintes du Premier Ordre dans la Théorie des Arbres Finis ou Infinis*. PhD thesis, Université Aix-Marseille 2, 2000.

[19] Maarten H. van Emden and John W. Lloyd. A logical reconstruction of Prolog II. *The Journal of Logic Programming*, 1(2):143–149, 1984.

[20] Andrei Voronkov. AVATAR: The architecture for first-order theorem provers. In *International Conference on Computer Aided Verification*, volume 8559 of *LNCS*, pages 696–710. Springer, 2014.

[21] Daniel Wand. *Superposition: Types and Induction*. PhD thesis, Saarland University, 2017.

[22] Lawrence Wos and George Robinson. Paramodulation and set of support. In *Symposium on Automatic Demonstration*, volume 125 of *LNM*, pages 276–310. Springer, 1970.