Superposition with Datatypes and Codatatypes

Citation for the original published paper (version of record):
Blanchette, J., Peltier, N., Robillard, S. (2018). Superposition with Datatypes and Codatatypes. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 10900 LNAI: 370-387.
http://dx.doi.org/10.1007/978-3-319-94205-6_25

Superposition with Datatypes and Codatatypes

Jasmin Christian Blanchette[1,2], Nicolas Peltier[3], and Simon Robillard[4(✉)]

[1] Vrije Universiteit Amsterdam, Amsterdam, The Netherlands
[2] Max-Planck-Institut für Informatik, Saarland Informatics Campus, Saarbrücken, Germany
[3] Univ. Grenoble Alpes, CNRS, Grenoble INP, LIG, F-38000 Grenoble, France
[4] Chalmers University of Technology, Gothenburg, Sweden
simon.robillard@chalmers.se

Abstract. The absence of a finite axiomatization of the first-order theory of datatypes and codatatypes represents a challenge for automatic theorem provers. We propose two approaches to reason by saturation in this theory: one is a conservative theory extension with a finite number of axioms; the other is an extension of the superposition calculus, in conjunction with axioms. Both techniques are refutationally complete with respect to nonstandard models of datatypes and non-branching codatatypes. They take into account the acyclicity of datatype values and the existence and uniqueness of cyclic codatatype values. We implemented them in the first-order prover Vampire and compare them experimentally.

1 Introduction

The ability to reason about inductive and coinductive datatypes has many applications in program verification, formalization of the metatheory of programming languages, and even formalization of mathematics. Inductive datatypes, or simply *datatypes*, consist of finite values freely generated from constructors. Coinductive datatypes, or *codatatypes*, additionally support infinite values. Non-freely generated (co)datatypes are also useful. All of these variants can be seen as members of a single unifying framework (Section 2).

It is well known that the first-order theory of datatypes cannot be finitely axiomatized. Distinctness, injectivity, and exhaustiveness of constructors are easy to axiomatize, but acyclicity is more subtle, and for induction we would need an axiom schema or a second-order axiom. Codatatypes are also problematic: Besides a coinduction principle that is dual to induction, they are characterized by the existence of all possible infinite values, corresponding intuitively to infinite ground terms. Both datatypes and codatatypes represent a challenge for automatic theorem provers.

Superposition [2] is a highly successful calculus for reasoning about first-order clauses and equality. There has been some work on extending superposition with induction [10, 24], including by Kersani and Peltier [11], and on the axiomatization of datatypes, including by Kovács, Robillard, and Voronkov [12]. In this paper, we propose both axiomatizations and extensions of the superposition calculus to support freely and non-freely generated datatypes as well as codatatypes.

We first focus on a conservative extension of the theory with a finite number of first-order axioms that capture the basic properties of constructors, acyclicity of datatype values, uniqueness of cyclic (ω-regular) codatatype values, and existence of all codatatype cyclic values (Section 3). These axioms admit nonstandard models; for example, for the

Peano-style natural numbers freely generated by $\mathsf{zero} : \mathit{nat}$ and $\mathsf{suc} : \mathit{nat} \to \mathit{nat}$, we cannot exclude the familiar nonstandard models of arithmetic, in which arbitrarily many copies of $\mathbb{Z}$ may appear besides $\mathbb{N}$. Similarly, the domains interpreting codatatypes are not guaranteed to contain all infinite acyclic values.

The axiomatization of codatatypes up to a suitable notion of nonstandard models constitutes the first theoretical contribution of this paper. Our second theoretical contribution is an extension of superposition with inference rules to reason about datatypes and codatatypes (Section 4). This is inspired by an acyclicity rule that Robillard presented at the Vampire 2017 workshop [22]. The main distinguishing feature of our rules is that they are (in combination with a few axioms) refutationally complete and their side conditions have some new order restrictions, helping prune the search space. On the other hand, our approach also requires a relaxation of the side conditions of the superposition rule: For clauses of the form $\mathsf{c}(\bar{s}) \approx t \vee \mathcal{C}$, where c is a constructor and the first literal is maximal, inferences onto t must be performed, as in ordered paramodulation [1]. In addition, we propose calculus extensions to reason about codatatypes.

Both the theory extension and the calculus extension are designed to be refutationally complete with respect to nonstandard models of datatypes and *nonbranching* codatatypes—codatatypes whose constructors have at most one corecursive argument (Section 5). Due to space restrictions, we can only briefly sketch the proof in this paper. We refer to our technical report [8] for detailed justifications and further explanations.

The calculus extension can be integrated into the given clause algorithm that forms the core of a prover's saturation loop (Section 6). The inference partners for the acyclicity and uniqueness rules can be located efficiently. We implemented both the axiomatic and the calculus approaches in the first-order prover Vampire [13] and compare them empirically on Isabelle/HOL [17] benchmarks and on crafted benchmarks (Section 7).

2 Syntax and Semantics

Our setting is a many-sorted first-order logic. We let τ, υ range over simple types (sorts), s, t, u, v range over terms, $\mathsf{a}, \mathsf{b}, \mathsf{c}, \ldots$ range over function symbols, x, y, z range over variables, and $\mathcal{C}, \mathcal{D}, \mathcal{E}$ range over clauses. Literals are atoms of the form $s \approx t$ or $\neg\, s \approx t$, also written $s \not\approx t$. Clauses are finite disjunctions of literals, viewed as multisets. Substitutions are written in postfix notation, with $s\sigma\theta = (s\sigma)\theta$. The notation $\bar{x}$ represents a tuple $(x_1, \ldots, x_m)$, and $[m, n]$ denotes the set $\{m, m+1, \ldots, n\}$, where $m \leq n+1$.

A *position p of type τ in t* is a position in t such that $t|_p$ is of type τ. If s, t are terms and P is a set of positions of the same type as s in t, then $t[s]_P$ denotes the term obtained from t by replacing the subterms occurring at a position in P by s: $t[s]_P := s$ if $\varepsilon \in P$; $t[s]_P := t$ if $P = \varnothing$; and $\mathsf{f}(t_1, \ldots, t_n)[s]_P := \mathsf{f}(t_i[s]_{P_i})_{i \in [1,n]}$, with $P_i = \{q \mid i.q \in P\}$ otherwise. Given two positions p and q, we write $p < q$ if p is a proper prefix of q. Let $\mathcal{C}tr$ be a distinguished finite set of function symbols, called *constructors*. We reserve the letters $\mathsf{c}, \mathsf{d}, \mathsf{e}$ for constructors. A *constructor position* in t is a position q in t such that for every $p < q$, the head symbol of $t|_p$ is a constructor.

Definition 1. The set of *constructor contexts* of profile $\tau \to \upsilon$ is defined inductively as follows: (1) if t is a term of type υ, then t is a constructor context of profile $\tau \to \upsilon$; (2) if $\Gamma_1, \ldots, \Gamma_n$ are constructor contexts of profile $\tau \to \tau_i$ and $\mathsf{c} : \tau_1 \times \cdots \times \tau_n \to \upsilon$ is a

constructor, then $\mathsf{c}(\Gamma_1,\ldots,\Gamma_n)$ is a constructor context of profile $\tau \to \upsilon$; (3) the hole $\bullet$ is a constructor context of profile $\upsilon \to \upsilon$.

Every constructor context can be written as $\Gamma[\bullet]_P$, where P is a set of constructor positions of the same type in Γ, denoting the positions of $\bullet$ in Γ. It is *empty* if $\varepsilon \in P$, and *constant* if $P = \varnothing$. We write $\Gamma[\bullet]_p$ as an abbreviation for $\Gamma[\bullet]_{\{p\}}$, and we write $\Gamma[t]_P$ to denote the term obtained by replacing every position of P by the term t in the context $\Gamma[\bullet]_P$. Moreover, we write $\tau \rhd \upsilon$ ("υ depends on τ") if there exists a constructor of profile $\tau_1 \times \cdots \times \tau_n \to \upsilon$, with $\tau = \tau_i$ for some $i \in [1,n]$, and $\tau \sim \upsilon$ if $\tau \rhd^* \upsilon$ and $\upsilon \rhd^* \tau$.

The axioms and rules in this paper are parameterized by the following sets. Let $\mathcal{T}_{\mathsf{ind}}$ and $\mathcal{T}_{\mathsf{coind}}$ be disjoint sets of types, intended to model datatypes and codatatypes, respectively, and assume that the codomain of every constructor is in $\mathcal{T}_{\mathsf{ind}} \cup \mathcal{T}_{\mathsf{coind}}$. Let $\mathcal{C}tr_{\mathsf{inj}} \subseteq \mathcal{C}tr$ be a set of constructors, denoting injective constructors. Let $\bowtie$ be a binary symmetric and irreflexive relation among constructors; $\mathsf{c} \bowtie \mathsf{d}$ indicates that terms with head symbol c are always distinct from terms with head symbol d.

We introduce some properties of interpretations that are intended to capture some of the properties of (co)datatypes. An interpretation $\mathcal{I}$ satisfies

- **Exh** (exhaustiveness) iff, for every type $\tau \in \mathcal{T}_{\mathsf{ind}} \cup \mathcal{T}_{\mathsf{coind}}$, $\mathcal{I} \models \bigvee_{i=1}^{m} \exists \bar{x}_i.\ x \approx \mathsf{c}_i(\bar{x}_i)$, where x is a variable of type τ, $\{\mathsf{c}_1,\ldots,\mathsf{c}_m\}$ is the set of constructors of codomain τ, and $\bar{x}_i$ is a vector of pairwise distinct variables of the appropriate length and types;
- **Inf** (infiniteness) iff, for every type $\tau \in \mathcal{T}_{\mathsf{ind}} \cup \mathcal{T}_{\mathsf{coind}}$, the domain of τ is infinite;
- **Acy** (acyclicity, for datatypes) iff, for every type $\tau \in \mathcal{T}_{\mathsf{ind}}$ and for every nonempty constructor context $\Gamma[\bullet]_p$ of profile $\tau \to \tau$, where p is a position, we have $\mathcal{I} \models \Gamma[x]_p \not\approx x$, where x is a variable of type τ not occurring in Γ;
- **FP** (existence and uniqueness of fixpoints, for codatatypes) iff, for every type $\tau \in \mathcal{T}_{\mathsf{coind}}$, for every nonempty constructor context $\Gamma[\bullet]_P : \tau \to \tau$, $\mathcal{I} \models (\exists x.\, \Gamma[x]_P \approx x) \wedge (\Gamma[x]_P \approx x \wedge \Gamma[y]_P \approx y \Rightarrow x \approx y)$, where x, y are fresh variables of type τ;
- **Dst** (distinctness of constructors) iff, for every pair of constructors c, d of the same codomain such that $\mathsf{c} \bowtie \mathsf{d}$, $\mathcal{I} \models \mathsf{c}(\bar{x}) \not\approx \mathsf{d}(\bar{y})$ where $\bar{x}$ and $\bar{y}$ are disjoint vectors of pairwise distinct variables of the appropriate length and types;
- **Inj** (injectivity) iff, for every n-ary constructor $\mathsf{c} \in \mathcal{C}tr_{\mathsf{inj}}$ and pairwise distinct variables $x_1,\ldots,x_n, y_1,\ldots,y_n$ of the appropriate types, $\mathcal{I} \models \mathsf{c}(x_1,\ldots,x_n) \approx \mathsf{c}(y_1,\ldots,y_n) \Rightarrow \bigwedge_{i=1}^{n} x_i \approx y_i$.

Most datatypes occurring in practice are recursive, so condition **Inf** is usually satisfied. In particular, it is the case for any nonempty freely generated (co)datatype τ such that $\tau \rhd^+ \tau$. Conditions **Dst** and **Inj** are defined by finite sets of axioms, but not conditions **Acy** and **FP**. In Section 3, we introduce conservative extensions of the considered formula so that conditions **Acy** and **FP** are satisfied. Then in Section 4, we replace some of these axioms by inference rules.

We assume that $\tau \not\rhd \upsilon$ whenever $\tau \in \mathcal{T}_{\mathsf{ind}}$ and $\upsilon \in \mathcal{T}_{\mathsf{coind}}$. Intuitively, this condition means that a datatype cannot be defined by mutual recursion with a codatatype, which is a very natural restriction [7]. If this condition does not hold, it is easy to see that there is no interpretation that satisfies both **Acy** and **FP**. On the other hand, we may have $\tau \rhd^+ \upsilon$ or $\upsilon \rhd^+ \tau$ with $\tau \in \mathcal{T}_{\mathsf{ind}}$ and $\upsilon \in \mathcal{T}_{\mathsf{coind}}$. There may also exist types not belonging to $\mathcal{T}_{\mathsf{ind}} \cup \mathcal{T}_{\mathsf{coind}}$, and the types in $\mathcal{T}_{\mathsf{ind}} \cup \mathcal{T}_{\mathsf{coind}}$ may depend on them. Finally, we assume that for each type τ, there exists a ground term t of type τ.

3 Axioms

The axioms Exhaust for exhaustiveness, Dist for distinctness, and Inj for injectivity correspond to the formulas used to express the properties **Exh**, **Dst**, and **Inj** in Section 2. The other axioms are introduced below.

For all types $\tau \sim \upsilon$, we introduce a predicate symbol $\mathsf{sub}^{\tau}_{\upsilon}$ on $\tau \times \upsilon$ together with the following axioms, where $\tau \sim \upsilon \sim \upsilon'$ and $\mathsf{c} : \cdots \times \upsilon \times \cdots \to \upsilon'$ is a constructor:

$$\mathsf{Sub}_1\colon\ \mathsf{sub}^{\tau}_{\tau}(x,x) \qquad \mathsf{Sub}_2\colon\ \neg\,\mathsf{sub}^{\tau}_{\upsilon}(x,y) \vee \mathsf{sub}^{\tau}_{\upsilon'}(x,\mathsf{c}(\bar{z},y,\bar{z}'))$$
$$\mathsf{NSub}\colon\ \neg\,\mathsf{sub}^{\upsilon'}_{\tau}(\mathsf{c}(\bar{z},x,\bar{z}'),x) \quad \text{if } \tau \in \mathcal{T}_{\mathsf{ind}}$$

Let $\mathsf{Sub} = \mathsf{Sub}_1 \wedge \mathsf{Sub}_2$. The least fixpoint model of Sub is the usual subterm relation for constructor terms. The axiom NSub states that no term of a type in $\mathcal{T}_{\mathsf{ind}}$ may occur at a nonempty constructor position in itself.

Definition 2. An interpretation $\mathcal{I}$ is sub-*minimal* if, for all $\tau \sim \upsilon$, it satisfies the equivalence $\mathsf{sub}^{\tau}_{\upsilon}(x,y) \Leftrightarrow \bigvee\{\exists \bar{z}.\ y \approx \Gamma[x]_p \mid \Gamma[\bullet]_p \text{ is a constructor context of profile } \tau \to \upsilon\}$, where $\bar{z}$ denotes the vector of variables in Γ that are distinct from x, y.

For every pair of types $\tau, \upsilon \in \mathcal{T}_{\mathsf{coind}}$ with $\tau \sim \upsilon$, we introduce a type $\boxed{\tau}_{\upsilon}$ to denote contexts $\Gamma[\bullet]_P$ of profile $\upsilon \to \tau$. Let hole_{υ} be a constant of type $\boxed{\upsilon}_{\upsilon}$, denoting an empty context. All constructors $\mathsf{c} : \tau_1 \times \cdots \times \tau_n \to \tau$ and types υ such that $\exists i\, \upsilon \rhd^* \tau_i$ are associated with new n-ary constructors $\boxed{\mathsf{c}}_{\upsilon} : \upsilon_1 \times \cdots \times \upsilon_n \to \boxed{\tau}_{\upsilon}$, where for every $i \in [1,n]$, $\upsilon_i = \boxed{\tau_i}_{\upsilon}$ if $\upsilon \rhd^* \tau_i$ and $\upsilon_i = \tau_i$ otherwise. Let $\mathsf{app}^{\tau}_{\upsilon} : \boxed{\tau}_{\upsilon} \times \upsilon \to \tau$, $\mathsf{cyc}_{\upsilon} : \boxed{\upsilon}_{\upsilon} \to \upsilon$, and $\mathsf{cst}^{\tau}_{\upsilon} : \tau \to \boxed{\tau}_{\upsilon}$ be new function symbols. Intuitively, if y denotes the context $\Gamma[\bullet]_P$, then $\mathsf{app}(y,x)$ denotes the term $\Gamma[x]_P$, $\mathsf{cyc}(y)$ denotes the fixpoint of $\Gamma[\bullet]_P$, and $\mathsf{cst}^{\tau}_{\upsilon}$ denotes a constant context (i.e., a context $\Gamma[\bullet]_P$ with $P = \varnothing$).

We consider the following axioms, where $\upsilon \in \mathcal{T}_{\mathsf{coind}}$ and x, y, x_i, z_i are pairwise distinct variables of the appropriate types:

$$\mathsf{App}_1\colon\ \mathsf{app}^{\tau}_{\upsilon}(\mathsf{cst}^{\tau}_{\upsilon}(x),y) \approx x \qquad \mathsf{App}_2\colon\ \mathsf{app}^{\upsilon}_{\upsilon}(\mathsf{hole}_{\upsilon},y) \approx y$$
$$\mathsf{App}_3\colon\ \mathsf{app}^{\tau}_{\upsilon}(\boxed{\mathsf{c}}_{\upsilon}(x_1,\ldots,x_n),y) \approx \mathsf{c}(t_1,\ldots,t_n)$$
$$\text{if } \mathsf{c} : \tau_1 \times \cdots \times \tau_n \to \tau \text{ is a constructor and } \exists i\, \upsilon \rhd^* \tau_i$$
$$\text{with } t_i = \mathsf{app}^{\tau_i}_{\upsilon}(x_i,y) \text{ if } \upsilon \rhd^* \tau_i \text{ and } t_i = x_i \text{ otherwise}$$
$$\mathsf{Uniq}\colon\ x \approx \mathsf{hole}_{\upsilon} \vee y \not\approx \mathsf{app}^{\upsilon}_{\upsilon}(x,y) \vee z \not\approx \mathsf{app}^{\upsilon}_{\upsilon}(x,z) \vee y \approx z$$
$$\mathsf{Cycl}\colon\ \mathsf{cyc}_{\upsilon}(x) \approx \mathsf{app}^{\upsilon}_{\upsilon}(x,\mathsf{cyc}_{\upsilon}(x))$$
$$\mathsf{Hole}_1\colon\ \mathsf{hole}_{\upsilon} \not\approx \mathsf{cst}^{\upsilon}_{\upsilon}(x) \qquad \mathsf{Hole}_2\colon\ \mathsf{hole}_{\upsilon} \not\approx \boxed{\mathsf{c}}_{\upsilon}(x_1,\ldots,x_n) \quad \text{if } \mathsf{c} : \cdots \to \upsilon$$

Let $\mathsf{App} = \mathsf{App}_1 \wedge \mathsf{App}_2 \wedge \mathsf{App}_3$ and $\mathsf{Hole} = \mathsf{Hole}_1 \wedge \mathsf{Hole}_2$.

Example 3. Let $\mathsf{c} : \tau_0 \times \upsilon \to \tau$ be a constructor, with $\upsilon \rhd^* \tau_0$. Then the profile of $\boxed{\mathsf{c}}_{\upsilon}$ is $\boxed{\tau_0}_{\upsilon} \times \boxed{\upsilon}_{\upsilon} \to \boxed{\tau}_{\upsilon}$. The term $t := \boxed{\mathsf{c}}_{\upsilon}(\mathsf{cst}^{\tau_0}_{\upsilon}(x), \mathsf{hole}_{\upsilon})$ encodes the constructor context $\mathsf{c}(x,\bullet)$. If $\mathsf{a} : \upsilon$, then $\mathsf{app}^{\tau}_{\upsilon}(t,\mathsf{a}) =_{\mathsf{App}} \mathsf{c}(x,\mathsf{a})$, where $=_{\mathsf{App}}$ denotes equality modulo App (i.e., $s =_{\mathsf{App}} t \Leftrightarrow \mathsf{App} \models s \approx t$).

Lemma 4 (Soundness of the Axioms). *If interpretation $\mathcal{I}$ satisfies* **Acy** *and* **FP**, *there exists a* sub-*minimal extension of $\mathcal{I}$ validating* Sub, NSub, App, Uniq, Cycl, *and* Hole.

Lemma 5 (Completeness of the Axioms). *Any model of the set of axioms* $\{\mathsf{Sub}, \mathsf{NSub}, \mathsf{App}, \mathsf{Uniq}, \mathsf{Cycl}, \mathsf{Hole}\}$ *fulfills* **Acy** *and* **FP**.

Lemma 6 (Completeness of the Theory). *Let $\mathcal{T}$ be the theory of free constructors, as defined by the properties* **Exh**, **Inf**, **Acy**, **FP**, **Dst**, *and* **Inj**, *with $\mathcal{C}tr_{\mathsf{inj}} = \mathcal{C}tr$ and $\mathsf{c} \bowtie \mathsf{d}$ for all distinct constructors c and d. If S is a first-order sentence in which the only symbols occurring are constructors and equality ($\approx$), then either $\mathcal{T} \models S$ or $\mathcal{T} \models \neg S$.*

Comon and Lescanne [9] provide a decision procedure for equational formulas over finite and infinite trees, which correspond respectively to freely generated datatypes and codatatypes. It is based on a collection of equivalence-preserving transformation rules for eliminating quantifiers and normalizing the formulas. The set of formulas $T = \{\mathsf{Dist}, \mathsf{Inj}, \mathsf{Exhaust}, \mathsf{Sub}, \mathsf{NSub}, \mathsf{App}, \mathsf{Uniq}, \mathsf{Cycl}, \mathsf{Hole}\}$ forms the axiomatization of a conservative extension of the theory of (co)datatypes. We can thus derive a decision procedure for testing satisfiability of first-order sentences S containing only constructors symbols and the equality predicate in the above theory. By interleaving the steps of two fair saturation procedures of the superposition calculus, the first over $S \cup T$ and the second over $\neg S \cup T$, one of the two attempts is guaranteed to derive a refutation in finite time.

4 Inference Rules

As an alternative to the above axiomatization, we propose an extension of the superposition calculus [2] with dedicated rules. Unless otherwise noted, the usual conventions of superposition apply. The standard notion of redundancy is used, with respect to the theory of equality. The notation $s \not\approx t$ indicates that the literal is selected, or that it is maximal in its clause, after the substitution σ has been applied, and nothing is selected, whereas $s \approx t$ indicates that the literal is strictly maximal in its clause, after σ, and no literal is selected. We let $[\neg]\, s \approx t$ stand for either $s \approx t$ or $s \not\approx t$.

Superposition. We denote by $\mathcal{SP}$ the usual rules of the superposition calculus with a slight relaxation: Superposition inside the nonmaximal term of an equation is allowed if the head of the maximal term is a constructor. This ensures that in the rewrite system built from saturated clause sets for defining a model, the right-hand side of every rule is irreducible if the head of the left-hand side is a constructor. Thus, our superposition rule is as follows:

$$\frac{u \approx v \vee \mathcal{D} \qquad [\neg]\, s[u'] \approx t \vee \mathcal{C}}{\sigma([\neg]\, s[v] \approx t \vee \mathcal{D} \vee \mathcal{C})}\ \mathsf{Sup}$$

where $\sigma = \mathrm{mgu}\,\{u \stackrel{?}{=} u'\}$, u' is not a variable, and $\sigma(u) \not\preceq \sigma(v)$; moreover, $\sigma(s[u']) \not\preceq \sigma(t)$ if $[\neg]$ is $\neg$ or if the head symbol of t is not a constructor. The equality resolution and equality factoring rules are the standard ones.

Infiniteness. The next rule captures infiniteness of (co)datatypes:

$$\frac{\left(\bigvee_{i=1}^{n} x \approx t_i\right) \vee \mathcal{C}}{\mathcal{C}}\ \mathsf{Inf}$$

if x is a variable of a type $\tau \in \mathcal{T}_{\mathsf{ind}} \cup \mathcal{T}_{\mathsf{coind}}$ and does not occur in $\mathcal{C}$ or $t_1, \ldots, t_n$.

Lemma 7 (Soundness of Inf). *Let N be a clause set, and let $\mathcal{I}$ be a model of N satisfying* **Inf**. *If $\mathcal{C}$ is derived from N by* Inf, *then $\mathcal{I} \models \mathcal{C}$.*

Distinctness. The distinctness property of constructors takes the form of two rules:

$$\frac{\mathsf{c}(\bar{s}) \approx t \vee \mathcal{C}}{\mathcal{C}\sigma}\ \mathsf{Dist}_1$$

if $\sigma = \mathrm{mgu}\{t \stackrel{?}{=} \mathsf{d}(\bar{x})\}$, where $\mathsf{c} \bowtie \mathsf{d}$ and $\bar{x}$ are fresh pairwise distinct variables; and

$$\frac{\mathsf{d}(\bar{t}) \approx u' \vee \mathcal{D} \quad \mathsf{c}(\bar{s}) \approx u \vee \mathcal{C}}{(\mathcal{C} \vee \mathcal{D})\sigma}\ \mathsf{Dist}_2$$

if $\mathsf{c} \bowtie \mathsf{d}$, $\sigma = \mathrm{mgu}\{u \stackrel{?}{=} u'\}$, $\mathsf{c}(\bar{s})\sigma \not\trianglelefteq u\sigma$, and $\mathsf{d}(\bar{t})\sigma \not\trianglelefteq u'\sigma$.

Proposition 8 (Soundness of Dist_1 and Dist_2). *Let N be a clause set, and let $\mathcal{I}$ be a model of N satisfying* **Dst**. *If a clause $\mathcal{C}$ is derived from N by* Dist_1 *or* Dist_2*, then $\mathcal{I} \models \mathcal{C}$.*

Remark 9. If t is not a variable, the premise of Dist_1 is redundant after the rule is applied. Unifying t with $\mathsf{c}(\bar{x})$ can be useful when t is a variable. For example, from the clause $\mathsf{c}(x) \approx x$, we can derive $\Box$ by unifying x with $\mathsf{d}(\bar{y})$, where $\mathsf{d} \bowtie \mathsf{c}$.

Injectivity. The injectivity property of constructors is also captured by two rules:

$$\frac{\mathsf{c}(s_1, \ldots, s_m) \approx t \vee \mathcal{C}}{(s_i \approx x_i \vee \mathcal{C})\sigma}\ \mathsf{Inj}_1$$

if $i \in [1,m]$, $\mathsf{c} \in \mathcal{C}tr_{\mathsf{inj}}$, $\sigma = \mathrm{mgu}\{t \stackrel{?}{=} \mathsf{c}(x_1, \ldots, x_m)\}$, and $x_1, \ldots, x_m$ are fresh; and

$$\frac{\mathsf{c}(s_1, \ldots, s_m) \approx u' \vee \mathcal{D} \quad \mathsf{c}(t_1, \ldots, t_m) \approx u \vee \mathcal{C}}{(s_i \approx t_i \vee \mathcal{C} \vee \mathcal{D})\sigma}\ \mathsf{Inj}_2$$

if $i \in [1,m]$, $\mathsf{c} \in \mathcal{C}tr_{\mathsf{inj}}$, $\sigma = \mathrm{mgu}\{u \stackrel{?}{=} u'\}$, $u\sigma \not\trianglerighteq \mathsf{c}(\bar{s})\sigma$, and $u'\sigma \not\trianglerighteq \mathsf{c}(\bar{t})\sigma$.

Proposition 10 (Soundness of Inj_1 and Inj_2). *Let N be a clause set, and let $\mathcal{I}$ be a model of N satisfying* **Inj**. *If a clause $\mathcal{C}$ is derived from N by* Inj_1 *or* Inj_2*, then $\mathcal{I} \models \mathcal{C}$.*

Remark 11. If Inj_1 is applied on every argument $i \in [1,m]$ and t is not a variable, the premise becomes redundant and can be removed. Unifying t with the term $\mathsf{c}(x_1, \ldots, x_m)$ is useful when t is a variable. For example, given the clause $\mathsf{c}(x, \mathsf{a}) \approx x$, we can derive $\mathsf{a} \approx x_2$ by Inj_1, from which $\Box$ can be derived by Inf.

Acyclicity. The acyclicity rule attempts to detect constraints that would force a datatype value to be cyclic. The simplest example is a clause of the form $\Gamma[s] \approx s$, where Γ is a nonempty constructor context. More generally, the clauses

$$s_1 \approx \Gamma_1[s_2] \qquad s_2 \approx \Gamma_2[s_3] \qquad \cdots \qquad s_{n-1} \approx \Gamma_{n-1}[s_n] \qquad s_n \approx \Gamma_n[s_1]$$

entail a constraint $s_1 \approx \Gamma_1[\Gamma_2[\cdots[\Gamma_{n-1}[\Gamma_n[s_1]]]\cdots]]$. Moreover, the rule must support variables and nonunit clauses, and it should be finitely branching if we want to incorporate it in saturation-based provers—i.e., the set of clauses derivable from a given finite set of premises by a single rule should be finite. Finally, clauses of the form $\Gamma[x] \approx s \vee \mathcal{C}$, where x occurs in $\mathcal{C}$, are problematic, because there are infinitely many instantiations of x that can result in a cyclic constraint: s, $\mathsf{c}(s)$, $\mathsf{c}(\mathsf{c}(s))$, etc. To cope with all these subtleties, we first need to develop a considerable theoretical apparatus.

Definition 12. A *chain* built on a nonempty sequence of clauses $(\mathcal{C}_1, \ldots, \mathcal{C}_n)$ under condition $\mathcal{D}$ is a sequence $(t_1, \ldots, t_{n+1})$ of terms satisfying the following conditions:

1. for every $i \in [1,n]$, $\mathcal{C}_i$ is of the form $s_i \approx \Gamma_i[s'_{i+1}]_{p_i} \vee \mathcal{C}'_i$, where p_i is a nonempty constructor position in Γ_i;
2. there exists a substitution σ such that either σ is an mgu of $E = \{s'_i \stackrel{?}{=} s_i \mid i \in [2,n]\}$ or σ is an mgu of $\{s'_{n+1} \stackrel{?}{=} s_1\} \cup E$;
3. $t_i = s_i\sigma$ for $i \in [1,n]$ and $t_{n+1} = s'_{n+1}\sigma$;
4. $\mathcal{D} = \bigvee_{i=1}^{n} \mathcal{C}'_i\sigma$;
5. $\mathrm{type}(t_1) \sim \cdots \sim \mathrm{type}(t_{n+1})$;
6. $(\Gamma_i[s'_{i+1}]_{p_i} \approx s_i)\sigma$ is strictly maximal in $\mathcal{C}_i\sigma$, and no literal is selected in $\mathcal{C}_i\sigma$;
7. $s_i\sigma \not\succ \Gamma_i[s'_{i+1}]_{p_1}\sigma$, for $i \in [1,n]$;
8. for every $i \in [2,n]$, s'_i is not a variable.

The expression $\Gamma_1[\cdots[\Gamma_n[\bullet]_{p_n}]\cdots]_{p_1}\sigma$ is the chain's constructor context, σ is its mgu, and $p_1.\cdots.p_n$ is its constructor position. If $t_1 = t_{n+1}$, the sequence is called a *cycle*. A chain is *direct* if $t_i \neq t_j$ for all $i, j \in [1,n+1]$ with $i \neq j$ and $\{i,j\} \neq \{1,n+1\}$, and *variable-ended* if s'_{n+1} is a variable.

Remark 13. Conditions 5 to 8 are optional. They help prune the search space.

Definition 14. A chain $(t_1, \ldots, t_{n+1})$ built on a clause sequence $(\mathcal{C}_1, \ldots, \mathcal{C}_n)$ is an *extension* of an acyclic chain $(s_1, \ldots, s_{m+1})$ if $n \geq m$, the latter chain is built on $(\mathcal{C}_1, \ldots, \mathcal{C}_m)$, and the same literals and positions are considered in each clause $\mathcal{C}_i$ in both chains.

Since chains can be arbitrarily long, we need to impose some additional conditions to prune them and ensure that the rules are finitely branching. Let **Keep** be a property of chains that fulfills the following requirements:

(i) if a chain $\bar{t}$ does not satisfy **Keep**, no extension of $\bar{t}$ satisfies **Keep**;
(ii) for every finite clause set N, the set of chains built on a sequence of renamings of clauses in N and satisfying **Keep** is finite;
(iii) for every cycle $(t_1, \ldots, t_n, t_1)$, there exists a chain $(s_1, \ldots, s_m)$ with $m \leq n$ satisfying **Keep** such that for some k, the cycle $(t_{1+k}, \ldots, t_{n+k}, t_{1+k})$ (with $t_i := t_{i-n}$ if $i > n$) is an extension of $(s_1, \ldots, s_m)$.

For example, **Keep** can be defined as the set of chains built on clauses $\mathcal{C}_i$ that are pairwise distinct modulo renaming and such that $\mathcal{C}_1$ is the most recently processed clause. This is the definition we use in our description of the extended saturation loop (Section 6) and in the implementation in Vampire.

Remark 15. Condition (i) is essential in practice, to ensure that the chains can be incrementally constructed in an efficient way, because it ensures that the construction can be stopped when a prefix not satisfying **Keep** is obtained. Condition (ii) ensures that the rule is finitely branching. Condition (iii) is essential for completeness.

Definition 16. A chain of length n is *eligible* if it is variable-ended and $n = 1$, or if it is not variable-ended, it satisfies **Keep**, and either it is a cycle or there exists an extension of length $n+1$ that does not satisfy **Keep**.

Remark 17. The conditions on eligible chains are the strongest ones preserving completeness, but they are not necessary for soundness.

The acyclicity rule follows:

$$\frac{\mathcal{C}_1 \quad \cdots \quad \mathcal{C}_n}{\mathcal{D} \vee \mathcal{E}}\ \mathsf{Acycl}$$

if there exists a direct, eligible chain $(t_1,\ldots,t_{n+1})$ built on $(\mathcal{C}_1,\ldots,\mathcal{C}_n)$ under condition $\mathcal{D}$ and either $t_1 = t_{n+1}$ and $\mathcal{E} = \varnothing$ or $t_1 \neq t_{n+1}$ and $\mathcal{E} = \neg\mathsf{sub}(t_1,t_{n+1})$

Intuitively, the existence of the chain guarantees (if $\mathcal{D}$ is false) that there exists a nonempty constructor context $\Gamma[\bullet]_p$ such that $t_1 \approx \Gamma[t_{n+1}]_p$ holds. If $t_1 = t_{n+1}$, this contradicts acyclicity. Otherwise, we deduce that t_1 cannot occur at a constructor position inside the constructor term corresponding to t_{n+1}; hence $\mathsf{sub}(t_1,t_{n+1})$ is false.

Lemma 18 (Soundness of Acycl). *Let N be a clause set, and let $\mathcal{I}$ be a* sub*-minimal model of N satisfying* **Acy**. *If $\mathcal{C}$ is derived from N by* Acycl*, then $\mathcal{I} \models \mathcal{C}$.*

Uniqueness of Fixpoints. The uniqueness rule also depends on the notion of chain:

$$\frac{\mathcal{C}_1 \quad \cdots \quad \mathcal{C}_n}{\mathcal{D} \vee \left(\bigvee_{p\in P} u|_p \not\approx \mathsf{app}(s_p,t_1)\right) \vee u' \not\approx z \vee z \approx t_1}\ \mathsf{Uniq}$$

if there exists an eligible chain $(t_1,\ldots,t_{n+1})$ of constructor context $\Gamma[\bullet]_q$ built on $(\mathcal{C}_1,$ $\ldots,\mathcal{C}_n)$ under condition $\mathcal{D}$ and the following requirements are met:

1. $u = \Gamma[t_{n+1}]_q$;
2. P is the set of prefix-minimal positions p of some type $\tau \sim \mathrm{type}(t_1)$ in u with $p \not\leq q$;
3. for every $p \in P$, s_p is a fresh variable of the appropriate context type;
4. u' is obtained from u by replacing all terms at a position $p \in P$ by $\mathsf{app}(s_p,z)$.

Intuitively, the existence of the chain ensures (if $\mathcal{D}$ is false) that $t_1 \approx \Gamma[t_{n+1}]_q$. If $t_1 = t_{n+1}$, we could derive $y \not\approx \Gamma[y]_q \vee y \approx t_1$ by uniqueness. However, this would not be sufficient for completeness. First, t_1 may be distinct from t_{n+1}, but we may have $t_{n+1} = \Delta[t_1]_Q$, for some constructor context Δ, in which case we should derive $y \not\approx \Gamma[\Delta[y]_Q]_q \vee y \approx t_1$ instead. Second, t_1 may also occur at other positions in Γ. To capture all these cases using a finitely branching rule, we introduce new variables s_p whose purpose is to denote the context Γ_p such that $\Gamma_p[t_1] = u|_p$. (If t_1 does not occur inside $u|_p$, then Γ_p is constant.)

Example 19. From the clause $\mathsf{a} \approx \mathsf{c}(\mathsf{b},x)$, using the chain (a,x), with the constructor context $\mathsf{c}(\mathsf{b},\bullet)$, we derive

$$\mathsf{b} \not\approx \mathsf{app}(x_1,\mathsf{a}) \vee x \not\approx \mathsf{app}(x_2,\mathsf{a}) \vee z \not\approx \mathsf{c}(\mathsf{app}(x_1,z),\mathsf{app}(x_2,z)) \vee z \approx \mathsf{a}$$

Then $u = \mathsf{c}(\mathsf{b},x)$ and $P = \{1,2\}$.

From the clauses $\mathsf{a} \approx \mathsf{c}(\mathsf{b},\mathsf{a})$ and $\mathsf{b} \approx \mathsf{d}(\mathsf{a},\mathsf{a})$, using the chain $(\mathsf{a},\mathsf{b},\mathsf{a})$, with the constructor context $\mathsf{c}(\mathsf{d}(\mathsf{a},\bullet),\mathsf{a})$, we derive

$$\begin{aligned}&\mathsf{a} \not\approx \mathsf{app}(x_{1.1},\mathsf{a}) \vee \mathsf{a} \not\approx \mathsf{app}(x_{1.2},\mathsf{a}) \vee \mathsf{a} \not\approx \mathsf{app}(x_2,\mathsf{a})\\ &\vee z \not\approx \mathsf{c}(\mathsf{c}(\mathsf{app}(x_{1.1},z),\mathsf{app}(x_{1.2},z)),\mathsf{app}(x_2,z)) \vee z \approx \mathsf{a}\end{aligned}$$

In this case, $u = \mathsf{c}(\mathsf{d}(\mathsf{a},\mathsf{a}),x)$ and $P = \{1.1,1.2,2\}$.

Lemma 20 (Soundness of Uniq). *Let N be a clause set, and let $\mathcal{I}$ be a model of $N \cup \{\mathsf{App}, \mathsf{Hole}\}$ satisfying* **FP**. *If $\mathcal{C}$ is derived from N by* Uniq, *then $\mathcal{I} \models \mathcal{C}$.*

We also introduce the following optional simplification rule:

$$\frac{\Gamma_n[\cdots[\Gamma_1[s']_P]\cdots]_P \approx s \vee \mathcal{C}}{(\Gamma_1[s]_P \approx s \vee \mathcal{C})\sigma}\ \mathsf{Compr}$$

where s and s' are terms of the same type $\tau \in \mathcal{T}_{\mathsf{coind}}$ and P is a nonempty set of constructor positions in Γ_i, for $i \in [1,n]$, such that $\varepsilon \notin P$, and $\sigma = \mathrm{mgu}\{s \stackrel{?}{=} s', \Gamma_1 \stackrel{?}{=} \cdots \stackrel{?}{=} \Gamma_n\}$.

Proposition 21 (Soundness of Compr). *Let N be a clause set, and let $\mathcal{I}$ be a model of N satisfying* **FP**. *If $\mathcal{D}$ is derived from N by* Compr, *then $\mathcal{I} \models \mathcal{D}$.*

5 Refutational Completeness

We establish the refutational completeness of the calculus presented in Section 4. This result ensures that the axioms for distinctness, injectivity, and acyclicity (NSub) may be omitted. The axiom Uniq may also be omitted in some cases, formally defined below. The axiom Sub is still needed since it is used in the completeness proof for Acycl.

If $N \not\ni \square$ is a clause set saturated under $\mathcal{SP}$, then R_N denotes the set of rewrite rules constructed as usual from N and $\to_{R_N}$ denotes the (one-step) reduction relation. We refer to the literature [2, 16] for details about the construction of R_N. The notation $\mathcal{M}_N$ denotes the model of N defined by the congruence $\stackrel{*}{\leftrightarrow}_{R_N}$ on ground terms.

We first establish some results about the form of the rules in R_N.

Proposition 22. *Let N be a clause set saturated under $\mathcal{SP}$ and* Inf. *Let $u \approx v \vee \mathcal{C} \in N$, and let θ be a substitution such that $u\theta \succ v\theta$, $(u \approx v)\theta \succ \mathcal{C}\theta$, and $\mathcal{M}_N \not\models \mathcal{C}\theta$. If* $\mathrm{type}(u) \in \mathcal{T}_{\mathsf{ind}} \cup \mathcal{T}_{\mathsf{coind}}$, *then u is not a variable.*

Corollary 23. *Let N be a clause set saturated under $\mathcal{SP}$ and* Inf. *For every rule $\mathsf{c}(\bar{t}) \to_{R_N} s$ in R_N, where c is a constructor, s is R_N-irreducible.*

Lemma 24 (Infiniteness). *Let N be a clause set saturated under $\mathcal{SP}$ and* Inf. *If $\square \notin N$, then $\mathcal{M}_N$ satisfies* **Inf**.

Lemma 25 (Distinctness). *Let N be a clause set saturated under $\mathcal{SP}$,* Dist_1, Dist_2, *and* Inf. *For all ground terms $a = \mathsf{c}(\bar{a})$ and $b = \mathsf{d}(\bar{b})$ such that $\mathsf{c} \bowtie \mathsf{d}$, we have $a \stackrel{*}{\not\leftrightarrow}_{R_N} b$.*

Lemma 26 (Injectivity). *Let N be a clause set saturated under $\mathcal{SP}$,* Inf, Inj_1, *and* Inj_2. *For all ground terms $a = \mathsf{c}(a_1,\ldots,a_n)$ and $b = \mathsf{c}(b_1,\ldots,b_n)$ with $\mathsf{c} \in \mathcal{C}tr_{\mathsf{inj}}$ and such that $a_i \stackrel{*}{\not\leftrightarrow}_{R_N} b_i$ for some $i \in [1,n]$, we have $a \stackrel{*}{\not\leftrightarrow}_{R_N} b$.*

The completeness proof for acyclicity requires further definitions and results.

Definition 27. Let $\mathcal{I}$ be an interpretation and t be a term. A constructor context $\Gamma[\bullet]_p$ is a *minimal cyclicity witness* for t and $\mathcal{I}$ if it is of the same type as t, p is a position of the same type as t in Γ, $\mathcal{I} \models t \approx \Gamma[t]_p$, and $|q| \geq |p|$ for every position $q \neq \varepsilon$ and constructor context $\Delta[\bullet]_q$ such that $\mathcal{I} \models t \approx \Delta[t]_q$.

Proposition 28. *Let $(t_1,\dots,t_n,t_1)$ be a cycle of constructor context $\Gamma[\bullet]_p$ for a clause set N under condition $\mathcal{D}$. If $\mathcal{I} \models N \cup \{\neg \mathcal{D}\sigma\}$, and $\Gamma[\bullet]_p$ is a minimal cyclicity witness for $t_1\sigma$ and $\mathcal{I}$, then $(t_1,\dots,t_n,t_1)$ is direct.*

Lemma 29. *Let $t : \tau$ and $s : \upsilon$ be ground terms with $\tau \sim \upsilon$. Let $\Gamma[\bullet]_p$ be a ground constructor context of type τ, where p is a position of type υ in Γ. Let N be a clause set saturated under $\mathcal{SP}$ and* Inf. *Assume that t, s, and $\Gamma|_{p'}$ are R_N-irreducible, for every position $p' \not\leq p$. If $\mathcal{M}_N \models \Gamma[s]_p \approx t$, then R_N contains n rules $\Gamma_i[a_{i+1}]_{p_i} \to_{R_N} a_i$, for $i \in [1,n]$, with $\Gamma[s]_p = \Gamma_0[\Gamma_1[\cdots[\Gamma_n[a_{n+1}]_{p_n}]\cdots]_{p_1}]_{p_0}$, $p_0.p_1.\cdots.p_n = p$, $a_{n+1} = s$, and $t = \Gamma_0[a_1]_{p_0}$.*

Lemma 30 (Acyclicity). *If* Sub $\subseteq N$ *and* $N \not\ni \square$ *is saturated under $\mathcal{SP}$,* Acycl*, and* Inf*, then $\mathcal{M}_N$ satisfies condition* **Acy**.

Remark 31. The Inf rule is needed for completeness. For example, it is clear that the clause $x \approx \mathsf{a} \vee x \approx \mathsf{b}$ contradicts acyclicity, but no contradiction can be derived without using Inf. The relaxation of the application conditions of Sup is also essential. Consider the set $N = \{\mathsf{a}_1 \approx \mathsf{c}(\mathsf{a}_2), \mathsf{a}_2 \approx \mathsf{a}_3, \mathsf{a}_3 \approx \mathsf{c}(\mathsf{a}_1)\}$, with $\mathsf{c}(\dots) \succ \mathsf{a}_{i+1} \succ \mathsf{a}_i$. It is clear that N is saturated without the relaxation, and N contradicts acyclicity, since $N \models \mathsf{a}_1 \approx \mathsf{c}(\mathsf{c}(\mathsf{a}_1))$. With the relaxation, Sup derives the clause $\mathsf{a}_2 \approx \mathsf{c}(\mathsf{a}_1)$; then Acycl exploits the cycle $(\mathsf{a}_1, \mathsf{a}_2, \mathsf{a}_1)$ to derive $\square$.

For the Uniq rule, we provide a restricted completeness result, under the assumption that the considered constructor context contains at most one occurrence of $\bullet$.

Lemma 32 (Uniqueness of Fixpoints). *If* App $\subseteq N$ *and* $N \not\ni \square$ *is saturated under $\mathcal{SP}$,* Uniq*, and* Inf*, then $\mathcal{M}_N \models x \approx \Gamma[x]_r \wedge y \approx \Gamma[y]_r \Rightarrow x \approx y$ for every constructor context of the form $\Gamma[\bullet]_r$ of type $\tau \in \mathcal{T}_{\mathsf{coind}}$, where r is a nonempty position of type τ in Γ.*

Definition 33. A signature is *coinductively nonbranching* if for every constructor $\mathsf{c} : \tau_1 \times \cdots \times \tau_n \to \tau$ such that $\tau \in \mathcal{T}_{\mathsf{coind}}$, there exists at most one $i \in [1,n]$ such that $\tau_i \sim \tau$.

For example, the signature is coinductively nonbranching for infinite streams and possibly infinite lists, but not for infinite binary trees.

Corollary 34 (Fixpoints). *Assume that the signature is coinductively nonbranching. If* Cycl $\cup$ App $\subseteq N$ *and* $N \not\ni \square$ *is saturated under $\mathcal{SP}$,* Uniq*, and* Inf*, then $\mathcal{M}_N$ satisfies condition* **FP**.

Example 35. Corollary 34 does not hold for arbitrary signatures. The clause set $\{\mathsf{a} \approx \mathsf{c}(\mathsf{d}(\mathsf{a},\mathsf{b})), \mathsf{b} \approx \mathsf{e}(\mathsf{d}(\mathsf{a},\mathsf{b})), \mathsf{a}' \approx \mathsf{c}(\mathsf{d}(\mathsf{a}',\mathsf{b}')), \mathsf{b}' \approx \mathsf{e}(\mathsf{d}(\mathsf{a}',\mathsf{b}')), \mathsf{d}(\mathsf{a},\mathsf{b}) \not\approx \mathsf{d}(\mathsf{a}',\mathsf{b}')\}$ contradicts **FP**, because $\mathsf{d}(\mathsf{a},\mathsf{b})$ and $\mathsf{d}(\mathsf{a}',\mathsf{b}')$ are both solutions of $x \approx \mathsf{d}(\mathsf{c}(x),\mathsf{e}(x))$. However, the Uniq rule applies only with constructor contexts of head symbol c (if the chain starts with a or a') or e (if it starts with b or b').

6 Saturation Procedure

The inference rules of the calculus presented in Section 4 are all finitely branching, provided that the eligibility criterion is applied for the Acycl and Uniq rules. As a result, saturation of a clause set can be carried out using standard saturation procedures. These generally work by maintaining a set of *passive clauses* that initially contains all the

clauses to saturate and a set of *active clauses* that is initially empty. The algorithm heuristically chooses a passive clause that becomes the *given clause*, moves it to the active clauses, and performs all possible inferences between it and the active clauses. Conclusions are added to the set of passive clauses, and the procedure is iterated until $\square$ is derived, or until the set of passive clauses is empty.

To improve search, it is useful to distinguish between *simplifying rules* and *generating rules*. In simplifying rules, at least one of the premises is redundant with respect to the conclusion. The Inf rule is simplifying, as well as the Dist_1 and Inj_1 rules when the term t is not a variable, and the Acycl rule when there is only one premise and $t_1 = t_n$.

In addition to the calculus, we propose the following simplifying rules to eliminate theory tautologies:

$$\frac{\mathsf{c}(\bar{s}) \not\approx \mathsf{d}(\bar{t}) \vee \mathcal{C}}{\varnothing}\ \mathsf{Dist}^- \qquad\qquad \frac{s \not\approx \Gamma[s] \vee \mathcal{C}}{\varnothing}\ \mathsf{Acycl}^-$$

where $\mathsf{c} \bowtie \mathsf{d}$, $\Gamma[\bullet]$ is a nonempty constructor context, and $\mathrm{type}(s) \in \mathcal{T}_{\mathsf{ind}}$. Moreover, the following rule applies injectivity of $\mathsf{c} \in \mathcal{C}tr_{\mathsf{inj}}$ to simplify literals:

$$\frac{\mathsf{c}(s_1,\ldots,s_n) \not\approx \mathsf{c}(t_1,\ldots,t_n) \vee \mathcal{C}}{\left(\bigvee_{i=1}^{n} s_i \not\approx t_i\right) \vee \mathcal{C}}\ \mathsf{Inj}^-$$

The soundness of Inj^- follows from c's being a function symbol, but since it is also injective, the premise is redundant with respect to the theory. We conjecture that the addition of these simplification rules preserves refutational completeness.

If all constructors are free (i.e., $\mathcal{C}tr_{\mathsf{inj}} = \mathcal{C}tr$ and $\mathsf{c} \bowtie \mathsf{d}$ holds for all distinct constructors c and d), by applying the above rules eagerly, we also guarantee that in any literal $[\neg]s \approx t$ in an active clause, at most one of s or t has a constructor for head symbol, as (dis)equalities between constructor terms will have been simplified directly after clause generation. This invariant enables a few optimizations in the implementation of the generating rules, notably during the detection of chains.

The relaxation of the application conditions of the Sup rule increases the number of clauses it must generate and may hence be detrimental to the search. We can reduce the incidence of this scenario by choosing a term order that considers constructors as smaller than non-constructors. For path orders, we can choose a symbol precedence $\succ$ such that $\mathsf{f} \succ \mathsf{c}$ for all non-constructor symbols f and constructors c.

To implement the Acycl and Uniq rules, we must be able to efficiently detect eligible chains among the set of active clauses. Testing all subsets of the active clauses is impractical, and the detection of a chain requires the computation of an mgu over a set of equations, instead of a single equation. We present a procedure that takes the given clause $\mathcal{C}_1$ as input and applies the two rules to all subsets of clauses containing $\mathcal{C}_1$ and upon which an eligible chain can be built. There are three cases in which the rules must be applied: when the chain is a cycle, when it is variable-ended and has length 1, and when there exists an extension of the chain that violates **Keep**. The procedure relies on a data structure that provides a $\mathsf{nextLinks}(s')$ operation, where s' is a term. For each literal $s \approx t$ in an active clause $\mathcal{C}$ such that s is unifiable with s' under an mgu σ and $s\sigma \not\succeq t\sigma$, the operation returns the tuple $(\mathcal{C}, \sigma, T)$, where T is the set of terms under

nonempty constructor positions in t. This operation can be implemented using term indexing techniques already found in state-of-the-art provers [22, Section 5.1].

The procedure considerGiven($\mathcal{C}_1$) applies the rule Acycl or Uniq to all subsets of actives clauses that contain the given clause $\mathcal{C}_1$ and form an eligible chain:

Procedure considerGiven($\mathcal{C}_1$) **is**
  for s_2' **such that** $\mathcal{C}_1 = s_1 \approx \Gamma[s_2'] \vee \mathcal{D}_1$ **do**
    extendChain($s_1, s_2', \{\}, \{\mathcal{C}_1\}$)

Procedure extendChain(s_1, s_i', θ, Ch) **is**
  if $s_1\theta = s_i'\theta$ **then**
    apply rule Acycl or Uniq to chain Ch under mgu θ
  else if s_i' is a variable **then**
    if $|Ch| = 1$ **then**
      apply rule Acycl or Uniq to chain Ch under mgu θ
  else if exists $(\mathcal{C}_i, \sigma, T) \in$ nextLinks($s_i'\theta$) **such that** $\mathcal{C}_i \in Ch$ **then**
    apply rule Acycl or Uniq to chain Ch under mgu θ
  else
    for $(\mathcal{C}_i, \sigma, T) \in$ nextLinks($s_i'\theta$) **do**
      for $s_{i+1}' \in T$ **do**
        extendChain($s_1, s_{i+1}', \sigma\theta, Ch \uplus \{\mathcal{C}_i\}$)

7 Evaluation

We implemented the calculus presented above in the first-order theorem prover Vampire [13]. Our source code is publicly available.[1] The new rules are added to the existing calculus, which includes other sound rules and a sophisticated redundancy elimination mechanism. Vampire can process input files in SMT-LIB [4] format and recognizes both the `declare-datatypes` command and the nonstandard `declare-codatatypes` command. These commands trigger the addition of relevant axioms or the activation of inference rules, according to user-specified options. This implementation is an extension of previous work done in Vampire [12]. The behavior of this older implementation can be replicated by enabling only the simplification rules of the calculus and adding the axioms Dist, Inj, Exhaust, Sub, and NSub to the initial clause set.

We evaluated the implementation on 4170 problems that were used previously by Reynolds and Blanchette [20] to evaluate CVC4. These were generated by translating Isabelle problems to SMT-LIB using the Sledgehammer bridge [18]. We also used synthetic problems that exercise the properties of cyclic values. Both benchmark sets and detailed results are available online.[2]

All the experiments in this section were carried out on a cluster on which each node is equipped with two quad-core Intel processors running at 2.4 GHz, with 24 GiB of memory. A 60 s time limit per problem was enforced. We used a single basic saturation strategy relying on the DISCOUNT saturation algorithm. The calculus was parameterized by a Knuth–Bendix term order, unless otherwise noted. This simple approach

[1] `http://github.com/vprover/vampire/releases/tag/ijcar2018-data`

[2] `http://matryoshka.gforge.inria.fr/pubs/supdata_data.tar.gz`

provides a homogeneous basis on which to compare the performance of the different procedures. It typically solves fewer problems than the portfolio approach commonly used with Vampire, in which several different strategies are tried in short time slices.

We first compare the performance of three configurations of the prover on the Isabelle problems. The first configuration corresponds to the axiomatic approach presented in Section 3: the axioms Dist, Inj, Exhaust, Sub, NSub, App, Uniq, Cycl, and Hole are added to the set of clauses to saturate, and only standard inferences rules are used by the prover. Superposition need not rewrite the nonmaximal side of an equation.

The second configuration implements part of the calculus presented in Section 4. Only the axioms Exhaust, Sub, NSub, App, Uniq, Cycl, and Hole are added to the clauses, and the rules Dist_1, Dist_2, Inj_1, and Inj_2 are used during the search, in addition to the simplification rules described in Section 6. The side conditions of Sup are also relaxed. The rules Acycl and Uniq are not used; instead, reasoning on the properties of cyclic terms is based on axioms.

The third configuration uses all the rules described in Section 4. Only the axioms Sub and App are added, on which the Acycl and Uniq rules depend, and the axioms Cycl and Exhaust. This configuration is the only one which does not ensure refutational completeness, since Uniq is incomplete with respect to the uniqueness of fixpoints for branching codatatypes.

The first two configurations both solved 1114 problems and the third one solved 1113 problems; 1116 problems are solved by at least one configuration. These homogeneous results do not reveal significant differences between the approaches. To assess the role of the acyclicity property of datatypes and the properties of codatatype fixpoints in the benchmarks, we also tested a system that did not include any axioms and rules related to these properties. With such an incomplete system, we found that 12 problems could not be solved. This is roughly in line with the results of Reynolds and Blanchette using CVC4 on the same problems [20]. No new problems were solved by this configuration, suggesting that reasoning about properties of cyclic terms does not lead to worse performance even when these properties are not needed for refutation.

We also tested variants of the last two configurations in which the calculus was parameterized by a lexicographic path order, to assess whether this term order could improve the performance when used with the relaxed superposition rule. These configurations solved a total of 1104 problems, including 5 new problems.

Since properties of cyclic values are seldom used in the Isabelle benchmarks, we crafted a set of (refutable) problems to assess the performance of the rules Acycl and Uniq. For a term s and a nonconstant context $\Gamma[\bullet]$, let $\mathsf{exchain}(s,\Gamma[\bullet])$ denote any sentence $\exists s_2, \ldots s_n \forall t_1, \ldots t_m.\ s \approx \Gamma_1[s_2] \wedge \cdots \wedge s_n \approx \Gamma_n[s]$, where $t_1, \ldots, t_m$ all occur in Γ and such that $\Gamma_1[\ldots[\Gamma_n[\bullet]]\ldots] = \Gamma[\bullet]$. The formula $\exists s.\, \mathsf{exchain}(s,\Gamma[\bullet])$, where $\mathrm{type}(s) \in \mathcal{T}_{\mathsf{ind}}$, forms an acyclicity problem. The set of acyclicity problems used in our experiments is denoted AC. If $m = 0$, the clausified form of this problem is ground (ACG). The formula $\exists s_1, s_2.\, \mathsf{exchain}(s_1,\Gamma[\bullet]) \wedge \mathsf{exchain}(s_2,\Gamma[\bullet]) \wedge s_1 \not\approx s_2$, where $\mathrm{type}(s_1) \in \mathcal{T}_{\mathsf{coind}}$, forms a uniqueness problem (U). Note that in such a problem, the two chains may not be formed upon the same equalities, although they build the same constructor context. Similarly, if $m = 0$, we obtain a ground uniqueness problem (UG). Finally, the sentence $\forall s.\ \neg\, \mathsf{exchain}(s,\Gamma[\bullet])$, for $\mathrm{type}(s) \in \mathcal{T}_{\mathsf{coind}}$, forms an existence problem (EX).

We generated 100 instances of each type of problem. The number of problems solved by Vampire (V) on these problems are presented in the following table, along with the results obtained using CVC4's [3] and Z3's [15] native support for datatypes and, in CVC4's case, for codatatypes:

	AC			ACG			U		UG		EX	
	V	CVC4	Z3	V	CVC4	Z3	V	CVC4	V	CVC4	V	CVC4
Axioms	65	–	–	**100**	–	–	**14**	–	10	–	**40**	–
Calculus	82	**100**	59	**100**	**100**	**100**	**14**	12	13	**100**	35	0

The number of problems solved shows that the Acycl rule performs better than the axioms for acyclicity problems with variables. Only one of these problems could be solved by the axiomatic approach and not by the Acycl rule. Both approaches managed to solve all of the ground acyclicity problems. Z3 solved all of the ground problems, performing slightly less well on those featuring universal quantifiers. CVC4 was able to solve all of the acyclicity problems, including those with universal quantifiers, a notable improvement over previous results obtained on similar problems [22, Section 6].

On uniqueness problems, the Uniq rule solved a superset of the ground problems solved by the axiomatic approach, whereas on nonground problems each approach uniquely solved 3 problems, for a total of 17 problems solved. Again, CVC4 performed remarkably well on ground problems, while the presence of variables in the problem led to a marked degradation of its performance. Finally, for existence problems, the refutation relies mostly on the Cycl axiom, which is included in the clause set in both Vampire configurations. Yet, the purely axiomatic approach was able to solve 6 problems that could not be solved when the Uniq rule was activated, indicating that the rule might lead the search in a suboptimal direction. The theory solver in CVC4 does not take into account the existence of fixpoints for codatatypes, which is a nonground property. Consequently, none of the existence problems were solved by CVC4.

From the results, it appears that the calculus supersedes the axiomatic approach for problems with datatypes. For codatatypes, both approaches solve different problems, suggesting that they should both be included in a strategy portfolio. However, the conceptual simplicity and easy implementation of the axiomatic approach may outweigh these differences in performance.

8 Related Work

The potential of (co)datatypes for automated reasoning has been studied mostly in the context of satisfiability modulo theories (SMT). Datatypes are parts of the SMT-LIB 2.6 standard [4]. They were implemented in CVC3 by Barrett et al. [5], in Z3 [15] by de Moura, and in CVC4 by Reynolds and Blanchette [20]. The CVC4 work also includes a decision procedure for the ground theory of codatatypes. Moreover, CVC4 supports automatic structural induction [21] and dedicated reasoning support for selectors.

Structural induction has also been added to superposition by Kersani and Peltier [11], Cruanes [10], and Wand [24]. In unpublished work, Wand implemented incomplete inference rules for datatypes, including acyclicity, in his superposition prover Pirate. Robillard's earlier Acycl rule [22] has inspired our Acycl rule, but it suffered

from many forms of incompleteness. For example, given the unsatisfiable clause set $\{\mathsf{a} \approx \mathsf{c}(x) \vee \mathsf{p}(x), \neg\mathsf{p}(\mathsf{c}(\mathsf{a}))\}$, the old Acycl rule derived only $\mathsf{p}(\mathsf{a})$ before reaching saturation. Another issue concerned cycles built from multiple copies of the same premise.

In the context of program verification, Bjørner [6] introduced a decision procedure for (co)datatypes in STeP, the Stanford Temporal Prover. The program verification tool Dafny provides both a syntax for defining (co)datatypes and some support for automatic (co)induction proofs [14]. Other verification tools such as Leon [23] and RADA [19] also include (semi-)decision procedures for datatypes. We refer to Barrett et al. [5] and Reynolds and Blanchette [20] for further discussions of related work.

9 Conclusion

We presented two approaches to reason about datatypes and codatatypes in first-order logic: an axiomatization and an extension of the superposition calculus. We established completeness results about both. We also showed how to integrate the new inference rules in a saturation prover's main loop and implemented them in the Vampire prover. The empirical results look promising, although it is not clear from our benchmarks how often the most difficult properties—acyclicity for datatypes, existence and uniqueness of fixpoints for codatatypes—are useful in practice.

This work is part of a wider research program that aims at bridging the gap between automatic theorem provers and their applications to program verification and interactive theorem proving. In future work, we want to reconstruct the new proof rules in Isabelle, to make it possible to enable datatype reasoning in Sledgehammer. We also believe that further tuning and evaluations could help improve the calculus and the heuristics.

Acknowledgment. We thank Alexander Bentkamp, Simon Cruanes, Uwe Waldmann, Daniel Wand, and Christoph Weidenbach for fruitful discussions that led to this work. We also thank Mark Summerfield and the anonymous reviewers for suggesting textual improvements.

Blanchette has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No. 713999, Matryoshka). Robillard has received funding from the ERC Starting Grant 2014 SYMCAR 639270, the Wallenberg Academy Fellowship 2014 TheProSE, the Swedish Research Council grant GenPro D0497701, and the Austrian FWF research project RiSE S11409-N23.

References

[1] Bachmair, L., Dershowitz, N., Hsiang, J.: Orderings for equational proofs. In: LICS '86. pp. 346–357. IEEE Computer Society (1986)

[2] Bachmair, L., Ganzinger, H.: Rewrite-based equational theorem proving with selection and simplification. J. Log. Comput. 4(3), 217–247 (1994)

[3] Barrett, C., Conway, C.L., Deters, M., Hadarean, L., Jovanović, D., King, T., Reynolds, A., Tinelli, C.: CVC4. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 171–177. Springer (2011)

[4] Barrett, C., Fontaine, P., Tinelli, C.: The SMT-LIB standard: Version 2.7. Tech. rep., University of Iowa (2017), http://smt-lib.org/

[5] Barrett, C., Shikanian, I., Tinelli, C.: An abstract decision procedure for satisfiability in the theory of inductive data types. J. Satisf. Boolean Model. Comput. 3, 21–46 (2007)
[6] Bjørner, N.S.: Integrating Decision Procedures for Temporal Verification. Ph.D. thesis, Stanford University (1998)
[7] Blanchette, J.C., Hölzl, J., Lochbihler, A., Panny, L., Popescu, A., Traytel, D.: Truly modular (co)datatypes for Isabelle/HOL. In: Klein, G., Gamboa, R. (eds.) ITP 2014. LNCS, vol. 8558, pp. 93–110. Springer (2014)
[8] Blanchette, J.C., Peltier, N., Robillard, S.: Superposition with datatypes and codatatypes. Technical report (2018), http://matryoshka.gforge.inria.fr/pubs/supdata_report.pdf
[9] Comon, H., Lescanne, P.: Equational problems and disunification. J. Symb. Comput 7(3–4), 371–425 (1989)
[10] Cruanes, S.: Superposition with structural induction. In: Dixon, C., Finger, M. (eds.) FroCoS 2017. LNCS, vol. 10483, pp. 172–188. Springer (2017)
[11] Kersani, A., Peltier, N.: Combining superposition and induction: A practical realization. In: Fontaine, P., Ringeissen, C., Schmidt, R.A. (eds.) FroCoS 2013. LNCS, vol. 8152, pp. 7–22. Springer (2013)
[12] Kovács, L., Robillard, S., Voronkov, A.: Coming to terms with quantified reasoning. In: Castagna, G., Gordon, A.D. (eds.) POPL 2017. pp. 260–270. ACM (2017)
[13] Kovács, L., Voronkov, A.: First-order theorem proving and Vampire. In: Sharygina, N., Veith, H. (eds.) Computer Aided Verification (CAV 2013). LNCS, vol. 8044, pp. 1–35. Springer (2013)
[14] Leino, K.R.M., Moskal, M.: Co-induction simply—Automatic co-inductive proofs in a program verifier. In: Jones, C.B., Pihlajasaari, P., Sun, J. (eds.) FM 2014. LNCS, vol. 8442, pp. 382–398. Springer (2014)
[15] de Moura, L., Bjørner, N.: Z3: An efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer (2008)
[16] Nieuwenhuis, R., Rubio, A.: Paramodulation-based theorem proving. In: Robinson, J.A., Voronkov, A. (eds.) Handbook of Automated Reasoning, vol. I, pp. 371–443. Elsevier and MIT Press (2001)
[17] Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL: A Proof Assistant for Higher-Order Logic, LNCS, vol. 2283. Springer (2002)
[18] Paulson, L.C., Blanchette, J.C.: Three years of experience with Sledgehammer, a practical link between automatic and interactive theorem provers. In: Sutcliffe, G., Schulz, S., Ternovska, E. (eds.) IWIL-2010. EPiC, vol. 2, pp. 1–11. EasyChair (2012)
[19] Pham, T., Whalen, M.W.: RADA: A tool for reasoning about algebraic data types with abstractions. In: Meyer, B., Baresi, L., Mezini, M. (eds.) ESEC/FSE '13. pp. 611–614. ACM (2013)
[20] Reynolds, A., Blanchette, J.C.: A decision procedure for (co)datatypes in SMT solvers. J. Autom. Reason. 58(3), 341–362 (2017)
[21] Reynolds, A., Kuncak, V.: Induction for SMT solvers. In: D'Souza, D., Lal, A., Larsen, K.G. (eds.) VMCAI 2015. LNCS, vol. 8931, pp. 80–98. Springer (2014)
[22] Robillard, S.: An inference rule for the acyclicity property of term algebras. In: Kovács, L., Voronkov, A. (eds.) Vampire 2017. EPiC, EasyChair, to appear
[23] Suter, P., Köksal, A.S., Kuncak, V.: Satisfiability modulo recursive programs. In: Yahav, E. (ed.) SAS 2011. LNCS, vol. 6887, pp. 298–315. Springer (2011)
[24] Wand, D.: Superposition: Types and Polymorphism. Ph.D. thesis, Universität des Saarlandes (2017)