Datatypes with Shared Selectors (Technical Report)

Andrew Reynolds[1], Arjun Viswanathan[1], Haniel Barbosa[1], Cesare Tinelli[1], and Clark Barrett[2]

[1] University of Iowa, Iowa City, USA
[2] Department of Computer Science, Stanford University

Abstract. We introduce a new theory of algebraic datatypes where selector symbols can be shared between multiple constructors, thereby reducing the number of terms considered by current SMT-based solving approaches. We show that the satisfiability problem for the traditional theory of algebraic datatypes can be reduced to problems where selectors are mapped to shared symbols based on a transformation provided in this paper. The use of shared selectors addresses a key bottleneck for an SMT-based enumerative approach to the Syntax-Guided Synthesis (SyGuS) problem. Our experimental evaluation of an implementation of the new theory in the SMT solver cvc4 on syntax-guided synthesis and other domains provides evidence that the use of shared selectors improves state-of-the-art SMT-based approaches for constraints over algebraic datatypes.

1 Introduction

Algebraic datatypes, also known as inductive or recursive datatypes, are composite types commonly used for expressing finite data structures in computer science applications, such as lists or trees. Reasoning efficiently about (algebraic) datatypes is thus paramount in such fields as program analysis and verification, which has led to numerous approaches for automating solving in this setting. In this paper, we follow the semantic approach introduced by Barrett et al. [10], which is generally the basis for datatype decision procedures in satisfiability modulo theories (SMT) solvers [11].

In semantic presentations of the theory of algebraic datatypes [10, 20], a datatype is an absolutely free algebra over a signature of function symbols called *constructors*; the immediate subterms of a datatype value are accessed with function symbols called *selectors*, or *projections*, which are specific for each constructor and its arguments. Datatypes also have *discriminators*, or *testers*, associated with each constructor. They are predicates indicating whether a given datatype value was built with a specific constructor.

The satisfiability of quantifier-free formulas in the theory of algebraic datatypes is decidable. A basic decision procedure for this problem [10, 20] used by a number of SMT solvers operates by *progressively unrolling* datatypes: it tries to satisfy constraints by guessing top-level constructors in order to build values for the constraint variables incrementally. Concretely, if x is a datatype variable and c is an n-ary constructor for the datatype, the procedure may guess the equality constraint $x \approx c(x_1, \ldots, x_n)$ where $x_1, \ldots, x_n$ are fresh variables. If such a choice leads to an inconsistency, the procedure backtracks and tries different constructors until it determines that the constraints are satisfiable or no more choices are possible. During this process, lemmas in the form of quantifier-free clauses may be learned by the procedure that prevent the procedure from making guesses already shown to be infeasible. However, these lemmas may include selectors, and because each selector is associated with only a single constructor, the generality and hence the usefulness of such lemmas is limited.

To address this limitation we introduce a new (formulation of the) theory of datatypes that allows certain selectors to be shared by multiple constructors. This way, information previously acquired when reasoning with a constructor, i.e., the learned lemmas on the applications of its selectors, can be reused when an argument of the same type is considered in another constructor. We illustrate this point with the following examplewhich will be used as a running example throughout the paper.

Example 1. Consider a binary tree whose internal nodes store either one or two integer values, and whose leaves store both a Boolean and an integer value. A datatype **Tree** modeling this data structure has three constructors: one (N_1) taking an integer and two **Tree** elements as arguments, another (N_2) taking two integers and two **Tree** elements as arguments, and a third (L) taking as arguments a Boolean and an integer element. We write this datatype in the following BNF-style notation:

$$\textbf{Tree} = \mathsf{N}_1(\textbf{Int}, \textbf{Tree}, \textbf{Tree}) \mid \mathsf{N}_2(\textbf{Int}, \textbf{Int}, \textbf{Tree}, \textbf{Tree}) \mid \mathsf{L}(\textbf{Bool}, \textbf{Int})$$

We assume each constructor has selectors associated with them. The *subfields* (i.e., the immediate subterms) of terms constructed by N_1 are accessed, respectively, by the selectors $\mathsf{S}^{\mathsf{N}_1,1}$, $\mathsf{S}^{\mathsf{N}_1,2}$, and $\mathsf{S}^{\mathsf{N}_1,3}$ of type $\textbf{Tree} \to \textbf{Int}$, $\textbf{Tree} \to \textbf{Tree}$ and $\textbf{Tree} \to \textbf{Tree}$. The selectors for the other constructors are similar. The subfields of terms constructed by N_2 are accessed by the selectors $\mathsf{S}^{\mathsf{N}_2,1}$, $\mathsf{S}^{\mathsf{N}_2,2}$, $\mathsf{S}^{\mathsf{N}_2,3}$, and $\mathsf{S}^{\mathsf{N}_2,4}$; the subfields of L by $\mathsf{S}^{\mathsf{L},1}$ and $\mathsf{S}^{\mathsf{L},2}$. We also assume each constructor is associated with a tester predicate, i.e. isN_1, isN_2, and isL, each of which takes a **Tree** as an argument. Given term t of type **Tree**, consider the following set of clauses:

$$\{ \neg\mathsf{isN}_1(t) \vee \mathsf{S}^{\mathsf{N}_1,1}(t) \geq 0,\ \neg\mathsf{isL}(t) \vee \mathsf{S}^{\mathsf{L},2}(t) \geq 0 \} \tag{1}$$

The first clause states that when t has top symbol N_1, its first subfield (which is of type **Int**) is non-negative. Similarly, the second says that when t has top symbol L, its second subfield is non-negative.

Consider now a different kind of selector symbol $\mathsf{S}^{\textbf{Int},1}$ of type $\textbf{Tree} \to \textbf{Int}$ which maps each value of type **Tree** to the first (i.e., leftmost) subfield of t of type **Int**, *regardless* of the top constructor symbol of t. We will refer to such selectors as *shared selectors*. While nine selectors in the standard sense are necessary for **Tree**, five shared selectors suffice to access all possible subfields of a value of type **Tree**: two to access the **Tree** subfields, two to access the **Int** subfields, and one to access the **Bool** subfield of L. In particular, clause set (1) can be rewritten as follows using only one shared selector:

$$\{ \neg\mathsf{isN}_1(t) \vee \mathsf{S}^{\textbf{Int},1}(t) \geq 0,\ \neg\mathsf{isL}(t) \vee \mathsf{S}^{\textbf{Int},1}(t) \geq 0 \} \tag{2}$$

stating that when t has top symbol N_1 or L, its first integer child is non-negative. •

In Example 1, the second set of clauses has one unique arithmetic constraint whereas the first set has two. In practice, reducing the number of unique constraints can substantially improve the performance of SMT solvers. Our experiments show that shared selectors lead to a significant reduction in the number of unique constraints for several classes of benchmarks from real applications, with resulting SMT solver performance improvements that are proportional to the magnitude of this reduction.

Contributions We introduce a conservative extension of the (generic) theory of algebraic datatypes that features shared selectors. We show how using shared selectors instead of standard (unshared) selectors can improve the performance of current satisfiability procedures for the theory and also, as a result, the performance of procedures for syntax-guided synthesis. Specifically:

1. We formalize the new theory and show that constraints in the original signature can be reduced to equisatisfiable constraints whose selectors are all shared selectors. We present a decision procedure for the satisfiability of quantifier-free formulas in this theory as a natural modification of an earlier procedure for datatypes [20].
2. We provide details on an SMT-based approach for syntax-guided synthesis [22], and demonstrate how it can significantly benefit from native support in the SMT solver for a theory of datatypes with shared selectors.
3. We present an extensive experimental evaluation of our implementation in the SMT solver cvc4 [7] on benchmarks from SMT-LIB [8] and from the most recent edition of SyGuS-COMP [3], the syntax-guided synthesis competition. This evaluation shows that shared selectors can reduce the number of terms introduced during solving, thus leading to more solved problems with respect to the state of the art.

Outline After preliminaries in Section 2, we formalize the new theory of datatypes in Section 3. In Section 3.2, we define a satisfiability-preserving transformation between (datatype) constraints containing only standard selectors and constraints containing only shared selectors. In Section 4, we present a decision procedure for satisfiability in the theory based on a preliminary elimination of standard selectors. Section 5 introduces syntax-guided synthesis and explains how this application benefits from datatypes with shared selectors. Section 6 presents our experimental evaluation. We cover related work in Section 7 and offer concluding remarks in Section 8.

2 Preliminaries

Our setting is a many-sorted classical first-order logic similar in essence to the one adopted by the SMT-LIB standard [9]. A signature $\Sigma = (\mathcal{Y}, \mathcal{F})$ consists of a set $\mathcal{Y}$ of first-order types, or *sorts*, and a set $\mathcal{F}$ of first-order function symbols over these types. Each symbol $\mathsf{f} \in \mathcal{F}$ is associated with a list $\tau_1, \ldots, \tau_n$ of argument types and a return type τ, written $\mathsf{f} : \tau_1 \times \cdots \times \tau_n \to \tau$ or just $\mathsf{f} : \tau$ if $n = 0$. The function $\mathsf{arity}(\mathsf{f})$ returns n. We assume that any signature contains a $\mathbf{Bool}$ type and constants $\mathsf{true}, \mathsf{false} : \mathbf{Bool}$; a family $(\approx : \tau \times \tau \to \mathbf{Bool})_{\tau \in \mathcal{Y}}$ of equality symbols; a family $(\mathsf{ite} : \mathbf{Bool} \times \tau \times \tau \to \tau)_{\tau \in \mathcal{Y}}$ of *if-then-else* symbols; and the Boolean connectives $\neg$, $\wedge$, $\vee$ with their expected types. Function symbols of $\mathbf{Bool}$ return type play the role of predicate symbols.

Typed terms are built as usual over function symbols from $\mathcal{F}$ and typed variables from a fixed family $(\mathcal{V}_\tau)_{\tau \in \mathcal{Y}}$ of pairwise-disjoint infinite sets. Formulas are terms of type $\mathbf{Bool}$. Equivalence is equality ($\approx$) on $\mathbf{Bool}$. The syntax $t \not\approx u$ is short for $\neg(t \approx u)$.

We reserve the names $\mathsf{a}, \mathsf{c}, \mathsf{f}, \mathsf{g}, \mathsf{p}, \mathsf{q}$ for function symbols; x, y, z for variables; r, s, t, u for terms (which may be formulas); and φ, ψ for formulas. We use the symbol = for equality at the meta-level. We write t^τ to indicate that t is a term of type τ. The *set of all terms* occurring in a term t is denoted by $\mathbf{T}(t)$. When convenient, we write an enumeration of (meta)symbols $a_1, \ldots, a_n$ as $\bar{a}$. If $b_1, \ldots, b_k$ is another enumeration, $\bar{a}\bar{b}$ denotes the enumeration $a_1, \ldots, a_n, b_1, \ldots, b_k$.

Given a signature $\Sigma = (\mathcal{Y}, \mathcal{F})$, a Σ-interpretation $\mathcal{I}$ maps: each $\tau \in \mathcal{Y}$ to a non-empty set $\tau^{\mathcal{I}}$, the *domain* of τ in $\mathcal{I}$, with $\mathsf{\mathbf{Bool}}^{\mathcal{I}} = \{\top, \bot\}$; each $x \in \mathcal{V}_\tau$ to an element of $\tau^{\mathcal{I}}$; each $\mathsf{f} \in \mathcal{F}$ s.t. $\mathsf{f} : \tau_1 \times \cdots \times \tau_n \to \tau$ to a total function $u^{\mathcal{I}} : \tau_1^{\mathcal{I}} \times \cdots \times \tau_n^{\mathcal{I}} \to \tau^{\mathcal{I}}$ when $n > 0$ and to an element of $\tau^{\mathcal{I}}$ when $n = 0$, with $\mathsf{true}^{\mathcal{I}} = \top$ and $\mathsf{false}^{\mathcal{I}} = \bot$. The interpretation $\mathcal{I}$ induces as usual a mapping from terms t of type τ to elements $t^{\mathcal{I}}$ of $\tau^{\mathcal{I}}$. If $x_1, \ldots, x_n$ are variables and $v_1, \ldots, v_n$ are well-typed values for them, we denote by $\mathcal{I}[x_1 \mapsto v_1, \ldots, x_n \mapsto v_n]$ the Σ-interpretation that maps each x_i to v_i, and is otherwise identical to $\mathcal{I}$. A satisfiability relation between Σ-interpretations and Σ-formulas is defined inductively as usual.

A *theory* is a pair $\mathcal{T} = (\Sigma, \mathbf{I})$ where Σ is a signature and $\mathbf{I}$ is a non-empty class of Σ-interpretations, the *models of* $\mathcal{T}$, that is closed under variable reassignment (i.e., every Σ-interpretation that differs from one in $\mathbf{I}$ only in how it interprets the variables is also in $\mathbf{I}$) and isomorphism. A Σ-formula φ is $\mathcal{T}$-satisfiable (respectively $\mathcal{T}$-unsatisfiable) if it is satisfied by some (resp., no) interpretation in $\mathbf{I}$. A satisfying interpretation for φ *models (or is a model of)* φ. A formula φ is *valid in* $\mathcal{T}$ (or $\mathcal{T}$*-valid*), written $\models_{\mathcal{T}} \varphi$, if every model of $\mathcal{T}$ is a model of φ.

3 Theory of Datatypes With Shared Selectors

In this section, we consider a theory $\mathcal{D}$ of algebraic datatypes over some signature $\Sigma = (\mathcal{Y}, \mathcal{F})$ and then extend it conservatively to an expanded signature *with shared selectors*. The terms of $\mathcal{D}$ are quantifier-free. As a technical convenience, we treat free variables as constants in a suitable expansion of Σ. The types of $\mathcal{D}$ are partitioned into a set of *datatypes* $\mathcal{Y}_{\mathrm{dt}}$, and a set of other types $\mathcal{Y}_{\mathrm{ord}}$. We use the metavariables δ, ϵ to refer to datatypes and τ, υ for arbitrary first-order types. Each datatype δ is equipped with one or more *constructors*, distinguished function symbols from $\mathcal{F}$ with return type δ. For every argument k of a constructor $\mathsf{C} : \tau_1 \ldots, \tau_n \to \delta$ for δ, we assume $\mathcal{F}$ contains a *(standard) selector* $\mathsf{S}_\delta^{\mathsf{C},k} : \delta \to \tau_k$. We omit δ from the selector name when it is understood or not important. We refer the reader to the SMT-LIB 2 reference document [9] or Barrett et al. [10] for a formal definition of this theory.[1] We recall salient properties of its symbols as needed.

To start, each model of the theory, when reduced to the constructors of a datatype in the theory, is isomorphic to a term (or *Herbrand*) algebra. Concretely, this means that if $\delta \in \mathcal{Y}_{\mathrm{dt}}$ is a datatype whose constructors are $\{\mathsf{C}_1, \ldots, \mathsf{C}_\mathsf{m}\}$, then the following formulas

[1] The two references differ on how they make selectors (which are naturally partial functions) total. We follow the SMT-LIB 2 standard here.

are all $\mathcal{D}$-valid for all distinct $i, j \in \{1, \ldots, \mathsf{m}\}$

$$\forall x_1, \ldots, x_{p_i}, z_1, \ldots, z_{q_i}.\ \mathsf{C_i}(x_1, \ldots, x_{p_i}) \not\approx \mathsf{C_j}(z_1, \ldots, z_{q_i}) \qquad (\textit{Distinctness})$$

$$\forall x_1, \ldots, x_{p_i}, z_1, \ldots, z_{p_i}.\ \mathsf{C_i}(x_1, \ldots, x_{p_i}) \approx \mathsf{C_i}(z_1, \ldots, z_{p_i}) \rightarrow x_1 \approx z_1 \wedge \ldots \wedge x_{p_i} \approx z_{p_i} \qquad (\textit{Injectivity})$$

$$\forall x.\ \mathsf{isC_1}(x) \vee \cdots \vee \mathsf{isC_m}(x) \qquad (\textit{Exhaustiveness})$$

Above, we write $\mathsf{isC_i}(t)$ to denote the predicate that holds if and only if the top symbol of t is $\mathsf{C_i}$. Strictly speaking, we do not need to extend our signature with the tester symbols isC since a term of the form $\mathsf{isC}(t)$ can be considered an abbreviation for the equality $t \approx \mathsf{C}(\mathsf{S}^{\mathsf{C},1}(t), \ldots, \mathsf{S}^{\mathsf{C},n}(t))$ where $n = \mathsf{arity}(\mathsf{C})$.

Interpretations must also respect *acyclicity*, which states that constructor terms cannot be equal to any of their proper subterms.

Since all models of $\mathcal{D}$ interpret a datatype δ in the same way modulo isomorphism, we will say that δ is *finite* if its interpretation is a finite set. For simplicity, we will assume that *every type τ in $\mathcal{D}$ that is not a datatype is interpreted as an infinite set in every model of $\mathcal{D}$*. This is not a strong restriction in practice, since types with some fixed, finite cardinality k can be treated as datatypes with k nullary constructors.

The relationship between an n-ary constructor C and each of its selectors $\mathsf{S}^{\mathsf{C},k}$ with $k = 1, \ldots, n$ is captured by the following $\mathcal{D}$-valid formula:

$$\forall x_1, \ldots, x_{n_i}.\ \mathsf{S}_{\delta}^{\mathsf{C},k}(\mathsf{C}(x_1, \ldots, x_{n_i})) \approx x_k \qquad (\textit{Standard selection})$$

3.1 Shared Selectors

We extend the signature of $\mathcal{D}$ with additional selectors which we call *shared selectors* and denote as $\mathsf{S}_{\delta}^{\tau,k}$, for each datatype δ and type τ in $\mathcal{D}$ and each natural number k. Intuitively, a shared selector $\mathsf{S}_{\delta}^{\tau,k}$ for δ, when applied to a δ-term $\mathsf{C}(t_1, \ldots, t_n)$ returns the k-th argument of C that has type τ, if one exists.

Example 2. Consider again the **Tree** datatype introduced in Example 1:

$$\mathbf{Tree} = \mathsf{N_1}(\mathbf{Int}, \mathbf{Tree}, \mathbf{Tree}) \mid \mathsf{N_2}(\mathbf{Int}, \mathbf{Int}, \mathbf{Tree}, \mathbf{Tree}) \mid \mathsf{L}(\mathbf{Bool}, \mathbf{Int})$$

For term

$$t = \mathsf{N_1}(1, \mathsf{N_2}(2, 3, \mathsf{L}(\mathsf{true}, 4), \mathsf{L}(\mathsf{false}, 5)), \mathsf{L}(\mathsf{true}, 6))$$

the equalities $\mathsf{S}^{\mathbf{Int},1}(t) \approx 1$, $\mathsf{S}^{\mathbf{Int},2}(\mathsf{S}^{\mathbf{Tree},1}(t)) \approx 3$, and $\mathsf{S}^{\mathbf{Int},1}(\mathsf{S}^{\mathbf{Tree},2}(\mathsf{S}^{\mathbf{Tree},1}(t))) \approx 6$ are all valid in our extension of $\mathcal{D}$ to shared selectors. •

To define shared selectors formally, let us first define a partial function stoa (for *selector to argument*) that takes as input a natural number k, a type τ, and a constructor C, and returns the index of the k-th argument of C of type τ . We leave stoa undefined if C has fewer than k arguments of type τ.

Example 3. For the **Tree** datatype, $\mathsf{stoa}(1, \mathbf{Int}, \mathsf{N_1}) = 1$, $\mathsf{stoa}(2, \mathbf{Tree}, \mathsf{N_1}) = 3$ and $\mathsf{stoa}(1, \mathbf{Int}, \mathsf{L}) = 2$, whereas $\mathsf{stoa}(2, \mathbf{Int}, \mathsf{N_1})$, $\mathsf{stoa}(1, \mathbf{Bool}, \mathsf{N_2})$, and $\mathsf{stoa}(1, \mathbf{Tree}, \mathsf{L})$ are undefined. •

$$
\begin{aligned}
\mathcal{H}(t,\ M) = \text{match } t \text{ with} \\
x &\to x \\
\mathsf{C}(t_1,\ \ldots,\ t_n) &\to \mathsf{C}(\mathcal{H}(t_1,\ M),\ \ldots,\ \mathcal{H}(t_n,\ M)) \\
\mathsf{S}_{\boldsymbol{\delta}}^{\tau,k}(t_1) &\to \mathsf{S}_{\boldsymbol{\delta}}^{\tau,k}(\mathcal{H}(t_1,\ M)) \\
\mathsf{S}_{\delta}^{\mathsf{C},k}(t_1) &\to \begin{cases} \mathsf{S}_{\boldsymbol{\delta}}^{\boldsymbol{\tau},\,\mathsf{atos}(\boldsymbol{\tau},\ \mathsf{C},\ k)}(\mathcal{H}(t_1,\ M)) & \text{if } M(t_1) = \mathsf{C} \\ \mathsf{S}_{\boldsymbol{\delta}}^{\boldsymbol{\tau},\,\mathsf{err}(\mathsf{C},\,k)}(\mathcal{H}(t_1,\ M)) & \text{otherwise} \end{cases} \\
&\quad \text{where } \mathsf{S}_{\delta}^{\mathsf{C},k} : \delta \to \tau
\end{aligned}
$$

Fig. 1: Definition of $\mathcal{H}(t,\ M)$

More formally, in our extension of theory $\mathcal{D}$ with shared selectors, which we also refer to as $\mathcal{D}$ for convenience, the following holds for all datatypes δ, constructors C of δ, and shared selectors $\mathsf{S}^{\boldsymbol{\tau},k}$, whenever $\mathsf{stoa}(k,\ \boldsymbol{\tau},\ \mathsf{C})$ is defined:

$$\forall x_1, \ldots, x_n.\ \mathsf{S}_{\boldsymbol{\delta}}^{\boldsymbol{\tau},k}(\mathsf{C}(x_1, \ldots, x_n)) \approx x_i, \text{ where } i = \mathsf{stoa}(k,\ \boldsymbol{\tau},\ \mathsf{C}) \qquad (\textit{Shared selection})$$

It is not difficult to argue that every Σ-formula φ without shared selectors is valid in the extended theory if and only if it is valid in the original theory.

3.2 From Standard Selectors to Shared Selectors

The satisfiability problem for *constraints*, i.e., finite sets of literals, over the original theory of datatypes (without shared selectors) is decidable [10]. In this section, we introduce a transformation $\mathcal{H}$ that reduces arbitrary constraints in our extended theory $\mathcal{D}$, which may have both standard and shared selectors, to constraints with no standard selectors. Applying this transformation as an initial step allows us to determine the satisfiability of arbitrary Σ-constraints by means of a decision procedure for Σ-constraints without standard selectors.

To define this transformation, let max_Σ denote some natural number that is greater than the arity of all constructors in Σ. We define the dual of the stoa function from Subsection 3.1 as the partial function atos (for *argument to selector*) that takes as input a type τ, a constructor $\mathsf{C} : \tau_1 \times \cdots \times \tau_n \to \delta$, and a natural number $k \leq n$, and returns the number of times τ occurs in $\tau_1, \ldots, \tau_k$.

Figure 1 defines the transformation $\mathcal{H}$, which takes as arguments a Σ-term t and a mapping M. The latter consists of one entry of the form $s \mapsto \mathsf{C}$ for each datatype term s in $\mathbf{T}(t)$ where C is one of the constructors for the type of s. Without loss of generality, we assume that all applications of shared selectors $\mathsf{S}_{\boldsymbol{\delta}}^{\boldsymbol{\tau},k}$ occurring in t are such that $k < \mathsf{max}_\Sigma$. The transformation $\mathcal{H}$ leaves variables unchanged; for terms whose top symbol is a constructor or a shared selector, $\mathcal{H}$ behaves homorphically. For terms t with a standard selector $\mathsf{S}_{\delta}^{\mathsf{C},k} : \delta \to \tau$ as top symbol, we distinguish whether the argument t_1 is mapped to C by M or not. In the first case, we replace $\mathsf{S}_{\delta}^{\mathsf{C},k}$ by the shared selector

$\mathsf{S}_\delta^{\tau,\,\mathsf{atos}(\tau,\,\mathsf{C},\,k)}$. In the second case, we replace $\mathsf{S}_\delta^{\mathsf{C},k}$ by the shared selector $\mathsf{S}_\delta^{\tau,\,\mathsf{err}(\mathsf{C},\,k)}$, where err is a function that takes as arguments a constructor and a k such that $1 \leq k \leq \mathsf{arity}(\mathsf{C})$, and returns a natural number. Additionally, err has the following properties:

1. If $\mathsf{C}_1 \neq \mathsf{C}_2$ or $k_1 \neq k_2$, then $\mathsf{err}(\mathsf{C}_1,\, k_1) \neq \mathsf{err}(\mathsf{C}_2,\, k_2)$, and
2. $\mathsf{err}(\mathsf{C}_1,\, k) \geq \mathsf{max}_\Sigma$.

We use the function err in this transformation to introduce shared selectors that are unique to the pair $(\mathsf{C},\, k)$, as guaranteed by Property 1 above, and whose return value is undefined, as guaranteed by Property 2. In either case, $\mathcal{H}$ is applied recursively to t_1.

We extend $\mathcal{H}$ to sets of equalities and disequalities E as follows:

$$\begin{aligned}\mathcal{H}(E,\, M) = \{\, &\mathcal{H}(t_1,\, M) \approx \mathcal{H}(t_2,\, M) \mid t_1 \approx t_2 \in E \,\} \cup \\ &\{\, \mathcal{H}(t_1,\, M) \not\approx \mathcal{H}(t_2,\, M) \mid t_1 \not\approx t_2 \in E \,\} \cup \{\, \mathsf{isC}(t) \mid t \mapsto \mathsf{C} \in M \,\}\end{aligned}$$

In other words, for each (dis)equality, we include the corresponding constraint where the transformation is applied to both its terms. We add to this set an application of the discriminator for C to t for each $t \mapsto \mathsf{C}$ in the mapping M.

Example 4. Consider again the **Tree** datatype from Example 1. Let:

$$E = \{x \approx \mathsf{N}_1(2, y,\, \mathsf{S}^{\mathsf{N}_1,2}(x)),\, \mathsf{S}^{\mathsf{N}_1,1}(x) \approx 2,\, \mathsf{S}^{\mathsf{L},2}(x) \not\approx 0\} \text{ and } M = \{x \mapsto \mathsf{N}_1,\ y \mapsto \mathsf{L}\}$$

Then, $\mathcal{H}(E,\, M)$ is the set:

$$\{x \approx \mathsf{N}_1(2, y,\, \mathsf{S}^{\mathbf{Tree},1}(x)),\, \mathsf{S}^{\mathbf{Int},1}(x) \approx 2\} \cup \{\mathsf{S}^{\mathbf{Int},\,\mathsf{err}(\mathsf{L},\,2)}(x) \not\approx 0\} \cup \{\mathsf{isN}_1(x),\, \mathsf{isL}(y)\}$$

Since M maps x to N_1, the standard selector application $\mathsf{S}^{\mathsf{N}_1,2}(x)$ is converted to the shared selector application $\mathsf{S}^{\mathbf{Tree},1}(x)$, whereas $\mathsf{S}^{\mathsf{L},2}(x)$ is converted to $\mathsf{S}^{\mathbf{Int},\,\mathsf{err}(\mathsf{L},\,2)}(x)$. •

The following theorem states the key property of the transformation $\mathcal{H}$, namely that a set of arbitrary Σ-constraints E is satisfiable if and only if there exists some mapping M for which $\mathcal{H}(E,\, M)$ is satisfiable. The full proof of this statement is available in an extended version of this paper [24].

Theorem 1. *E is $\mathcal{D}$-satisfiable iff $\mathcal{H}(E,\, M)$ is $\mathcal{D}$-satisfiable for some M.*

Proof. We split the statement into its two implications. The proof relies on the construction of a mapping M from a model of E.
"$\Rightarrow$": If E is satisfied by some Σ-model $\mathcal{I}$ of $\mathcal{D}$, there exists a mapping $M_\mathcal{I}$ and Σ-model $\mathcal{J}$ of $\mathcal{D}$ such that $\mathcal{H}(E,\, M_\mathcal{I})$ is satisfied by $\mathcal{J}$. We show this by a particular construction for $M_\mathcal{I}$ and $\mathcal{J}$. Let the mapping $M_\mathcal{I}$ be $\{t \mapsto C \mid \mathcal{I} \models \mathsf{isC}(t), t \in \mathbf{T}(E)\}$. Construct $\mathcal{J}$ as follows. First, all types τ and constructors are interpreted by $\mathcal{J}$ the same way as in $\mathcal{I}$. Furthermore, we interpret all variables and standard selectors in $\mathcal{J}$ the same as in $\mathcal{I}$. It remains to state how shared selectors are interpreted in $\mathcal{J}$. Notice that our transformation generates shared selectors of the form $\mathsf{S}_\delta^{\tau,\,\mathsf{err}(\mathsf{C},\,k)}$. We distinguish these in the following construction.

$$\mathsf{S}_\delta^{\tau,\,\mathsf{err}(\mathsf{C},\,k)\,\mathcal{J}} = \mathsf{S}_\delta^{\mathsf{C},k\,\mathcal{I}}, \text{ and } \mathsf{S}_\delta^{\tau,k\,\mathcal{J}} = \mathsf{S}_\delta^{\tau,k\,\mathcal{I}} \text{ for all other shared selectors.}$$

The above construction is well-defined due to our definition of err. In particular, $\mathsf{err}(\mathsf{C},\, k)$ is defined uniquely for each (constructor, natural number) pair. We show $\mathcal{H}(t, M_\mathcal{I})^\mathcal{J} = t^\mathcal{I}$ for all terms $t \in \mathbf{T}(E)$ by structural induction on t.

- Case $t = x$, for some variable x: By the definition of $\mathcal{H}$ and the construction of $\mathcal{J}$, we have that $\mathcal{H}(x, M_{\mathcal{I}})^{\mathcal{J}} = x^{\mathcal{J}} = x^{\mathcal{I}}$. $\square$
- Case $t = \mathsf{C}(t_1, \ldots, t_n)$: We have that

$$\begin{aligned}
&\mathcal{H}(\mathsf{C}(t_1, \ldots, t_n), M_{\mathcal{I}})^{\mathcal{J}} \\
&= \mathsf{C}^{\mathcal{J}}(\mathcal{H}(t_1, M_{\mathcal{I}})^{\mathcal{J}}, \ldots, \mathcal{H}(t_n, M_{\mathcal{I}})^{\mathcal{J}}) \text{ by the definition of } \mathcal{H} \\
&= \mathsf{C}^{\mathcal{J}}(t_1^{\mathcal{I}}, \ldots, t_n^{\mathcal{I}}) \text{ by the induction hypothesis} \\
&= \mathsf{C}^{\mathcal{I}}(t_1^{\mathcal{I}}, \ldots, t_n^{\mathcal{I}}), \text{ by the construction of } \mathcal{J} \\
&= \mathsf{C}(t_1, \ldots, t_n)^{\mathcal{I}}.
\end{aligned}$$

- Case $t = \mathsf{S}_{\delta}^{\tau, k}(t_1)$: Firstly,

$$\begin{aligned}
&\mathcal{H}(\mathsf{S}_{\delta}^{\tau, k}(t_1), M_{\mathcal{I}})^{\mathcal{J}} \\
&= (\mathsf{S}_{\delta}^{\tau, k})^{\mathcal{J}}(\mathcal{H}(t_1, M_{\mathcal{I}})^{\mathcal{J}}) \text{ by the definition of } \mathcal{H} \\
&= (\mathsf{S}_{\delta}^{\tau, k})^{\mathcal{J}}(t_1^{\mathcal{I}}) \text{ by the induction hypothesis}
\end{aligned}$$

  By assumption, we have that $k < \mathsf{max}_{\Sigma}$. Hence, by the construction of $\mathcal{J}$, $(\mathsf{S}_{\delta}^{\tau, k})^{\mathcal{J}}(t_1^{\mathcal{I}}) = (\mathsf{S}_{\delta}^{\tau, k})^{\mathcal{I}}(t_1^{\mathcal{I}})$, which is $\mathsf{S}_{\delta}^{\tau, k}(t_1)^{\mathcal{I}}$.
- Case $t = \mathsf{S}_{\delta}^{\mathsf{C}, k}(t_1)$ where $\mathsf{S}_{\delta}^{\mathsf{C}, k} : \delta \to \tau$:
  - If $M_{\mathcal{I}}(t_1) = \mathsf{C}$, then

$$\begin{aligned}
&\mathcal{H}(\mathsf{S}_{\delta}^{\mathsf{C}, k}(t_1), M_{\mathcal{I}})^{\mathcal{J}} \\
&= (\mathsf{S}_{\delta}^{\tau, \mathsf{atos}(\tau,\, \mathsf{C},\, k)})^{\mathcal{J}}(\mathcal{H}(t_1, M_{\mathcal{I}}))^{\mathcal{J}} \text{ by the definition of } \mathcal{H} \\
&= (\mathsf{S}_{\delta}^{\tau, \mathsf{atos}(\tau,\, \mathsf{C},\, k)})^{\mathcal{J}}(t_1^{\mathcal{I}}), \text{ by the induction hypothesis}
\end{aligned}$$

    By definition of atos and since k is a valid argument position of C, we have that $\mathsf{atos}(\tau, \mathsf{C}, k) \leq \mathsf{arity}(\mathsf{C}) < \mathsf{max}_{\Sigma}$. Hence, by the construction of $\mathcal{J}$, we have that $(\mathsf{S}_{\delta}^{\tau, \mathsf{atos}(\tau, \mathsf{C}, k)})^{\mathcal{J}}(t_1^{\mathcal{I}}) = (\mathsf{S}_{\delta}^{\tau, \mathsf{atos}(\tau, \mathsf{C}, k)})^{\mathcal{I}}(t_1^{\mathcal{I}})$. Since $M_{\mathcal{I}}(t_1) = \mathsf{C}$, by construction of $M_{\mathcal{I}}$, we have that $t_1^{\mathcal{I}}$ is a term of the form $\mathsf{C}(s_1, \ldots, s_n)$, and hence $(\mathsf{S}_{\delta}^{\tau, \mathsf{atos}(\tau, \mathsf{C}, k)})^{\mathcal{I}}(t_1^{\mathcal{I}}) = s_{\mathsf{stoa}(\mathsf{atos}(\tau, \mathsf{C}, k), \tau, \mathsf{C})} = s_k$. Furthermore, $s_k = \mathsf{S}_{\delta}^{\mathsf{C}, k}(t_1)^{\mathcal{I}}$.
  - If $M_{\mathcal{I}}(t_1) \neq \mathsf{C}$, then

$$\begin{aligned}
&\mathcal{H}(\mathsf{S}_{\delta}^{\mathsf{C}, k}(t_1), M_{\mathcal{I}})^{\mathcal{J}} \\
&= (\mathsf{S}_{\delta}^{\tau, \mathsf{err}(\mathsf{C}, k)})^{\mathcal{J}}(\mathcal{H}(t_1, M_{\mathcal{I}})^{\mathcal{J}}) \text{ by the definition of } \mathcal{H}, \\
&= (\mathsf{S}_{\delta}^{\tau, \mathsf{err}(\mathsf{C}, k)})^{\mathcal{J}}(t_1^{\mathcal{I}}), \text{ by the induction hypothesis,} \\
&= (\mathsf{S}_{\delta}^{\mathsf{C}, k})^{\mathcal{I}}(t_1^{\mathcal{I}}), \text{ by the construction of } \mathcal{J}, \\
&= \mathsf{S}_{\delta}^{\mathsf{C}, k}(t_1)^{\mathcal{I}}.
\end{aligned}$$

$\square$

Since $\mathcal{I}$ satisfies E and since $\mathcal{H}(t, M_{\mathcal{I}})^{\mathcal{J}} = t^{\mathcal{I}}$ for all terms $t \in \mathbf{T}(E)$, we have that $\mathcal{J}$ satisfies the equalities and disequalities in $\mathcal{H}(E, M_{\mathcal{I}})$ of the form $(\neg)\mathcal{H}(t_1, M_{\mathcal{I}}) \approx$

$\mathcal{H}(t_2,\ M_{\mathcal{I}})$. By construction of $M_{\mathcal{I}}$, we have that $\mathcal{J}$ satisfies the constraints in $\mathcal{H}(E,\ M_{\mathcal{I}})$ of the form $\mathsf{isC}(t)$ where $t \mapsto \mathsf{C} \in M_{\mathcal{I}}$. Hence, $\mathcal{J}$ satisfies $\mathcal{H}(E,\ M_{\mathcal{I}})$.

"$\Leftarrow$": If $\mathcal{H}(E,\ M)$ is satisfied by some Σ-model $\mathcal{J}$ of $\mathcal{D}$ for some mapping M, then E is satisfied by some Σ-model $\mathcal{I}$ of $\mathcal{D}$. We show this by constructing $\mathcal{I}$ as follows. First, all types, constructors, variables and have the same interpretation in $\mathcal{I}$ as in $\mathcal{J}$. Furthermore, all shared selectors have the same interpretation in $\mathcal{I}$ as in $\mathcal{J}$. We interpret standard selectors in $\mathcal{I}$ as follows.

$$\mathsf{S}_{\boldsymbol{\delta}}^{\mathsf{C},k}(t)^{\mathcal{I}} = \begin{cases} \mathsf{S}_{\boldsymbol{\delta}}^{\tau,\,\mathsf{atos}(\boldsymbol{\tau},\,\mathsf{C},\,k)}(t)^{\mathcal{I}} & \text{if } M(t) = \mathsf{C} \\ \mathsf{S}_{\boldsymbol{\delta}}^{\tau,\,\mathsf{err}(\mathsf{C},\,k)}(t)^{\mathcal{I}} & \text{otherwise} \end{cases}$$

We show $t^{\mathcal{I}} = \mathcal{H}(t, M)^{\mathcal{J}}$ for all terms $t \in \mathbf{T}(E)$ by structural induction on t as follows.

- Case $t = x$, for some variable x: By the definition of $\mathcal{H}$ and the construction of $\mathcal{J}$, we have that $\mathcal{H}(x, M)^{\mathcal{J}} = x^{\mathcal{J}} = x^{\mathcal{I}}$. □
- Case $t = \mathsf{C}(t_1, ..., t_n)$: We have that

$$\begin{aligned} &\mathcal{H}(\mathsf{C}(t_1, ..., t_n), M_{\mathcal{I}})^{\mathcal{J}} \\ &= \mathsf{C}^{\mathcal{J}}(\mathcal{H}(t_1, M)^{\mathcal{J}}, ..., \mathcal{H}(t_n, M)^{\mathcal{J}}) \text{ by the definition of } \mathcal{H}, \\ &= \mathsf{C}^{\mathcal{J}}(t_1^{\mathcal{I}}, ..., t_n^{\mathcal{I}}) \text{ by the induction hypothesis,} \\ &= \mathsf{C}^{\mathcal{I}}(t_1^{\mathcal{I}}, ..., t_n^{\mathcal{I}}), \text{ by the construction of } \mathcal{I} \\ &= \mathsf{C}(t_1, ..., t_n)^{\mathcal{I}}. \end{aligned}$$

□
- Case $t = \mathsf{S}_{\boldsymbol{\delta}}^{\boldsymbol{\tau},k}(t_1)$:

$$\begin{aligned} &\mathcal{H}(\mathsf{S}_{\boldsymbol{\delta}}^{\boldsymbol{\tau},k}(t_1), M)^{\mathcal{J}} \\ &= (\mathsf{S}_{\boldsymbol{\delta}}^{\boldsymbol{\tau},k})^{\mathcal{J}}(\mathcal{H}(t_1, M)^{\mathcal{J}}) \text{ by the definition of } \mathcal{H}, \\ &= (\mathsf{S}_{\boldsymbol{\delta}}^{\boldsymbol{\tau},k})^{\mathcal{J}}(t_1^{\mathcal{I}}) \text{ by the induction hypothesis,} \\ &= (\mathsf{S}_{\boldsymbol{\delta}}^{\boldsymbol{\tau},k})^{\mathcal{I}}(t_1^{\mathcal{I}}) \text{ by the construction of } \mathcal{I}, \\ &= \mathsf{S}_{\boldsymbol{\delta}}^{\boldsymbol{\tau},k}(t_1)^{\mathcal{I}} \end{aligned}$$

□
- Case $t = \mathsf{S}_{\boldsymbol{\delta}}^{\mathsf{C},k}(t_1)$ where $\mathsf{S}_{\boldsymbol{\delta}}^{\mathsf{C},k} : \delta \to \tau$:
  - If $M(t_1) = \mathsf{C}$, then

$$\begin{aligned} &\mathcal{H}(\mathsf{S}_{\boldsymbol{\delta}}^{\mathsf{C},k}(t_1), M)^{\mathcal{J}} \\ &= (\mathsf{S}_{\boldsymbol{\delta}}^{\boldsymbol{\tau},\,\mathsf{atos}(\boldsymbol{\tau},\,\mathsf{C},\,k)})^{\mathcal{J}}(\mathcal{H}(t_1, M_{\mathcal{I}})^{\mathcal{J}}) \text{ by the definition of } \mathcal{H}, \\ &= (\mathsf{S}_{\boldsymbol{\delta}}^{\boldsymbol{\tau},\,\mathsf{atos}(\boldsymbol{\tau},\,\mathsf{C},\,k)})^{\mathcal{J}}(t_1^{\mathcal{I}}), \quad \text{by the induction hypothesis,} \\ &= (\mathsf{S}_{\boldsymbol{\delta}}^{\boldsymbol{\tau},\,\mathsf{atos}(\boldsymbol{\tau},\,\mathsf{C},\,k)})^{\mathcal{I}}(t_1^{\mathcal{I}}), \quad \text{by the construction of } \mathcal{I}, \\ &= \mathsf{S}_{\boldsymbol{\delta}}^{\mathsf{C},k}(t_1)^{\mathcal{I}}, \quad \text{by the construction of } \mathcal{I}, \text{ since } M(t_1) = \mathsf{C}. \end{aligned}$$

- If $M(t_1) \neq \mathsf{C}$, then

$$
\begin{aligned}
&\mathcal{H}(\mathsf{S}^{\mathsf{C},k}_{\boldsymbol{\delta}}(t_1), M_{\mathcal{I}})^{\mathcal{J}} \\
&= (\mathsf{S}^{\boldsymbol{\tau},\,\mathsf{err}(\mathsf{C},\,k)}_{\boldsymbol{\delta}})^{\mathcal{J}}(\mathcal{H}(t_1, M_{\mathcal{I}})^{\mathcal{J}}) \text{ by the definition of } \mathcal{H} \\
&= (\mathsf{S}^{\boldsymbol{\tau},\,\mathsf{err}(\mathsf{C},\,k)}_{\boldsymbol{\delta}})^{\mathcal{J}}(t_1^{\mathcal{I}}), \text{ by the induction hypothesis,} \\
&= (\mathsf{S}^{\boldsymbol{\tau},\,\mathsf{err}(\mathsf{C},\,k)}_{\boldsymbol{\delta}})^{\mathcal{I}}(t_1^{\mathcal{I}}), \text{ by the construction of } \mathcal{I}, \\
&= \mathsf{S}^{\mathsf{C},k}_{\boldsymbol{\delta}}(t_1)^{\mathcal{I}}, \text{ by the construction of } \mathcal{I}, \text{ since } M(t_1) \neq \mathsf{C}.
\end{aligned}
$$

□

Since $\mathcal{J}$ satisfies $\mathcal{H}(E, M)$ and $t^{\mathcal{I}} = \mathcal{H}(t, M)^{\mathcal{J}}$ for all $t \in \mathbf{T}(E)$, we have that $\mathcal{I}$ satisfies the equalities and disequalities in $\mathcal{H}(E, M_{\mathcal{I}})$ of the form $(\neg)\mathcal{H}(t_1, M) \approx \mathcal{H}(t_2, M)$. Furthermore, since $\mathcal{J}$ satisfies the constraints $\mathsf{isC}(t)$ for all $t \mapsto \mathsf{C} \in M$ and since $t^{\mathcal{I}} = t^{\mathcal{J}}$, we have that $\mathcal{I}$ satisfies these constraints as well. Thus, $\mathcal{I}$ satisfies $\mathcal{H}(E, M)$. □

Corollary 1. *For some index sets I and J, and set E of Σ-literals without standard selectors, let*

$$
\begin{aligned}
&E_0 = E \cup \{\, \mathsf{S}^{\mathsf{C}_{\mathsf{j_i}}, k_i}(x_i) \approx y_i \mid i \in I, j \in J \,\} \qquad \text{and} \\
&E_1 = E \wedge \{\, \mathsf{ite}(\mathsf{isC}_{\mathsf{j_i}}(x_i),\ \mathsf{S}^{\tau,\,\mathsf{atos}(\tau,\,\mathsf{C}_{\mathsf{j_i}},\,k_i)}(x_i),\ \mathsf{S}^{\tau,\,\mathsf{err}(\mathsf{C}_{\mathsf{j_i}},\,k_i)}(x_i)) \approx y_i \mid i \in I, j \in J \,\} .
\end{aligned}
$$

The sets E_0 and E_1 are equisatisfiable in $\mathcal{D}$.

Proof. For an interpretation $\mathcal{I}$, then let $M_{\mathcal{I}}$ be the mapping $\{t \mapsto C \mid \mathcal{I} \models \mathsf{isC}(t), t \in \mathbf{T}(E_0 \cup E_1)\}$. We first show that $\mathcal{I}$ satisfies $\mathcal{H}(E_0, M_{\mathcal{I}})$ if and only if $\mathcal{I}$ satisfies E_1.

First, for each (dis)equality $(\neg)t_1 \approx t_2 \in E$, we have that $\mathcal{H}(E_0, M_{\mathcal{I}})$ contains a (dis)equality of the form $(\neg)\mathcal{H}(t_1, M_{\mathcal{I}}) \approx \mathcal{H}(t_2, M_{\mathcal{I}})$, which by definition of $\mathcal{H}$ is equivalent to $(\neg)t_1 \approx t_2$, since E does not contain standard selectors. Second, by definition of $M_{\mathcal{I}}$, we have that $\mathcal{I} \models \{\mathsf{isC}(t) \mid t \mapsto \mathsf{C} \in M_{\mathcal{I}}\}$. Since $\mathcal{H}(y_i, M_{\mathcal{I}}) = y_i$ for all y_i, it suffices to show $\mathcal{H}(\mathsf{S}^{\mathsf{C}_{\mathsf{j_i}}, k_i}(x_i), M_{\mathcal{I}})^{\mathcal{I}}$ is equal to $\mathsf{ite}(\mathsf{isC}_{\mathsf{j_i}}(x_i),\ \mathsf{S}^{\tau,\,\mathsf{atos}(\tau,\,\mathsf{C}_{\mathsf{j_i}},\,k_i)}(x_i),\ \mathsf{S}^{\tau,\,\mathsf{err}(\mathsf{C}_{\mathsf{j_i}},\,k_i)}(x_i))^{\mathcal{I}}$ for all C_{j_i}, k_i, and x_i. By the definition of $\mathcal{H}$ and $M_{\mathcal{I}}$, we have that $\mathcal{H}(\mathsf{S}^{\mathsf{C}_{\mathsf{j_i}}, k_i}(x_i), M_{\mathcal{I}}) = \mathsf{S}^{\tau,\,\mathsf{atos}(\tau,\,\mathsf{C}_{\mathsf{i}},\,k_i)}(x_i)$ when $\mathcal{I} \models \mathsf{isC}_{\mathsf{j_i}}(t)$, and $\mathsf{S}^{\tau,\,\mathsf{err}(\mathsf{C}_{\mathsf{i}},\,k_i)}(x_i)$ when $\mathcal{I} \not\models \mathsf{isC}_{\mathsf{j_i}}(t)$. Hence, $\mathcal{H}(\mathsf{S}^{\mathsf{C}_{\mathsf{j_i}}, k_i}(x_i), M_{\mathcal{I}})^{\mathcal{I}}$ is equal to $\mathsf{ite}(\mathsf{isC}_{\mathsf{j_i}}(x_i),\ \mathsf{S}^{\tau,\,\mathsf{atos}(\tau,\,\mathsf{C}_{\mathsf{j_i}},\,k_i)}(x_i),\ \mathsf{S}^{\tau,\,\mathsf{err}(\mathsf{C}_{\mathsf{j_i}},\,k_i)}(x_i))^{\mathcal{I}}$.

Thus, $\mathcal{I}$ satisfies $\mathcal{H}(E_0, M_{\mathcal{I}})$ if and only if $\mathcal{I}$ satisfies E_1. Thus, by Theorem 1, we have that E_0 is satisfiable if and only if E_1 is satisfiable. □

Using this corollary, we can reduce (possibly after some literal flattening) the satisfiability of an arbitrary set of Σ-constraints E_0 to a set of Σ-constraints E_1 not containing standard selectors. In particular, our implementation in cvc4 replaces each application of the form $\mathsf{S}^{\mathsf{C}_{\mathsf{j_i}}, k_i}(x_i)$ by the term $\mathsf{ite}(\mathsf{isC}_{\mathsf{j_i}}(x_i),\ \mathsf{S}^{\tau,\,\mathsf{atos}(\tau,\,\mathsf{C}_{\mathsf{j_i}},\,k_i)}(x_i),\ \mathsf{S}^{\tau,\,\mathsf{err}(\mathsf{C}_{\mathsf{j_i}},\,k_i)}(x_i))$ during a preprocessing pass on the input formula.

4 Decision Procedure for Datatypes with Shared Selectors

This section describes a tableau-like calculus for deciding constraint satisfiability in $\mathcal{D}$, with constraint variables interpreted existentially. The calculus is parametrized by the theory's signature Σ. By the results of the previous section, we can restrict with no loss of generality the input language to sets of equalities and disequalities between Σ-terms with no standard selectors and no discriminators. Since our calculus is based on similar calculi for datatypes that have been presented in detail in previous work [10, 20], we focus on our modifications to accommodate shared selectors.

The derivation rules of the calculus operate on a current set E of constraints as specified in Figure 2. A derivation rule can be applied to E if its premises are met. Some of those premises check membership in the *congruence closure* E^* of E, the smallest superset of E that is closed under entailment in the theory of equality.[2] A rule's conclusion either modifies E or replaces it by $\bot$ to indicate unsatisfiability. There, the notation $E, t \approx s$ abbreviates $E \cup \{t \approx s\}$; the notation $\bar{t} \approx \bar{u}$ stands for the set of equalities between the corresponding elements of $\bar{t}$ and $\bar{u}$. The Split rule has multiple alternative conclusions, denoting branching.

A rule application is *redundant* if (one of) its conclusion(s) leaves E unchanged. The rules are applied to build a *derivation tree*, i.e., a tree whose nodes are finite sets of (dis)equalities, with an *initial constraint set* E_0 as its root and child nodes obtained by a non-redundant rule application to their parent. We say that E_0 *has* a derivation tree D if D is a derivation tree with root E_0. A node is *saturated* if it admits only redundant rule applications. A derivation tree is *closed* if all of its leaf nodes are $\bot$. Intuitively, a derivation tree is generated progressively from E_0 by applying a derivation rule to a leaf node. The rules are applied until the derivation tree becomes closed (indicating that the initial set E_0 is $\mathcal{D}$-unsat) or contains a saturated leaf node (indicating that E_0 is $\mathcal{D}$-sat).

In the calculus, all reasoning based on the general properties of equality is encapsulated in the rule Conflict, which detects that congruent terms are forced to be distinct. The remaining rules perform datatype reasoning proper, with Decompose computing a downward equality closure based on the injectivity of constructors and Clash detecting failures based on their distinctness. The Cycle rule recognizes when a constructor term must be equivalent to one of its subterms, which is forbidden in all models of the theory.

The calculus also incrementally unrolls terms by branching on different constructors, with the Split rule performing case distinctions on constructors for various terms occurring in E. The main modification from the previous calculi for the theory of datatypes is that this Split rule operates on shared selectors. Its application can be seen as an on-the-fly transformation from standard to shared selectors as described in Section 3.2. Indeed, for each constructor $\mathsf{C_i}$ in its conclusion, the following holds with a mapping M such that $M(t) = \mathsf{C_i}$:

$$\mathsf{S}_{\boldsymbol{\delta}}^{\tau_1,\, \mathsf{atos}(\boldsymbol{\tau_1},\, \mathsf{C_i},\, 1)}(t) = \mathcal{H}(\mathsf{S}_{\boldsymbol{\delta}}^{\mathsf{C_i},1}(t),\ M),\ \ldots,\ \mathsf{S}_{\boldsymbol{\delta}}^{\tau_{\mathsf{n_i}},\, \mathsf{atos}(\boldsymbol{\tau_{\mathsf{n_i}}},\, \mathsf{C_i},\, n_i)}(t) = \mathcal{H}(\mathsf{S}_{\boldsymbol{\delta}}^{\mathsf{C_i},n}(t),\ M)$$

Any derivation strategy for the calculus that does not stop until it generates a closed tree or a saturated node yields a decision procedure for the $\mathcal{D}$-satisfiability of sets of

[2] Such tests are effective by well-known results about the theory of equality [6].

$$\frac{t \approx u \in E^* \quad t \not\approx u \in E}{\bot}\ \textsc{Conflict}$$

$$\frac{\mathsf{C_1}(\bar{t}) \approx \mathsf{C_1}(\bar{u}) \in E^*}{E := E,\ \bar{t} \approx \bar{u}}\ \textsc{Decompose} \qquad \frac{\mathsf{C_1}(\bar{t}) \approx \mathsf{C_2}(\bar{u}) \in E^* \quad \mathsf{C_1} \neq \mathsf{C_2}}{\bot}\ \textsc{Clash}$$

$$\frac{\mathsf{C_n}(\bar{u}_n u \bar{v}_n) \approx u_{n-1},\ \ldots,\ \mathsf{C_2}(\bar{u}_2 u_2 \bar{v}_2) \approx u_1,\ \mathsf{C_1}(\bar{u}_1 u_1 \bar{v}_1) \approx u \in E^* \quad n \geq 1}{\bot}\ \textsc{Cycle}$$

$$\frac{\mathsf{S}_{\boldsymbol{\delta}}^{\boldsymbol{\tau}, n}(t) \in \mathbf{T}(E)\ \text{ or } \delta \text{ is finite}}{\begin{array}{l} E := E,\ t \approx \mathsf{C_1}(\mathsf{S}_{\boldsymbol{\delta}}^{\tau_{1,1},\, \mathsf{atos}(\boldsymbol{\tau_{1,1}},\, \mathsf{C_1},\, 1)}(t),\ \ldots,\ \mathsf{S}_{\boldsymbol{\delta}}^{\tau_{1,\mathsf{n_1}},\, \mathsf{atos}(\boldsymbol{\tau_{1,n_1}},\, \mathsf{C_1},\, n_1)}(t)) \\ \quad \vdots \\ E := E,\ t \approx \mathsf{C_m}(\mathsf{S}_{\boldsymbol{\delta}}^{\tau_{\mathsf{m},1}\, 1,\, \mathsf{atos}(\boldsymbol{\tau_{m,1}},\, \mathsf{C_m},\, 1)}(t),\ \ldots,\ \mathsf{S}_{\boldsymbol{\delta}}^{\tau_{\mathsf{m},\mathsf{n_m}},\, \mathsf{atos}(\boldsymbol{\tau_{m,n_m}},\, \mathsf{C_m},\, n_m)}(t)) \end{array}}\ \textsc{Split}$$

where δ has constructors $\mathsf{C_1}, \ldots, \mathsf{C_m}$ and $\mathsf{C_i} : \tau_{1,i} \times \cdots \times \tau_{i,n_i} \to \delta$, $1 \leq i \leq m$

Fig. 2: Derivation rules.

Σ-literals. We prove this similarly to previous work [10, 20], but using shared selectors and in the simpler setting obtained by assuming the availability of a congruence closure procedure. The full proofs are available in an extended version of this paper [24].

Proposition 1 (Termination). *All derivation trees in the calculus are finite.*

Proof. Consider a derivation tree with a root rode E. Let $D \subseteq \mathbf{T}(E)$ be the set of terms whose types are finite datatypes, and let $N \subseteq \mathbf{T}(E)$ be the set of terms occurring as arguments to shared selectors. For each term $t \in D$, let

$$S_t^0 = \{t\} \qquad S_t^{i+1} = S_t^i \cup \{\mathsf{S}_{\boldsymbol{\delta}}^{\tau,n}(u)\} \mid u^\delta \in S_t^i,\ \delta \in \mathcal{Y}_{\mathrm{dt}},\ |\delta| \text{ is finite},\ \mathsf{S}_{\boldsymbol{\delta}}^{\tau,n} \in \mathcal{F}_{\mathrm{sel}}\}$$

and let S_t^∞ be the limit of this sequence. This is a finite set for each t, since all selector chains applied to t are finite. Let S^∞ be the union of all sets S_t^∞ where $t \in D$, and let

$$\mathbf{T}^\infty(E) = \mathbf{T}\left(E \cup \left\{ \mathsf{C}_i(\mathsf{S}_{\boldsymbol{\delta}}^{\tau_{\mathsf{i}},\, \mathsf{atos}(\boldsymbol{\tau_i},\, \mathsf{C_i},\, 1)}(t), \ldots, \mathsf{S}_{\boldsymbol{\delta}}^{\tau_{\mathsf{n_i}},\, \mathsf{atos}(\boldsymbol{\tau_{n_i}},\, \mathsf{C_i},\, n_i)}(t)) \;\middle|\; \begin{array}{l} t^\delta \in N \cup S^\infty, \\ \mathsf{C}_i \in \mathcal{F}_{\mathrm{ctr}}^{\delta} \end{array} \right\}\right)$$

In a derivation tree with root node E, it can be shown by induction on the rules of the calculus that each non-root nor $\bot$-node F is such that $\mathbf{T}(F) \subseteq \mathbf{T}^\infty(E)$, and hence contains an equality between two terms from $\mathbf{T}^\infty(E)$ not occurring in its parent node. Thus, the depth of a branch in a derivation tree with root node E is at most $|\mathbf{T}^\infty(E)|^2$, which is finite since $\mathbf{T}^\infty(E)$ is finite. ◻

Proposition 2 (Refutation Soundness). *If a constraint set E_0 has a closed derivation tree, then it is $\mathcal{D}$-unsatisfiable.*

Proof. The proof is by structural induction on the derivation tree with root node E. If the tree is an application of *Conflict*, *Clash* or *Cycle*, then E is trivially $\mathcal{D}$-unsatisfiable, due to equality reasoning and for $\mathcal{D}$ not accepting models in which *distinctness* is violated or a constructor term has the same value as one of its subterms, respectively. If a child node of E is a closed derivation tree whose root node $E \cup t \approx u$ is obtained by applying *Inject*, since $\mathsf{C}_1(t) \approx \mathsf{C}_\mathsf{i}(i) \in E^*$, by *injectivity* and equality reasoning, $E \models_{\mathcal{D}} t \approx u$. Thus, by the induction hypothesis, $E \cup t \approx u$ is $\mathcal{D}$-unsatisfiable and thus E is $\mathcal{D}$-unsatisfiable. The remaining case is that child nodes of E are closed derivation trees whose root nodes are the result of applying *Split* on a term t^δ. By the induction hypothesis $E \cup t \approx \mathsf{C}_i(\mathsf{S}_\delta^{\tau_\mathsf{i},\,\mathsf{atos}(\tau_\mathsf{i},\,\mathsf{C}_\mathsf{i},\,1)}(t),\ \ldots,\ \mathsf{S}_\delta^{\tau_{\mathsf{n}_\mathsf{i}},\,\mathsf{atos}(\tau_{\mathsf{n}_\mathsf{i}},\,\mathsf{C}_\mathsf{i},\,n_i)}(t))$ is $\mathcal{D}$-unsatisfiable for each $\mathsf{C}_i \in \mathcal{F}^\delta_{\mathrm{ctr}}$. By *exhaustiveness* all models of $\mathcal{D}$ entail exactly one $t \approx \mathsf{C}_i(\mathsf{S}_\delta^{\mathsf{C}_\mathsf{i},\,1}(t),\ \ldots,\ \mathsf{S}_\delta^{\mathsf{C}_\mathsf{i},\,n}(t))$. Since Theorem 1 guarantees that in a model $\mathcal{I}$ of $\mathcal{D}$ in which t is interpreted as a term constructed with C_i it holds that $\mathsf{S}_\delta^{\mathsf{C}_\mathsf{i},\,j}(t)^\mathcal{I} = \mathsf{S}_\delta^{\tau_\mathsf{i},\,\mathsf{atos}(\tau_\mathsf{i},\,\mathsf{C}_\mathsf{i},\,j)}(t)^\mathcal{I}$, for $1 \leq j \leq n$, then we can conclude that E is $\mathcal{D}$-unsatisfiable. □

Proposition 3 (Solution Soundness). *If a constraint set E_0 has a derivation tree with a saturated node, then it is $\mathcal{D}$-satisfiable.*

Proof. The proof relies on the construction of a specific term-generated interpretation $\mathcal{I}$ from the set of equality literals F in a saturated node of a tree whose root is E. We will show that by construction $\mathcal{I}$ models E.

We build $\mathcal{I}$ by assigning to terms as their values constructor terms from their congruence classes in F^*. Since *Inject* and *Clash* cannot be applied on F, no congruence class in F^* contains more than one constructor term modulo congruence, i.e. no two terms with different constructors $\mathsf{C}_1(\bar{t})$ and $\mathsf{C}_2(\bar{u})$ and all terms $\mathsf{C}_1(\bar{t}_1), \ldots, \mathsf{C}_1(\bar{t}_n)$ in the same class are equivalent modulo congruence, since $\bar{t}_1, \ldots, \bar{t}_n$ are also congruent. Since *Cycle* cannot be applied on F, each term modulo congruence is also acyclic. Finally, since *Split* cannot be applied as well, every term of the form $\mathsf{S}_\delta^{\tau_\mathsf{i},\,n}(t)$ is such that t contains a constructor term in its congruence class.

To finish building $\mathcal{I}$ it remains to specify how it interprets terms t whose congruence classes do not contain constructor terms. If t is not a selector application, it can be assigned any distinct value from its respective domain in $\mathcal{I}$ according to two conditions: the value has not been assigned before to a term in a congruence class; and it is distinct from the resulting values computed, after such an assignment, to previously unassigned terms. The first condition can always be trivially satisfied since if a term is not congruent to a constructor term, it must have an infinite type, and therefore there are infinitely many distinct values to be assigned. To satisfy the second condition it suffices to perform the assignment by adding an equality to F, derive a saturated node F', which is always different from $\bot$, and perform the next assignment with a distinct value according to constructor terms of F' modulo congruence.

In the case that t is a selector application $\mathsf{S}_\delta^{\tau,k}(u)$, we distinguish two cases, depending on the congruence class of u containing a constructor term modulo congruence $\mathsf{C}(u_1,\ \ldots,\ u_n)$ with $\mathsf{stoa}(k,\ \tau,\ \mathsf{C})$ being defined or not. If it is defined, $\mathsf{S}_\delta^{\tau,k}(u)^\mathcal{I} = \mathsf{stoa}(k,\ \tau,\ \mathsf{C})^I$. Otherwise we apply an analogous process as before of assigning a distinct value for $\mathsf{S}_\delta^{\tau,k}(u)$.

It is easy to show that $\mathcal{I}$ satisfies distinctness, injectivity, exhaustiveness, shared selection, and acyclicity by construction. Therefore it satisfies all properties of a $\mathcal{D}$-model. Finally, we show that $\mathcal{I}$ satisfies all equalities and disequalities in F. Since $\mathcal{I}$ can be successfully built from any saturated node by assigning the same value to congruent terms, it trivially satisfies all equalities in F. Since *Conflict* cannot be applied, no two congruent terms occur in a disequality in F. Since each congruence class is assigned a distinct value by $\mathcal{I}$, it satisfies all disequalities. Therefore $\mathcal{I}$ is a model of F, and since $E \subseteq F$, $\mathcal{I}$ is also a model of E. ◻

Theorem 2. *Constraint satisfiability in the theory $\mathcal{D}$ of datatypes with (standard and) shared selectors is decidable.*

Proof. Completeness, i.e. derivation trees with root E are closed if E is $\mathcal{D}$-unsatisfiable or have a saturated node if E is $\mathcal{D}$-satisfiable, is a direct consequence of Propositions 1, 2, and 3. Therefore, since the rules from Figure 2 are sound and complete, the calculus constitutes a decision procedure for the satisfiability of Σ-constraints. ◻

5 Using Shared Selectors for Syntax-Guided Synthesis

In this section, we show how the theory of datatypes with shared selectors can substantially improve the performance of an approach by Reynolds et al. [22] for performing *syntax-guided synthesis* (SyGuS) [1] directly within an SMT solver.

Syntax-guided synthesis is the problem of automatically synthesizing a function that satisfies a given specification, but with the addition of explicit syntactic restrictions on the solution space. These restrictions specify that the function must be built with selected operators over basic types (such as arithmetic and Boolean operators) and belong to the language generated by a given grammar. Grammars allow users to specify formally a set of candidates for the desired function, thus reducing the search effort of a SyGuS solver.

More technically, a syntax-guided synthesis problem for a function f in a background theory T of the basic types consists of:

1. a set of semantic restrictions, or specification, given by a (second-order) T-formula of the form $\exists f.\, \forall \bar{x}.\, \varphi[f, \bar{x}]$, and
2. a set of syntactic restrictions on the solutions for f, given by a grammar R.

A solution for f is a lambda term $\lambda \bar{y}.\, e$ of the same type as f, such that (*i*) $\forall \bar{x}.\, \varphi[\lambda \bar{y}.\, e, \bar{x}]$ is valid in T (modulo beta-reductions) and (*ii*) e is in the language generated by R.

cvc4 incorporates a SyGuS solver that automatically encodes the solution space of a SyGuS problem as a set of algebraic datatypes mirroring the problem's syntactic restrictions [22]. A deep embedding of the datatypes in the problem's background theory T, realized as a set of automatically generated axioms, provides a semantics for datatype values in terms of the semantic values in T.

Example 5. Consider the problem of synthesizing a binary function f over the integers such that f is commutative (i.e., $\exists f\, \forall x y.\, f(x, y) \approx f(y, x)$), and with the solution space for f defined by a context-free grammar R with start symbol A and production rules:

$$A \to x \mid y \mid 0 \mid 1 \mid A + A \mid A - A \mid \mathsf{ite}(B,\ A,\ A) \qquad\qquad B \to A \geq A \mid A \approx A \mid \neg B$$

The following mutually recursive datatypes capture the grammar R. The datatypes themselves correspond to R's non-terminals (e.g., $\mathbf{a}$ corresponds to A), their constructors correspond to production rules (e.g., X corresponds to $A \rightarrow x$):

$$\begin{aligned}\mathbf{a} &= \mathsf{X} \mid \mathsf{Y} \mid \mathsf{Zero} \mid \mathsf{One} \mid \mathsf{Plus}(\mathbf{a}, \mathbf{a}) \mid \mathsf{Minus}(\mathbf{a}, \mathbf{a}) \mid \mathsf{Ite}(\mathbf{b},\ \mathbf{a},\ \mathbf{a}) \\ \mathbf{b} &= \mathsf{Geq}(\mathbf{a},\ \mathbf{a}) \mid \mathsf{Eq}(\mathbf{a},\ \mathbf{a}) \mid \mathsf{Neg}(\mathbf{b})\end{aligned}$$

Datatypes like the ones above are associated with the programs they represent through *evaluation functions* that map datatype values, expressed as variable-free constructor terms, to expressions over the basic types. For example, the evaluation function for $\mathbf{a}$ is denoted by a function symbol $\mathsf{eval}_{\mathbf{a}} : \mathbf{a} \times \mathbf{Int} \times \mathbf{Int} \rightarrow \mathbf{Int}$, and the specific term $\mathsf{eval}_{\mathbf{a}}(\mathsf{Plus}(\mathsf{X}, \mathsf{X}), 2, 3)$ is interpreted as $(x + x)\{x \mapsto 2, y \mapsto 3\} = 2 + 2 = 4$. The evaluation functions are defined axiomatically by a set of quantified formulas that, in this case, can be handled by any SMT solver that, like cvc4, supports the combined theory of datatypes, linear arithmetic, and uninterpreted functions. The SyGuS problem for f in this example can then be stated as the *first-order* formula:

$$\forall x y.\ \mathsf{eval}_{\mathbf{a}}(d, x, y) \approx \mathsf{eval}_{\mathbf{a}}(d, y, x) \tag{3}$$

where d is a fresh constant of type $\mathbf{a}$. This formula has models in which d is interpreted as Zero or $\mathsf{Plus}(\mathsf{X}, \mathsf{Y})$, which correspond to solutions $f = \lambda x y.\ 0$ and $f = \lambda x y.\ x + y$ for the original problem, respectively.[3] •

Since cvc4 is a DPLL(T)-based solver [11], for a problem like the one in the example above, it will find a possible solution for d by first guessing its top constructor symbol with an application of the Split rule from Figure 2. The effect of the rule is achieved in practice with the generation of *splitting lemmas* such as the following, which we write here with discriminators and standard selectors for simplicity:

$$\mathsf{isX}(d) \vee \mathsf{isY}(d) \vee \cdots \vee \mathsf{isIte}(d) \tag{4}$$
$$\mathsf{isX}(\mathsf{S}^{\mathsf{Plus},1}(d)) \vee \mathsf{isY}(\mathsf{S}^{\mathsf{Plus},1}(d)) \vee \cdots \vee \mathsf{isIte}(\mathsf{S}^{\mathsf{Plus},1}(d)) \tag{5}$$
$$\mathsf{isGeq}(\mathsf{S}^{\mathsf{Ite},1}(d)) \vee \mathsf{isEq}(\mathsf{S}^{\mathsf{Ite},1}(d)) \vee \mathsf{isNeg}(\mathsf{S}^{\mathsf{Ite},1}(d)) \tag{6}$$
$$\mathsf{isX}(\mathsf{S}^{\mathsf{Ite},2}(d)) \vee \mathsf{isY}(\mathsf{S}^{\mathsf{Ite},2}(d)) \vee \cdots \vee \mathsf{isIte}(\mathsf{S}^{\mathsf{Ite},2}(d)) \tag{7}$$

The solver will subsequently guess the top constructor for other subterms of d's value. These guesses are represented symbolically by *selector chains*, i.e. zero or more applications of selectors to d; for example, $\mathsf{S}^{\mathsf{Plus},1}(d)$ is a selector chain that corresponds to the first *child* of d (if we think of the value of d as a tree) when d is an application of Plus; $\mathsf{S}^{\mathsf{Plus},1}(\mathsf{S}^{\mathsf{Plus},1}(d))$ is a selector chain that corresponds to the first child of the first child of d when d and its first child are both applications of Plus; and so on.

The bottleneck in solving (3) is the large number of splitting lemmas for selector chains introduced during search which, depending on the datatypes involved, is often highly exponential. Our key observation is that datatypes generated by the SyGuS approach sketched above very often include constructors with arguments of the same

[3] For a thorough description of this approach, see [22].

type. In Example 5, both a and b have multiple constructors with arguments of type a. Using shared selectors, we can reduce the number of selectors in the example from 7 to 3 for a and from 5 to 3 for b. Moreover, *using shared selectors in selector chains makes splitting lemmas relevant in multiple contexts*. For example, a splitting lemma for a selector chain $\mathsf{S}^{\mathsf{a},1}(d)$ is relevant when d is either Plus, Minus or Ite; likewise $\mathsf{S}^{\mathsf{a},1}(\mathsf{S}^{\mathsf{a},1}(d))$ is relevant when d *and* its first child of type a are applications of either Plus, Minus or Ite. Notice that by using the decision procedure for shared selectors from Section 4, lemmas (5) and (7) would be instead both provided to the SAT engine as:

$$\mathsf{isX}(\mathsf{S}^{\mathsf{Int},1}(d)) \vee \mathsf{isY}(\mathsf{S}^{\mathsf{Int},1}(d)) \vee \cdots \vee \mathsf{isIte}(\mathsf{S}^{\mathsf{Int},1}(d))$$

Using shared selectors can lead to a reduction in the number of other kinds of lemmas as well. For instance, during synthesis cvc4 implements *symmetry breaking* techniques to avoid spending time on multiple candidates that are all equivalent in T [22, 23]. Redundant candidates are avoided by adding blocking clauses to the SAT engine that are also expressed in terms of discriminators applied to selector chains.

Example 6. Consider again the function f, grammar R, and datatypes a and b from Example 5. Assume that the solver considers X as a candidate solution for d, and later considers another candidate solution, Plus(X, Zero). Since the corresponding arithmetic terms x and $x + 0$ are equivalent in integer arithmetic, the solver infers a *lemma template* of the form:

$$\neg\mathsf{isPlus}(z) \vee \neg\mathsf{isX}(\mathsf{S}^{\mathsf{Int},1}(z)) \vee \neg\mathsf{isZero}(\mathsf{S}^{\mathsf{Int},2}(z))$$

to block a redundant candidate solution like (the one corresponding to) $x + 0$. This is achieved by instantiating the template with the substitution $\{z \mapsto d\}$ for variable z. More interestingly, z can be instantiated with other selector chains to rule out *entire families* of redundant candidate solutions. For instance, the lemma obtained with $\{z \mapsto \mathsf{S}^{\mathsf{Int},1}(d)\}$ rules out all terms that have $x + 0$ as their first child of type a, such as the terms $(x + 0) + y$, $\mathsf{ite}(x \geq y, x + 0, y)$ and $(x + 0) - 1$, which are equivalent to the smaller expressions $x + y$, $\mathsf{ite}(x \geq y, x, y)$ and $x - 1$, respectively, and hence redundant as candidate solutions. Sharing selectors allows the same blocking clause to be reused for the different constructors, whereas standard selectors would require three different clauses in this case, with $z \mapsto \mathsf{S}^{\mathsf{Plus},1}(d)$, $z \mapsto \mathsf{S}^{\mathsf{Ite},2}(d)$, and $z \mapsto \mathsf{S}^{\mathsf{Minus},1}(d)$, respectively. •

A majority of SyGuS problems can be encoded as datatypes that have significant sharing of selectors across multiple constructors, thus making the use of shared selectors particularly effective in this domain. The next section measures the impact of shared selections when solving SyGuS problems in cvc4.

6 Experiments

We implemented our calculus for the theory of datatypes with shared selectors in cvc4 Version 1.5, together with a preprocessing pass to convert standard selectors in input formulas to shared ones and other modifications to the existing decision procedure for

Family	#	Solved: sh / std	Time	SAT Decs	Terms	Sels
General	535	319 / 235 (232)	15.4 / 144.9	67k / 151k	189k / 284k	5.8 / 16.8
CLIA	73	18 / 17 (17)	25.1 / 142.4	158k / 405k	25k / 60k	9.6 / 22.2
Invariant	67	46 / 46 (46)	49.1 / 114.6	374k / 896k	37k / 61k	5.7 / 13.1
PBE_BV	750	665 / 253 (253)	27.4 / 211.9	54k / 3873k	14k / 202k	3.0 / 16.0
PBE_Strings	108	93 / 64 (64)	13.3 / 39.9	90k / 334k	14k / 41k	8.6 / 18.7

Fig. 3: Performance of cvc4 on benchmarks from five families of SyGuS Comp 2017.

datatypes, as described in Sections 3.2 and 4. We discuss here our evaluation of two configurations of cvc4, one with and one without support for shared selectors, on two different sets of benchmarks: the SyGuS benchmark suite from the 2017 SyGuS competition [4]; and a subset of the SMT-LIB [8] benchmarks containing datatype computations . Our experiments[4] were performed on the StarExec logic solving service [26].

6.1 Syntax-guided Synthesis Benchmarks

The benchmarks from the 2017 SyGuS competition are divided into five families across four tracks:

- The "General" track - problems over the theories of linear integer arithmetic (LIA) or bit-vectors.
- The conditional linear integer arithmetic track (CLIA) - problems over the theory of LIA.
- The "Invariant" synthesis track - invariant synthesis benchmarks specifying the problem with pre- and post- conditions, and a transition relation, over LIA.
- Programming by example track - consists of the bit-vector and strings subtracks, where problems are semantically constrained by examples.

We measured the impact of shared selectors by comparing for the two configurations of cvc4 the total number of solved problems and the average solving time, number of decisions performed by the SAT engine, quantifier-free terms generated, and number of selectors in the signature. Averages were computed over the set of problems solved by both configurations. We used a timeout of 30 minutes per benchmark.

A summary of the results is given in Figure 5. The first two columns show the evaluated family and the number of benchmarks in it, while the other columns present the statistics listed above, with average times expressed in seconds. The number of problems solved by both configurations is given in parentheses in the third column. The results clearly show that sharing selectors reduces the number of selectors in the signature, which generally leads to fewer terms and SAT decisions, with a positive impact on solving speed and number of problems solved. Except for the invariant family, the cvc4 configuration with shared selectors solves more problems than the one without. The impact of shared selectors is particularly significant for the bit-vector benchmark

[4] The data and details on how to reproduce our results are available at `https://cvc4.cs.stanford.edu/papers/IJCAR2018-shsel/`.

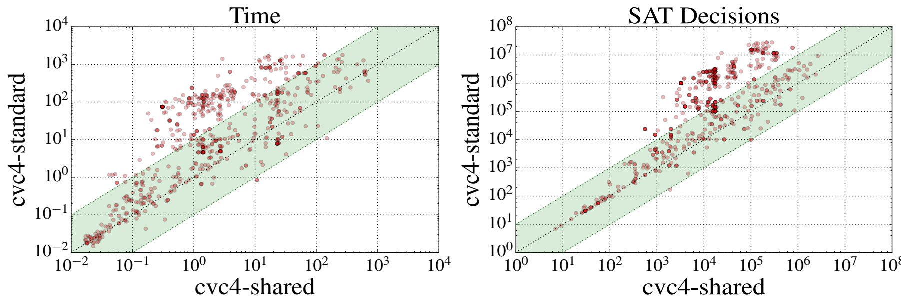

Fig. 4: Impact of shared selectors on solving time and number of SAT decisions.

Family	#	EUS	cvc4-si-sh	cvc4-si-std
General	535	**404**	391	334
CLIA	73	71	**73**	73
Invariant	67	42	**46**	46
PBE_BV	750	**739**	665	253
PBE_Strings	108	68	**93**	64

Fig. 5: Performance of cvc4 and EUSolver on benchmarks from five families of SyGuS Comp 2017.

suite (PBE_BV), with a reduction of over 80% in the average number of selectors. In that case, cvc4 is over eight times faster with shared selectors than without, solving 412 more problems, thus reducing the percentage of unsolved problems in this category from over 65% to less than 12%. Significant improvements can also be observed in the PBE_Strings and General families, with the percentages of unsolved problems being reduced from over 40% to almost 13% and from over 55% to almost 40%, respectively.

We present a per-problem comparison in the scatter plots of Figure 4, which clearly shows that for the vast majority of the benchmarks, sharing selectors reduces the number of SAT decisions and improves the solving time, often by orders of magnitude.

Comparison against other SyGuS solvers We also compared cvc4's performance with the state-of-the-art SyGuS solver EUSolver [2, 5]. For fairness, in this comparison we combine the results of the above configurations of cvc4 with its other approach for solving single-invocation synthesis problems (see [22] for details), which impacts the CLIA and General families of benchmarks. The results are summarized in Figure 5. We obtained the following results for the problems solved by EUSolver and cvc4 with and without shared selectors: 71/73/73 for CLIA, 404/391/334 for General, 42/46/46 for Invariant, 739/665/253 for PBE_BV, and 68/93/64 for PBE_Strings. These numbers show that overall cvc4 is significantly more competitive with shared selectors than without, surpassing EUSolver's performance in three of the five families.

Family	#	Solved: sh / std	Time	Decs	Terms	Sels
Leon	410	179 / 175 (175)	0.96 / 0.75	9920 / 9925	718 / 929	8.67 / 23.10
Sledgehammer	321	113 / 112 (112)	0.47 / 0.47	6949 / 6942	185 / 185	10.50 / 12.76
Nunchaku	158	67 / 67 (67)	0.49 / 0.44	7149 / 6653	1373 / 1297	6.22 / 7.22

Fig. 6: Performance of cvc4 on benchmarks from three families of SMT LIB.

6.2 Datatype benchmarks from SMT LIB

We also considered all SMT-LIB benchmarks containing datatypes. Among these, we excluded from consideration $14,387$ benchmarks that do not have any shareable selectors, as cvc4 with and without shared selectors perform the same on these benchmarks. The remaining 889 benchmarks are divided into three families:

- A set of benchmarks generated by Leon [12] (and Nunchaku and cvc4) for counterexample generation for higher-order theorem provers (AUFBVDTLIA logic). We will refer to this set as ‘Leon’.
- Benchmarks generated for verification in Isabelle [19] by the Sledgehammer tool [14] (UFDT logic). This set will be referred to as ‘Sledgehammer’.
- A benchmark set generated for higher order theorem provers by Nunchaku [21] (and Leon, and cvc4) (UFDT logic). This is the ‘Nunchaku’ set.

We summarize our results over the two configurations of cvc4, with and without shared selectors, in Figure 6, following the same schema as in Figure 5. We used a timeout of 60 seconds, since in this setting we evaluate SMT solvers as backends of verification and ITP tools, which require fast answers. The configuration with shared selectors solved at least all the benchmarks as the one without. The Leon benchmark set shows the most significant impact of sharing selectors, with a reduction of over 60% in the average number of selectors, and 4 more problems solved. It is important to remark that cvc4 employs heuristic instantiation techniques for solving these benchmarks, which can be very sensible for changes in the signature. The relevant number to measure then is the number of solved benchmarks, which either remains the same or improves with shared selectors, even if the average solving time is marginally worse.

Comparison against other SMT solvers To put the shared selector version of cvc4 in context with the state of the art, we also compared it with the only two provers that can reason about datatypes and support the SMT-LIB format: z3 [16] and Vampire [17]. On the Nunchaku and Sledgehammer benchmarks, the number of problems solved by cvc4/z3/Vampire is 67/29/30 and 113/119/138, respectively. The comparison on the Leon set excludes Vampire, since it does not support the theory of bit-vectors; the split between cvc4 and z3 is 179/173 on that set. The results show that cvc4 compares favorably with the other tools.

7 Related Work

The motivation of our work is to reduce the number of terms considered by a decision procedure for the theory of algebraic of datatypes, based on procedures introduced in previous work [10, 20]. Thus, our contributions apply to other systems that handle datatypes semantically, such as SMBC [15] and the SMT solver z3 [16]. On systems that reason about datatypes axiomatically, such as the first-order theorem prover and SMT solver Vampire [17], and the higher-order systems Isabelle [19] and Dafny [18], whether to share selectors and how to handle them is simply a matter of axiomatizing the datatypes theory accordingly. For example, the axiomatization in Vampire avoids selectors altogether [17, Sect. 4.3], while in Isabelle users are encouraged to write specifications directly with shared selectors [13, Sec. 3].

Most SyGuS solvers employ a variation of counter-example guided inductive synthesis (CEGIS), introduced by Solar-Lezama [25]. While CVC4 benefits from sharing selectors by representing syntax restrictions with datatypes, other systems use an outer layer with an underlying reasoning engine, for instance using an SMT solver to verify the correctness of candidate solutions, but not for performing the enumerative search [5].

8 Conclusion

We have presented an extension of the theory of algebraic datatypes that adds shared selectors. We have discussed and proved correct a calculus for deciding the constraint satisfiability problem in the new theory. Moreover, we have described how algebraic datatypes can be leveraged in an SMT solver to solve syntax-guided synthesis problems and explained how the use of shared selectors in this setting can lead to significant performance gains. Our experiments demonstrate that an implementation of the new calculus in the CVC4 solver significantly enhances its performance on syntax-guided synthesis problems and is responsible for making CVC4 the best known solver for certain classes of problems.

In future work, we plan to generalize our approach so that distinct selector *chains* can be compressed to a single application of the same selector symbol. This requires more sophisticated criteria for recognizing when two selector chains for a datatype cannot be simultaneously constrained for arbitrary values of that datatype. We believe that this further extension can be done in a manner similar to the one presented here and expect that this will lead to further performance improvements.

References

1. R. Alur, R. Bodík, G. Juniwal, M. M. K. Martin, M. Raghothaman, S. A. Seshia, R. Singh, A. Solar-Lezama, E. Torlak, and A. Udupa. Syntax-guided synthesis. In *FMCAD*, pages 1–8. IEEE, 2013.
2. R. Alur, P. Cerný, and A. Radhakrishna. Synthesis through unification. In *CAV*, volume 9207 of *LNCS*, pages 163–179. Springer, 2015.
3. R. Alur, D. Fisman, R. Singh, and A. Solar-Lezama. Sygus-comp 2017: Results and analysis. In D. Fisman and S. Jacobs, editors, *Proceedings Sixth Workshop on Synthesis (SYNT)*, volume 260 of *EPTCS*, pages 97–115, 2017.
4. R. Alur, D. Fisman, R. Singh, and A. Solar-Lezama. Sygus-comp 2017: Results and analysis. *CoRR*, abs/1711.11438, 2017.
5. R. Alur, A. Radhakrishna, and A. Udupa. Scaling enumerative program synthesis via divide and conquer. In *TACAS*, volume 10205 of *LNCS*, pages 319–336, 2017.

6. F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, New York, NY, USA, 1998.
7. C. Barrett, C. L. Conway, M. Deters, L. Hadarean, D. Jovanović, T. King, A. Reynolds, and C. Tinelli. CVC4. In *CAV*, pages 171–177. Springer, 2011.
8. C. Barrett, P. Fontaine, and C. Tinelli. The Satisfiability Modulo Theories Library (SMT-LIB). `www.SMT-LIB.org`, 2016.
9. C. Barrett, P. Fontaine, and C. Tinelli. The SMT-LIB Standard: Version 2.6. Technical report, Department of Computer Science, The University of Iowa, 2017.
10. C. Barrett, I. Shikanian, and C. Tinelli. An abstract decision procedure for a theory of inductive data types. *JSAT*, 3(1-2):21–46, 2007.
11. C. Barrett and C. Tinelli. Satisfiability modulo theories. In *Handbook of Model Checking*. Springer, 2014. (to appear).
12. R. Blanc, V. Kuncak, E. Kneuss, and P. Suter. An overview of the leon verification system: Verification by translation to recursive functions. In *Proceedings of the 4th Workshop on Scala*, pages 1:1–1:10. ACM, 2013.
13. J. C. Blanchette, J. Hölzl, A. Lochbihler, L. Panny, A. Popescu, and D. Traytel. Truly modular (co)datatypes for isabelle/hol. In *ITP*, volume 8558 of *LNCS*, pages 93–110. Springer, 2014.
14. S. Böhme and T. Nipkow. Sledgehammer: Judgement day. In *IJCAR*, volume 6173 of *LNCS*, pages 107–121. Springer-Verlag, 2010.
15. S. Cruanes. Satisfiability modulo bounded checking. In *CADE*, volume 10395 of *LNCS*, pages 114–129. Springer, 2017.
16. L. de Moura and N. Bjørner. Z3: An Efficient SMT Solver. In *TACAS*, volume 4963 of *LNCS*, pages 337–340. Springer, 2008.
17. L. Kovács, S. Robillard, and A. Voronkov. Coming to terms with quantified reasoning. In *POPL*, pages 260–270. ACM, 2017.
18. K. R. M. Leino. Developing verified programs with dafny. *Ada Lett.*, 32(3):9–10, Dec. 2012.
19. T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL - A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.
20. A. Reynolds and J. C. Blanchette. A decision procedure for (co)datatypes in smt solvers. *Journal of Automated Reasoning*, 58(3):341–362, Mar 2017.
21. A. Reynolds, J. C. Blanchette, S. Cruanes, and C. Tinelli. Model finding for recursive functions in SMT. In *IJCAR*, volume 9706 of *LNCS*, pages 133–151. Springer, 2016.
22. A. Reynolds, V. Kuncak, C. Tinelli, C. Barrett, and M. Deters. Refutation-based synthesis in smt. *Formal Methods in System Design*, Feb 2017.
23. A. Reynolds and C. Tinelli. Sygus techniques in the core of an smt solver. *arXiv preprint arXiv:1711.10641*, 2017.
24. A. Reynolds, A. Viswanathan, H. Barbosa, C. Tinelli, and C. Barrett. Datatypes with Shared Selectors. Technical report, The University of Iowa, 2018. `http://cvc4.cs.stanford.edu/papers/IJCAR2018-shsel/`.
25. A. Solar-Lezama, L. Tancau, R. Bodík, S. A. Seshia, and V. A. Saraswat. Combinatorial sketching for finite programs. In *ASPLOS*, pages 404–415. ACM, 2006.
26. A. Stump, G. Sutcliffe, and C. Tinelli. Starexec: A cross-community infrastructure for logic solving. In *IJCAR*, volume 8562 of *LNCS*, pages 367–373. Springer, 2014.