Combined Satisfiability Modulo Parametric Theories

Sava Krstić[1], Amit Goel[1], Jim Grundy[1], and Cesare Tinelli[2]

[1] Strategic CAD Labs, Intel Corporation
[2] Department of Computer Science, The University of Iowa

Abstract. We give a fresh theoretical foundation for designing comprehensive SMT solvers, generalizing in a practically motivated direction. We define *parametric theories* that most appropriately express the "logic" of common data types. Our main result is a combination theorem for decision procedures for disjoint theories of this kind. Virtually all of the deeply nested data structures (lists of arrays of sets of ...) that arise in verification work are covered.

1 Introduction

Formal methods for hardware or software development require checking validity (or, dually, satisfiability) of formulas in logical theories modeling relevant datatypes. Satisfiability procedures have been devised for the basic ones—reals, integers, arrays, lists, tuples, queues, and so on—especially when restricted to formulas in some some quantifier-free fragment of first-order logic. Thanks to a seminal result by Nelson and Oppen [11], these basic procedures can often be modularly combined to cover formulas that mingle several datatypes.

Most research on *Satisfiability Modulo Theories (SMT)* has traditionally used classical first-order logic as a foundation for defining the language of satisfiability procedures, or *SMT solvers*, and reasoning about their correctness. However, the untypedness of this most familiar logic is a major limitation. It unnecessarily complicates correctness arguments for combination methods and restricts the applicability of sufficient conditions for their completeness. Thus, researchers have recently begun to frame SMT problems directly in terms of richer typed logics and to develop combination results for these logics [21,4,24,15,3,6]. Ahead of the theory, solvers supporting the PVS system [19], solvers of the CVC family [2], and some others adopted a typed setting early on.

The SMT-LIB initiative, an international effort aimed at developing common standards for the SMT field, proposes a version of many-sorted first-order logic as an initial underlying logic for SMT [16]. We see this as a step in the right direction, but only the first one, because the many-sorted logic's rudimentary type system is still inadequate for describing and working with typical cases of combined theories and their solvers. For example, in this logic one can define a generic theory of lists using a sort List for the lists and the sort E for the list elements. Then, a theory of integer lists can be defined formally as the union

O. Grumberg and M. Huth (Eds.): TACAS 2007, LNCS 4424, pp. 602–617, 2007.

of the list theory with the integer theory, modulo the identification of the sort E with the integer sort of the second theory. This combination mechanism gets quickly out of hand if we want to reason about, say, lists of arrays of lists of integers, and it cannot be used at all to specify *arbitrarily* nested lists. Because of the frequent occurrence of such combined datatypes in verification practice, this is a serious shortcoming.

Fortunately, virtually all structured datatypes arising in formal methods are *parametric*, the way arrays or lists are. Combined datatypes like those mentioned above are constructed simply by parameter instantiation. For this reason, we believe that any logic for SMT should directly support parametric types and, consequently, parametric polymorphism. The goal of this paper is to provide a Nelson-Oppen-style framework and results for theories combinable by parameter instantiation.

The key concept of *parametric theory* can likely be defined in various logics with polymorphic types. We adopt the higher-order logic of the theorem provers *HOL* [7], *HOL Light* [9], and *Isabelle/HOL* [14]. It is well studied and widely used, and has an elegant syntax and intuitive set-theoretic semantics.

Integration of SMT solvers with other reasoning tools, in particular with interactive provers, is a topic of independent interest [5,1] with a host of issues, including language compatibility [8]. This paper contributes a solid theoretical foundation for the design of *HOL*-friendly SMT solvers.

Finally, a striking outcome of this work is that in practically oriented (that is, dealing with common datatypes) SMT research, the vexatious stable infiniteness condition of the traditional Nelson-Oppen approach does not need to be mentioned. Its role is played by a milder flexibility condition that, by our results, is automatically satisfied for all *fully parametric* theories.

Related Work. Observations that the congruence closure algorithm of [12] effectively translates a first-order goal into *HOL* via currying, and that the solver for algebraic datatypes of [3] actually works for lists of lists and the like, were key to the unveiling of parametric *HOL* theories.

Like all other work on combining SMT solvers for disjoint theories, from [11] on, our approach is based on inter-solver propagation of constraints over a common language. Similarly to [22], the constraints also involve cardinalities, so our method can manage both infinite and finite datatypes. The purification procedure that transforms the input query in the mixed language of several solvers into pure parts is more involved here than anywhere else because of the complexity brought by the rich type system.

We give model-theretical correctness arguments, analogous to those used in other modern treatments of Nelson-Oppen combination, from [18,20] to the recent work [6] which also tackles some non-disjoint combinations. However, in the completeness proof, we rely on the parametricity of the types modeled by the component theories, not on the theories' stable infiniteness. This difference has important practical consequences. While our results do not subsume existing results nor are subsumed by them, they apply more widely because most of the datatypes relevant in applications are described by theories that satisfy our

parametricity requirements without necessarily satisfying the stable infiniteness requirements of other combination methods.

In this, our approach is closely related to the recent work of Ranise *et al.* [15]. They present an extension of the Nelson-Oppen method in which a many-sorted theory S modeling a data structure like lists or arrays can be combined with an arbitrary theory T modeling the elements of the data structure. The theory S is required to satisfy a technical condition ("politeness") for each element sort. This corresponds to our requiring that the data structure be a parametric type with flexibility conditions. (More specifically, the "smoothness" and "finite witnessability" parts of politeness correspond to our up-flexibility and down-flexibility, the latter being significantly weaker than its counterpart in [15].) The results in [15] can be extended in principle to more than two theories by incremental pairwise combinations. However, as we argued, many-sorted logic is not well-suited for working with elaborate combinations of theories, while in a logic with parametric types such combinations are straightforward. In particular, our main result about combination of multiple pairwise disjoint parametric theories, would be difficult even to state in the language of [15]. Yet, the important insight that it is parametricity and not stable infiniteness that justifies Nelson-Oppen cooperation of common solvers is already in [15]; we have given it full expression.

Outline. In Section 2, reviewing the standard *HOL* material, we define the syntactic concept of signatures, and their semantic counterpart, structures. In Section 3, we introduce the crucial (fully) parametric structures, which are essentially collections of polymorphic constants with uniform behavior specified by relational parametricity. In Section 4, we discuss satisfiability in parametric structures and a process that corresponds to the familiar reduction of satisfiability of arbitrary quantifier-free formulas to sets of literals. In Section 5, we describe the algorithm for combining solvers and identify conditions under which it is complete. All proofs, omitted for lack of space, can be found in the accompanying technical report [10].

2 Syntax and Semantics of Higher Order Logic

We give a brief account of the standard syntax and semantics of higher-order logic, similar to that given by Pitts for the logic of the *HOL* theorem prover [7]. Much of it has been formalized by Harrison in a "*HOL* in *HOL*" fashion [9].

2.1 Syntax of *HOL* Types and Terms

The syntactic world of *HOL* TYPES is built using TYPE OPERATORS and TYPE VARIABLES. Each type operator has a non-negative ARITY. Given a set O of type operators, the set Type_O is the smallest set containing all type variables and expressions of the form $F(\sigma_1, \ldots, \sigma_n)$, where $F \in O$ has arity n and $\sigma_i \in \mathsf{Type}_O$. The set of type variables occurring in σ will be denoted $\mathtt{tyvar}(\sigma)$.

A TYPE INSTANTIATION is a finite map from type variables to types. The notation $[\sigma_1/\alpha_1, \ldots, \sigma_n/\alpha_n]$ is for the finite map that takes $\alpha_1, \ldots, \alpha_n$ to $\sigma_1, \ldots, \sigma_n$.

For any type σ and type instantiation θ, $\theta(\sigma)$ denotes the simultaneous substitution of every occurrence of α_i in σ with σ_i. We say that τ is an INSTANCE of σ and write $\tau \preceq \sigma$ if there is some θ such that $\tau = \theta(\sigma)$. Clearly, $\theta(\sigma) = \theta'(\sigma)$ holds if and only if θ and θ' agree on $\mathtt{tyvar}(\sigma)$. Thus, if $\tau \preceq \sigma$, then there is a unique minimal type instantiation that maps σ to τ; its domain is $\mathtt{tyvar}(\sigma)$ and it will be denoted $[\tau /\!\!/ \sigma]$.

A *HOL* SIGNATURE $\Sigma = \langle O \,|\, K \rangle$ consists of a set O of type operators and a set K of TYPED CONSTANTS. Each constant $k^\sigma \in K$ is a pair of a symbol k and a type $\sigma \in \mathsf{Type}_O$, with no two constants sharing the same symbol. Let K^+ be the set of all pairs (also called constants) k^τ where $k^\sigma \in K$ and $\tau \preceq \sigma$.

The standard boolean connectives and equality make up the signature Σ_{Eq}:[1]

$$\Sigma_{\mathsf{Eq}} = \langle \mathsf{Bool}, \Rightarrow \mid =^{\alpha^2 \Rightarrow \mathsf{Bool}}, \mathsf{true}^{\mathsf{Bool}}, \mathsf{false}^{\mathsf{Bool}}, \neg^{\mathsf{Bool} \Rightarrow \mathsf{Bool}}, \wedge^{\mathsf{Bool}^2 \Rightarrow \mathsf{Bool}}, \ldots \rangle$$

The constants of Σ_{Eq} will be called LOGICAL. From now on we will assume that every signature we consider includes Σ_{Eq}. When—as in the following examples—we write a concrete signature $\Sigma = \langle O \,|\, K \rangle$, we will tacitly assume that the Σ_{Eq}-part is there, even if it is not explicitly shown.

Example 1. Here are some familiar signatures.

$$\Sigma_{\mathsf{Int}} = \langle \mathsf{Int} \mid 0^{\mathsf{Int}}, 1^{\mathsf{Int}}, (-1)^{\mathsf{Int}}, \ldots, +^{\mathsf{Int}^2 \Rightarrow \mathsf{Int}}, -^{\mathsf{Int}^2 \Rightarrow \mathsf{Int}}, \times^{\mathsf{Int}^2 \Rightarrow \mathsf{Int}}, \leq^{\mathsf{Int}^2 \Rightarrow \mathsf{Bool}}, \ldots \rangle$$

$$\Sigma_{\mathsf{Array}} = \langle \mathsf{Array} \mid \mathsf{mk\_arr}^{\beta \Rightarrow \mathsf{Array}(\alpha,\beta)}, \mathsf{read}^{[\mathsf{Array}(\alpha,\beta),\alpha] \Rightarrow \beta}, \mathsf{write}^{[\mathsf{Array}(\alpha,\beta),\alpha,\beta] \Rightarrow \mathsf{Array}(\alpha,\beta)} \rangle$$

$$\Sigma_{\mathsf{List}} = \langle \mathsf{List} \mid \mathsf{cons}^{[\alpha,\mathsf{List}(\alpha)] \Rightarrow \mathsf{List}(\alpha)}, \mathsf{nil}^{\mathsf{List}(\alpha)}, \mathsf{head}^{\mathsf{List}(\alpha) \Rightarrow \alpha}, \mathsf{tail}^{\mathsf{List}(\alpha) \Rightarrow \mathsf{List}(\alpha)} \rangle$$

$$\Sigma_{\mathsf{Monoid}} = \langle \mathsf{Monoid} \mid 1^{\mathsf{Monoid}}, *^{\mathsf{Monoid}^2 \Rightarrow \mathsf{Monoid}} \rangle$$

The ARITY of a constant $k^\sigma \in K$ is the number m from the unique expression of σ in the form $[\sigma_1, \ldots, \sigma_m] \Rightarrow \tau$, where τ is not a function type. If all σ_i are non-function types too, we will say that the constant is ALGEBRAIC. All signatures in Example 1 are algebraic in the sense that all their constants are such.

The set Term_Σ of *HOL* TERMS over a signature Σ is defined by the rules in Figure 1. The four rules classify terms into VARIABLES, CONSTANTS, APPLICATIONS, and ABSTRACTIONS. The rules actually define the set of term-type pairs $M\!:\!\sigma$, which we read as "term M has type σ". By structural induction, every term has a unique type. Non-typeable expressions like $v^\sigma u^\sigma$ are not considered to be terms at all.

Each occurrence of a variable in a term is FREE or BOUND, by the usual inductive definition. We regard two terms M and N as equal if they are equal up to renaming of bound variables. The set of free variables occurring in M is denoted $\mathtt{var}(M)$. We define $\mathtt{tyvar}(M)$ to be the set of type variables occurring in the type of any variable or constant subterm of M.

2.2 Semantics of Types

Type operators of arity n are interpreted as n-ary set operations—functions $\mathcal{U}^n \to \mathcal{U}$, where $\mathcal{U}$ is a suitably large universe of sets. Fixing such an

[1] By convention, $[\alpha^2, \beta] \Rightarrow \gamma$ is $\alpha \Rightarrow \alpha \Rightarrow \beta \Rightarrow \gamma$, and $\Rightarrow$ associates to the right.

$$\frac{}{v^\sigma : \sigma} \qquad \frac{k^\tau \in K^+}{k^\tau : \tau} \qquad \frac{M : \sigma \Rightarrow \tau \quad N : \sigma}{M\,N : \tau} \qquad \frac{M : \tau}{\lambda v^\sigma . M : \sigma \Rightarrow \tau}$$

Fig. 1. Typing rules for *HOL* terms

interpretation that associates with every $F \in O$ a set operation $[F]$, we define the INTERPRETATION OF $\sigma \in \mathsf{Type}_O$ in Figure 2. The interpretation of a type σ in a TYPE ENVIRONMENT ι—a finite map from type variables to $\mathcal{U}$ —is a set $[\![\sigma]\!]\,\iota$, "the meaning of σ in ι". The set $[\![\sigma]\!]\,\iota$ is defined when $\mathtt{tyvar}(\sigma) \subseteq \mathtt{dom}(\iota)$ and will be unchanged if ι is replaced with ι' as long as ι and ι' agree on $\mathtt{tyvar}(\sigma)$. (Here and in what follows, $\mathtt{dom}$ is used to denote the domain of a finite map.)

$$[\![\alpha]\!]\,\iota = \iota(\alpha) \quad \text{for every } \alpha \in \mathtt{dom}(\iota)$$
$$[\![F(\sigma_1, \ldots, \sigma_n)]\!]\,\iota = [F]\big([\![\sigma_1]\!]\,\iota, \ldots, [\![\sigma_n]\!]\,\iota\big)$$

Fig. 2. Interpretation of *HOL* types

Common type operators usually come with a unique intended interpretation, so it becomes awkward to make a notational distinction between F and $[F]$. But, for the sake of clarity, we will distinguish syntax from the semantics. For constant types (0-ary type operators) Unit, Bool and Int we will use $[\mathsf{Unit}] = \mathbb{U} = \{*\}$, $[\mathsf{Bool}] = \mathbb{B} = \{\mathit{true}, \mathit{false}\}$ and $[\mathsf{Int}] = \mathbb{Z}$. The symbols $\Rightarrow$ and $\mathbf{\Rightarrow}$ will be used for the syntactic type operator and the full function space set operation it represents; that is, we have $[\Rightarrow] = \mathbf{\Rightarrow}$. Similar convention holds for the Cartesian product and disjoint sum operators $\times$ and $+$, and operations $\mathbf{\times}$, $\mathbf{+}$. The unary type operator List is interpreted as the set operation $\mathbf{List}$, where $\mathbf{List}(A)$ is the set of finite lists with elements in the set A.

The meaning of an instantiated type in some environment is the same as that of the original type in an appropriately updated environment. Precisely, if $\tau = \theta(\sigma)$, then $[\![\tau]\!]\,\iota = [\![\sigma]\!]\,\iota'$, where ι' is defined by $\iota'(\alpha) = [\![\theta(\alpha)]\!]\,\iota$. The environment ι' will be denoted $\theta{\cdot}\iota$. (See Figure 3 for its use.) For example, if $\sigma = (\alpha \Rightarrow \beta)$, $\tau = (\gamma \Rightarrow \gamma \Rightarrow \mathsf{Bool})$, and $\iota = [X/\gamma]$, then $\iota' = [X/\alpha, (X \mathbf{\Rightarrow} \mathbb{B})/\beta]$.

2.3 Semantics of Terms

Suppose now an interpretation $[\![\sigma]\!]$ for $\sigma \in \mathsf{Type}_O$ is given as in Section 2.2. We define an INDEXED ELEMENT of $[\![\sigma]\!]$ to be a family of elements $a\,\iota$ indexed by type environments ι whose domains contain $\mathtt{tyvar}(\sigma)$; the requirements are that $a\,\iota \in [\![\sigma]\!]\,\iota$ and and that $a\,\iota = a\,\iota'$ when ι and ι' agree on $\mathtt{tyvar}(\sigma)$. For example, the list length function *len* is an indexed element of $[\![\mathsf{List}(\alpha) \Rightarrow \mathsf{Int}]\!]$; for every ι with $\iota(\alpha) = A$, $\mathit{len}\,\iota$ is the concrete length function len_A, an element of $\mathbf{List}(A) \mathbf{\Rightarrow} \mathbb{Z}$. Similarly, the identity function is an indexed element of $[\![\alpha \Rightarrow \alpha]\!]$, but note that there are no "natural" indexed elements of $[\![\alpha \Rightarrow \beta]\!]$.

Given an arbitrary signature $\Sigma = \langle O \mid K \rangle$, a Σ-STRUCTURE $\mathcal{S}$ consists of

- an arity-respecting assignment $\mathcal{S}_{\mathsf{typeop}}$ that maps every F in O to a set operation $[F]$, as in Section 2.2;
- an assignment $\mathcal{S}_{\mathsf{const}}$ of an indexed element $[k^\sigma]$ of $[\![\sigma]\!]$ to every k^σ in K.[2]

We stipulate that the type operators Bool and $\Rightarrow$, as well as boolean connectives and the equality predicate be always assigned their standard meanings. For example, $[\wedge^{\mathsf{Bool}^2 \Rightarrow \mathsf{Bool}}]\,\iota$ is the conjunction operation on booleans for all type environments ι. Also, $[=^{\alpha^2 \Rightarrow \mathsf{Bool}}]\,\iota$ is always the identity relation on the set $\iota(\alpha)$. In other words, there is only one Σ_{Eq}-structure we care about, and it is "part of" all Σ-structures that include it.

Example 2. For signatures associated with datatypes, we normally associate a unique structure. Referring to Example 1, this is clear for Σ_{Int}. For Σ_{Array}, we define $[\mathsf{Array}](X, Y)$ to be the set of functions from X to Y that give the same result for all but finitely many arguments; the interpretation of the constants is obvious. For Σ_{List} there is an issue with partiality of head and tail, which can be resolved, for example, by defining $[\mathsf{head}^{\mathsf{List}(\alpha) \Rightarrow \alpha}]\,\iota$ to be an arbitrary element of $\iota(\alpha)$. (See Example 5 below for better solutions.) Unlike these examples, there are multiple Σ_{Monoid}-structures of interest; every monoid gives us one.

Interpretation of terms requires two environments: one for type variables and one for the free term variables. For example, the meaning of the Σ_{Eq}-term $\lambda u^{\alpha \Rightarrow \beta}.\, u^{\alpha \Rightarrow \beta} v^\alpha$ in the pair of environments $\langle [\mathbb{Z}/\alpha, \mathbb{Z}/\beta],\ [0/v^\alpha] \rangle$ is the function that maps its argument $f \in (\mathbb{Z} \Rightarrow \mathbb{Z})$ to $f(0)$. To make this precise, define first, for a given type environment ι, a TERM ENVIRONMENT OVER ι to be any finite map that associates to each variable v^σ in its domain an element of the set $[\![\sigma]\!]\,\iota$. Then, for any term M, an ENVIRONMENT FOR M is a pair $\langle \iota, \rho \rangle$, where ι is a type environment such that $\mathtt{tyvar}(M) \subseteq \mathtt{dom}(\iota)$ and ρ is a term environment over ι such that $\mathtt{var}(M) \subseteq \mathtt{dom}(\rho)$.

Given a Σ-structure, a Σ-term M of type σ, and an environment $\langle \iota, \rho \rangle$ for M, the INTERPRETATION OF M is an element $[\![M]\!]\langle \iota, \rho \rangle$ of the set $[\![\sigma]\!]\,\iota$ defined inductively by the equations in Figure 3. The interpretation of a variable v^τ is found by consulting the term environment ρ. To interpret a constant k^τ, which must be an instance of a unique $k^\sigma \in K$, we transform ι from a type environment for τ to the type environment $[\tau /\!\!/ \sigma] \cdot \iota$ for σ (see the last paragraph of Section 2.2), whereupon we can find the interpretation for k^τ using the function $[k^\sigma]$ supplied by the Σ-structure. The interpretations of applications and abstractions are straightforward. The notation $\rho[v^\sigma \mapsto x]$ is for the environment that maps v^σ to x, and is otherwise equal to ρ. It is easy to check that $[\![M]\!]\langle \iota, \rho \rangle$ is determined by the restriction of ι and ρ to $\mathtt{tyvar}(M)$ and $\mathtt{var}(M)$ respectively.

[2] The proper notation would be $[F]_{\mathcal{S}}, [k^\sigma]_{\mathcal{S}}, [\![\sigma]\!]_{\mathcal{S}}$, but the structure $\mathcal{S}$ will always be understood from the context.

$$[\![v^\tau]\!]\langle\iota,\rho\rangle = \rho(v^\tau) \qquad [\![M\,N]\!]\langle\iota,\rho\rangle = \big([\![M]\!]\langle\iota,\rho\rangle\big)\big([\![N]\!]\langle\iota,\rho\rangle\big)$$

$$[\![k^\tau]\!]\langle\iota,\rho\rangle = [k^\sigma](\lceil\tau/\!\!/\sigma\rceil\cdot\iota) \qquad [\![\lambda v^\sigma.\, M]\!]\langle\iota,\rho\rangle = \boldsymbol{\lambda} x \in [\![\sigma]\!]\,\iota.\, [\![M]\!]\langle\iota,\rho[v^\sigma\mapsto x]\rangle$$

Fig. 3. Interpretation of *HOL* terms

3 Parametric Structures

The uniformity exhibited by commonly used polymorphic type operators and constants is not captured by the semantics in Section 2, but has been formalized by the notion of *relational parametricity* [17,23]. It leads us to the concept of *fully parametric structures* and gives us powerful techniques, based on the *Abstraction Theorem* [17] to reason about them. See [10] for full statements and proofs of results needed in this paper.

3.1 Relational Semantics

A PARAMETRIC SET OPERATION is a pair consisting of a set operation G and an operation $G^\sharp$ on relations such that if $R_1\colon A_1 \leftrightarrow B_1, \ldots, R_n\colon A_n \leftrightarrow B_n$, then $G^\sharp(R_1,\ldots,R_n)\colon G(A_1,\ldots,A_n) \leftrightarrow G(B_1,\ldots,B_n)$. It is also required that $G^\sharp$ be functorial on bijections: $G^\sharp(R_1,\ldots,R_n)$ must be a bijection if the R_i are all bijections, and the identities $G^\sharp(R_1,\ldots,R_n)\circ G^\sharp(S_1,\ldots,S_n) = G^\sharp(R_1\circ S_1,\ldots,R_n\circ S_n)$ and $G^\sharp(id_{A_1},\ldots,id_{A_n}) = id_{G(A_1,\ldots,A_n)}$ must hold, where $R_i\colon A_i \leftrightarrow B_i$ and $S_i\colon B_i \leftrightarrow C_i$ are arbitrary bijections and id_A denotes the identity relation on A. Note that the conditions are meaningful when $n = 0$: every set G together with $G^\sharp = id_G$ is a parametric 0-ary set operation.

Informally, we will say that a set operation G is parametric if there is a $G^\sharp$ such that $(G, G^\sharp)$ is a parametric set operation.

Example 3. List is parametric: for a given relation $R\colon A \leftrightarrow B$, the relation $\mathsf{List}^\sharp(R)\colon \mathsf{List}(A) \leftrightarrow \mathsf{List}(B)$ is the generalization of the familiar *map* function. The binary set operations $\boldsymbol{\times}$ and $\boldsymbol{\Rightarrow}$ are also parametric: given relations $R_1\colon A_1 \leftrightarrow B_1$ and $R_2\colon A_2 \leftrightarrow B_2$, the relation $R_1 \boldsymbol{\times}^\sharp R_2\colon A_1 \boldsymbol{\times} A_2 \leftrightarrow B_1 \boldsymbol{\times} B_2$ relates $\langle x_1, x_2\rangle$ with $\langle y_1, y_2\rangle$ iff $\langle x_1, y_1\rangle \in R_1$ and $\langle x_2, y_2\rangle \in R_2$; the relation $R_1 \boldsymbol{\Rightarrow}^\sharp R_2\colon (A_1 \boldsymbol{\Rightarrow} B_1) \leftrightarrow (A_2 \boldsymbol{\Rightarrow} B_2)$ relates f_1 with f_2 iff for every x_1, x_2, $\langle x_1, x_2\rangle \in R_1$ implies $\langle f_1(x_1), f_2(x_2)\rangle \in R_2$.

Let ι_1 and ι_2 be two type environments with equal domains. An ENVIRONMENT RELATION $R\colon \iota_1 \leftrightarrow \iota_2$ is a collection of relations $R(\alpha)\colon \iota_1(\alpha) \leftrightarrow \iota_2(\alpha)$, for each α in the domain of ι_1 and ι_2. The identity relation $id_\iota\colon \iota \leftrightarrow \iota$ is defined by $id_\iota(\alpha) = id_{\iota(\alpha)}$.

Suppose O is a set of type operators, and that for each $F \in O$ the set operation $[F]$ is parametric, with the relational part denoted $[F]^\sharp$. Then for any type σ and a relation $R\colon \iota_1 \leftrightarrow \iota_2$ between type environments whose domain contains $\mathtt{tyvar}(\sigma)$, there is an induced relation $[\![\sigma]\!]^\sharp\, R\colon [\![\sigma]\!]\,\iota_1 \leftrightarrow [\![\sigma]\!]\,\iota_2$, defined in Figure 4. It is easy to prove that $[\![\sigma]\!]^\sharp\, id_\iota = id_{[\![\sigma]\!]\,\iota}$ holds for every σ, the result known as *Identity Extension Lemma* [17].

$$[\![\alpha]\!]^\sharp\, R = R(\alpha)$$
$$[\![F(\sigma_1, \ldots, \sigma_n)]\!]^\sharp\, R = [F]^\sharp([\![\sigma_1]\!]^\sharp\, R, \ldots, [\![\sigma_n]\!]^\sharp\, R)$$

Fig. 4. Relational type semantics

An indexed element a of $[\![\sigma]\!]$ is called PARAMETRIC if

$$\langle a\,\iota_1, a\,\iota_2\rangle \in [\![\sigma]\!]^\sharp\, R \quad \text{for every relation } R\colon \iota_1 \leftrightarrow \iota_2. \tag{1}$$

Example 4. Let us check that *len* is a parametric indexed element of $[\![\mathsf{List}(\alpha) \Rightarrow \mathsf{Int}]\!]$. Pick a relation $R\colon [A/\alpha] \leftrightarrow [B/\alpha]$ between type environments, i.e., $R(\alpha)$ is some relation $r\colon A \leftrightarrow B$. By definition $len\,[A/\alpha]$ is the concrete length function $len_A \in \mathbf{List}(A) \mathbin{\boldsymbol{\Rightarrow}} \mathbb{Z}$; and similarly $len\,[B/\alpha] = len_B$. To verify the condition (1), we need to check that $\langle len_A, len_B\rangle \in [\![\mathsf{List}(\alpha) \Rightarrow \mathsf{Int}]\!]^\sharp\, R$. By the equations in Figure 4, the relation on the right is equal to $map(r) \mathbin{\boldsymbol{\Rightarrow}}^\sharp id_{\mathbb{Z}}$. By the definition of $\mathbin{\boldsymbol{\Rightarrow}}^\sharp$, we need to check that for every $x \in \mathbf{List}(A), y \in \mathbf{List}(B)$ such that $\langle x, y\rangle \in map(r)$ one must have $len_A(x) = len_B(y)$—which is true.

Example 5. Standard interpretations of constants in Σ_{List} and Σ_{Array} are parametric, except for the partiality of head and tail. This can be fixed by giving head the type $\mathsf{List}\,\alpha \Rightarrow \alpha + \mathsf{Unit}$ or $\mathsf{List}\,\alpha \Rightarrow \alpha \Rightarrow \mathsf{Bool}$, and similarly for tail.

3.2 Fully Parametric Structures

Polymorphic equality is not parametric! Indeed, given $R\colon A \leftrightarrow B$, condition (1) says: if $\langle x, y\rangle, \langle x', y'\rangle \in R$, then $(x =_A x') \Leftrightarrow (y =_B y')$ [23]. This condition is not true in general, but holds if and only if R is a partial bijection. To account for this limited parametricity of equality, we define a set operation G to be FULLY PARAMETRIC if $G^\sharp$ is functorial on partial bijections. We also define an indexed element a to be FULLY PARAMETRIC if (1) holds for all partial bijections R. (Thus, to specify a fully parametric set operation G, one need define $G^\sharp(R_1, \ldots, R_n)$ only for the case where all the R_i are partial bijections.)

Note that the "Reynolds parametricity" defined in Section 3.1 and full parametricity are incomparable: to get from the former to the latter, we strengthened the functoriality condition and weakened the condition (1) on elements.

The following definition is crucial. An $\langle O \mid K\rangle$-structure $\mathcal{S}$ is FULLY PARAMETRIC if $\mathcal{S}_{\mathsf{typeop}}(F)$ is a fully parametric set operation for every $F \in O - \{\Rightarrow\}$ and $\mathcal{S}_{\mathsf{const}}(k^\sigma)$ is a fully parametric indexed element for every $k^\sigma \in K$.

The function space operation $\mathbin{\boldsymbol{\Rightarrow}}$ is not fully parametric; for example, if $R\colon A \to A'$ is an injection, then $R \mathbin{\boldsymbol{\Rightarrow}}^\sharp id_B\colon (A \mathbin{\boldsymbol{\Rightarrow}} B) \leftrightarrow (A' \mathbin{\boldsymbol{\Rightarrow}} B)$ is not a partial bijection. Fortunately, this is an exception.

Lemma 1. *The structures corresponding to the following datatypes are fully parametric: datatypes with 0-ary type constructors (such as* Bool, Int, *etc.); all algebraic datatypes (including sums, products, lists); arrays; sets; and multisets.*

In Section 5, we will see that full parametricity legitimizes structures' participation in the Nelson-Oppen combination algorithm.

4 *HOL* Theories and Satisfiability

In *HOL*, FORMULAS are simply terms of type Bool. If ϕ is a Σ-formula, $\mathcal{S}$ is a Σ-structure, and $e = \langle \iota, \rho \rangle$ is an environment for ϕ, we write $e \models \phi$ as an abbreviation for $[\![\phi]\!]\, e = \mathit{true}$. We say that ϕ is $\mathcal{S}$-SATISFIABLE if $e \models \phi$ for some e, in which case we also say that the environment e is a MODEL for ϕ. When Φ is a set of formulas (for which we will use the term QUERY), we write $e \models \Phi$ to mean that $e \models \phi$ holds for all $\phi \in \Phi$.

We will need to discuss satisfiability in models with specified cardinality, so let the "equality" $\sigma \doteq n$ denote a CARDINALITY CONSTRAINT: by $\langle \iota, \rho \rangle \models \sigma \doteq n$ we mean that the set $[\![\sigma]\!]\, \iota$ has n elements.

Similarly, we will consider TYPE CONSTRAINTS of the form $\alpha \doteq \sigma$ and VARIABLE CONSTRAINTS of the form $u^\sigma \doteq v^\tau$. By definition, $\langle \iota, \rho \rangle \models \alpha \doteq \sigma$ holds iff $\iota(\alpha) = [\![\sigma]\!]\, \iota$, and $\langle \iota, \rho \rangle \models u^\sigma \doteq v^\tau$ holds iff $[\![\sigma]\!]\, \iota = [\![\tau]\!]\, \iota$ and $\rho(u^\sigma) = \rho(v^\tau)$.

Example 6. Consider the $\mathcal{S}_{\mathsf{Eq}}$-queries $\{f(f(f\,x)) = x, f(f\,x) = x, f\,x \neq x\}$ and $\{f\,x = g\,x, g\,x = h\,x, f \neq g, g \neq h, h \neq f\}$, where f, g, h are variables of type $\alpha \Rightarrow \alpha$ and x is one of type α. The first query is unsatisfiable. The second query is satisfiable, but is not simultaneously satisfiable with the cardinality constraint $\alpha \doteq 2$. (E.g., there are only two functions $\mathbb{B} \to \mathbb{B}$ that map *true* to *false*.)

A Σ-THEORY is a set of Σ-structures. If $\mathcal{T}$ is a Σ-theory, we say that a formula ϕ is $\mathcal{T}$-SATISFIABLE if it is $\mathcal{S}$-satisfiable for some $\mathcal{S} \in \mathcal{T}$.

The theories $\mathcal{T}_{\mathsf{Int}}$, $\mathcal{T}_{\mathsf{List}}$, $\mathcal{T}_{\mathsf{Array}}$ (Examples 1 and 2) are each the theory of a single structure: $\mathcal{S}_{\mathsf{Int}}$, $\mathcal{S}_{\mathsf{List}}$, $\mathcal{S}_{\mathsf{Array}}$ respectively. On the other hand, $\mathcal{T}_{\mathsf{Monoid}}$ is the theory of all monoids. From now on, we assume that *every theory is defined by a single algebraic structure*, since such theories are of greatest practical interest.

By a SOLVER we will mean a sound and complete satisfiability procedure for Σ-queries whose formulas belong to a specified subset (FRAGMENT) of Term_Σ. For example, integer linear arithmetic is the Σ_{Int}-fragment consisting of boolean combinations of linear equalities and inequalites, and the integer linear programming algorithms can be seen as solvers for this fragment. Solvers that can check satisfiability with cardinality constraints will be called STRONG.

We will concern ourselves only with subfragments of the APPLICATIVE FRAGMENT of theories, where a Σ-term is called applicative if it contains no subterms that are abstractions and all occurrences of constants are fully applied. The latter means that every occurrence of a constant k^τ is part of a subterm of the form $k^\tau\, M_1 \cdots M_m$, where m is the constant's arity. Define also the ALGEBRAIC FRAGMENT to consist of all applicative terms that do not contain any occurrences of subterms of the form $x\, N$, where x is a variable ("uninterpreted function").

In the rest of this section we will narrow down the applicative fragment to a subfragment whose queries have a particularly simple form. First, we minimize the size of the formulas occurring in the query at the price of increasing

the number of formulas in the query. Second, we do away with the propositional complexity of the query by case splitting over boolean variables. Finally, with a substitution, we remove equalities between variables from the query. This reduction will further ease our reasoning, and will incur no cost in generality.

Lemma 2. *Every applicative query over $\langle O \,|\, K \rangle$ is equisatisfiable with a query all of whose formulas are* ATOMIC*, i.e. have one of the following forms:*

(A) $x_0 = k\, x_1 \ldots x_n$*, where* $k \in K^+$ *has arity* n
(B) $x_0 = x_1\, x_2$

where the x_i *are variables. Also, an algebraic query is equisatisfiable with a query whose formulas all have the form (A).*

Transforming an applicative formula into a set of atomic formulas is done simply by introducing proxy variables for subterms, a process often called *variable abstraction*. For example, $(f\, x\, 1 \geq 1) \vee (x = 1)$ is equisatisfiable with: *(A)* $y = 1, p = (z \geq y), q = (x = y), r = p \vee q, r = \mathsf{true}$; *(B)* $g = f\, x, z = g\, y$.

An ARRANGEMENT is a query determined by a set V of variables of the same type and an equivalence relation $\sim$ on V. For every $x, y \in V$, the arrangement contains either $x = y$ or $x \neq y$, depending on whether $x \sim y$ holds or not. The arrangement that forces all variables in V to be distinct will be denoted $\mathsf{Dist}(V)$.

Suppose now Φ is a set of atomic formulas and let X^σ be the set of variables of type σ that occur in Φ. Let E^σ be the subset of Φ consisting of formulas of the form $z = (x = y)$, where $x, y \in X^\sigma$. We can assume that E^{Bool} is empty by using the alternative way $z = (x \Leftrightarrow y)$ of writing $z = (x = y)$. We can also assume that for every $\sigma \neq \mathsf{Bool}$ and every $x, y \in X^\sigma$ there exists z such that $z = (x = y)$ occurs in E^σ; just add this equality with a fresh z if necessary.

There are finitely many substitutions $\xi \colon X^{\mathsf{Bool}} \to \{\mathsf{true}, \mathsf{false}\}$ and Φ is satisfiable iff some $\xi(\Phi)$ is. Let Φ_0 be the subset of Φ consisting of formulas *(A)* in which k is a boolean connective. Note that for any ξ, the query $\xi(E^\sigma)$ is either unsatisfiable, or equivalent to an arrangement on X^σ. Searching for a model for Φ, we can enumerate all ξ such that $\xi(\Phi_0)$ is satisfiable, and every $\xi(E^\sigma)$ is an arrangement. Thus, we will have a solver for all applicative $\mathcal{T}$-queries as soon as we have a solver for ALMOST-REDUCED queries that consist of

- arrangements Δ^σ for every type $\sigma \neq \mathsf{Bool}$ that occurs in the query
- the set Δ^{Bool} containing $x = \mathsf{true}$ or $x = \mathsf{false}$ for every $x \in X^{\mathsf{Bool}}$
- non-logical atomic formulas (where constants k in *(A)* are not logical)

Observe finally that for every almost-reduced query there is an equisatisfiable REDUCED query in which (1) $\Delta^\sigma = \mathsf{Dist}(X^\sigma)$ for every $\sigma \neq \mathsf{Bool}$ and (2) there are only two variables of type Bool—say $\mathbb{t}$ and $\mathbb{f}$—and two equations in Δ^{Bool}, namely $\mathbb{t} = \mathsf{true}$ and $\mathbb{f} = \mathsf{false}$. Indeed, we can bring a given almost-reduced query to this simpler form by choosing a representative for each class of the arrangements Δ^σ and then replacing every occurrence of $x \in X^\sigma$ with its representative.

Example 7. Let $\mathcal{T} = \mathcal{T}_1 + \mathcal{T}_2$, where $\mathcal{T}_1 = \mathcal{T}_{\mathsf{Int}}$ and $\mathcal{T}_2 = \mathcal{T}_\times$ is the simple parametric theory of pairs over the signature

$$\Sigma_\times = \langle \times \mid \langle \text{-}, \text{-} \rangle^{[\alpha,\beta]\Rightarrow\alpha\times\beta}, \mathsf{fst}^{\alpha\times\beta\Rightarrow\alpha}, \mathsf{snd}^{\alpha\times\beta\Rightarrow\beta} \rangle.$$

Consider the query $\Phi = \{x_2 = \langle \mathsf{snd}(\mathsf{snd}\, x_3), x_1\, x_2\rangle,\ \mathsf{fst}(\mathsf{snd}\, x_3) > 0\}$ whose variables are typed as follows: $x_1 : \omega \times \mathsf{Bool} \Rightarrow \mathsf{Bool}; x_2 : \omega \times \mathsf{Bool}; x_3 : \omega \times (\mathsf{Int} \times \omega)$, where ω is a type variable. The types of instances of fst and snd can be inferred, so we leave them implicit. Variable abstraction produces $\Phi' = \{x_4 = x_1\, x_2,\ x_5 = \mathsf{snd}\, x_3,\ x_6 = \mathsf{snd}\, x_5,\ x_2 = \langle x_6, x_4\rangle,\ x_7 = \mathsf{fst}\, x_5,\ x_8 = 0,\ x_9 = (x_7 > x_8),\ x_9 = \mathsf{true}\}$. Proxy variables have the following types: $x_4, x_9 : \mathsf{Bool}$; $x_7, x_8 : \mathsf{Int}$; $x_5 : \mathsf{Int} \times \omega$; $x_6 : \omega$. The assignment $\xi = [\mathsf{false}/x_4, \mathsf{true}/x_9]$ to propositional variables and the arrangement $\mathsf{Dist}\{x_7, x_8\}$ produce the reduced query $\Phi'' = \Delta^{\mathsf{Bool}} \cup \mathsf{Dist}\{x_7, x_8\} \cup \Phi_0 \cup \Phi_1 \cup \Phi_2$, where $\Phi_0 = \{\mathbb{f} = x_1\, x_2\}$, $\Phi_1 = \{x_8 = 0,\ \mathbb{t} = (x_7 > x_8)\}$, $\Phi_2 = \{x_5 = \mathsf{snd}\, x_3,\ x_6 = \mathsf{snd}\, x_5,\ x_2 = \langle x_6, \mathbb{f}\rangle,\ x_7 = \mathsf{fst}\, x_5\}$.

5 Nelson-Oppen Cooperation

The signatures $\Sigma_1 = \langle O_1 \mid K_1\rangle, \ldots, \Sigma_n = \langle O_N \mid K_N\rangle$ are DISJOINT if each *properly* contains Σ_{Eq} and the only constants and type operators that any two have in common are those of Σ_{Eq}. Their SUM SIGNATURE is $\Sigma = \Sigma_1 + \cdots + \Sigma_N = \langle O_1 \cup \cdots \cup O_N \mid K_1 \cup \cdots \cup K_N\rangle$. If each $\mathcal{T}_i$ is a Σ_i-theory determined by the structure $\mathcal{S}_i$, the SUM THEORY $\mathcal{T}$ is defined by the SUM Σ-STRUCTURE $\mathcal{S} = \mathcal{S}_1 + \cdots + \mathcal{S}_N$ that interprets every $F \in O_i$ and every $k^\sigma \in K_i$ the same way the structure $\mathcal{S}_i$ does it.

Our main result is the construction of a strong solver for the applicative fragment of $\mathcal{T}$, assuming the existence of strong solvers for the *applicative* fragment of $\mathcal{T}_{\mathsf{Eq}}$ and the *algebraic* fragment of every $\mathcal{T}_i$. The construction follows the original Nelson-Oppen approach [11], as revised by Tinelli and Harandi [20]. The completeness proof, however, is radically different and relies essentially on the parametricity of the component structures $\mathcal{S}_i$.

5.1 The Combined Solver

Let Σ and $\mathcal{T}$ be a sum signature and sum theory as above; for convenience, from now on, Σ_0 will stand for Σ_{Eq}. Given an input applicative Σ-query Φ_{in} and a set of cardinality constraints Γ, the combined solver proceeds as follows.

1. Create, as described in Section 4, a set $\mathcal{F}$ of reduced queries such that Φ_{in}, Γ is $\mathcal{T}$-satisfiable iff Φ, Γ is $\mathcal{T}$-satisfiable for some $\Phi \in \mathcal{F}$.[3]
2. Processing a $\Phi \in \mathcal{F}$, partition it into subqueries $\Delta^{\mathsf{Bool}} = \{\mathbb{t} = \mathsf{true}, \mathbb{f} = \mathsf{false}\}$, $\mathsf{Dist}(X^\sigma)$ for all $\sigma \neq \mathsf{Bool}$, and $\Phi_0, \Phi_1, \ldots, \Phi_N$, where Φ_0 is a set of atomic formulas of the form *(B)*, and Φ_i is a set of non-logical atomic formulas of the form *(A)* with the constant k taken from K_i^+. (See Example 7.)

[3] The terrible inefficiency of enumerating propositional assignments and arrangements can be alleviated with techniques involving the use of a SAT solver, but is not our concern here. See, e.g., [13].

3. *Purify* each Φ_i into a reduced Σ_i-query Ψ_i, algebraic for $i > 0$, and a set of constraints Γ_i that are all together $\mathcal{T}$-equisatisfiable with Φ, Γ. (See Example 8 below.)
4. Use strong solvers for $\mathcal{T}_i$ to check the joint $\mathcal{T}_i$-satisfiability of Ψ_i and the cardinality constraints in Γ_i. Return "Φ, Γ satisfiable" iff all solvers return "satisfiable".

Purification in 3. is a four-step procedure:

1. Proxying types. Let T be the set of types containing the types of all subterms of formulas in Φ, and all types that occur as subexpressions of these. Partition T into the set of type variables T^{var}, the set T_0 of function types, and the sets T_i $(i = 1, , , ., N)$ of types of the form $F(\sigma_1, \ldots, \sigma_n)$ where $F \in O_i - \{\Rightarrow\}$. For every $\sigma \in T_i$, let α_σ be a fresh (proxy) type variable, and let σ° be the type obtained from σ by replacing each maximal alien (i.e., element of T_j for $j \neq i$) type τ that occurs as a subexpression in σ with the proxy α_τ.

2. Proxying variables. Partition the set X of variables occurring in Φ into $\{\mathbb{t}, \mathbb{f}\}$, X^{var}, $X_0, \ldots, X_N$, where $x \in X^{\mathrm{var}}$ iff the type of x is in T^{var}, and $x \in X_i$ iff the type of x is in T_i. For convenience, let us assume that the elements of X are $x_1, x_2, \ldots$. Introduce sets of fresh variables $Y_i = \{y_j \mid x_j \in X_i\}$ and $Z_i = \{z_j \mid x_j \in X_i\}$. By definition, the type of each y_j is σ°, and the type of z_j is α_σ, where σ is the type of x_j. Let $Y^\sigma = \{y_j \mid x_j \in X^\sigma\}$ and $Z^\sigma = \{z_j \mid x_j \in X^\sigma\}$. Let Y be the union of all the Y_i and Z be the union of the Z_i. Finally, let $\Delta_i = \Delta^{\mathsf{Bool}} + \bigcup_{\sigma \notin T_i} \mathsf{Dist}(Y^\sigma) + \bigcup_{\sigma \in T_i} \mathsf{Dist}(Z^\sigma)$—a union of arrangements.

3. Generating constraints. Let Γ_i^{card} be the union of Γ and cardinality constraints $\alpha_\sigma \doteq n$, where $\sigma \in T_j$, $j \neq i$, and $\sigma \doteq n$ is implied by Γ. Let also Γ_i^{type} be the set of type constraints $\alpha_\sigma \doteq \sigma^\circ$, where σ is an i-type. Note that these type constraints imply $\alpha_\sigma \doteq \sigma$ for every non-variable type σ. Let Γ_i^{var} be the set of variable constraints $z_j \doteq y_j$, where $x_j \in X_i$. Finally, let Γ_i be the union of Γ_i^{card}, Γ_i^{type}, and Γ_i^{var}.

4. Purifying atomic formulas. For every $x \in X$ and $i = 0, \ldots, N$ define

$$x_j^{[i]} = \begin{cases} x_j & \text{if } x_j \in \{\mathbb{t}, \mathbb{f}\} \cup X^{\mathrm{var}} \\ y_j & \text{if } x_j \in X_i \\ z_j & \text{if } x_j \in X_{i'} \text{ and } i' \neq i \end{cases}$$

and then (with k' and k in (3) being appropriately typed instances of the same constant in K_i)

$$\Psi_0 = \Delta_0 \cup \{u_0^{[0]} = u_1^{[0]}\, u_2^{[0]} \mid (u_0 = u_1\, u_2) \in \Phi_0\} \tag{2}$$

$$\Psi_i = \Delta_i \cup \{u_0^{[i]} = k'\, u_1^{[i]}\, \ldots\, u_n^{[i]} \mid (u_0 = k\, u_1\, \ldots\, u_n) \in \Phi_i\} \quad (i > 0) \tag{3}$$

Lemma 3 (Purification). *Every Ψ_i is a well-defined Σ_i-query and Γ_i is a set of Σ_i-constraints. The union of all the Ψ_i and Γ_i is $\mathcal{T}$-equisatisfiable with Φ, Γ.*

Example 8. Continuing with Example 7, purification of $\Phi_0 \cup \Phi_1 \cup \Phi_2$ produces:

$$\Psi_0 = \Delta^{\mathsf{Bool}} \cup \{\mathbb{f} = y_1\, z_2\} \qquad \Gamma_0 = \{\alpha_{\omega\times\mathsf{Bool}\Rightarrow\mathsf{Bool}} \doteq \alpha_{\omega\times\mathsf{Bool}} \Rightarrow \mathsf{Bool},\ z_1 \doteq y_1\}$$
$$\Psi_1 = \Delta^{\mathsf{Bool}} \cup \{y_7 \neq y_8;\ y_8 = 0,\ \mathbb{t} = (y_7 > y_8)\}\ \Gamma_1 = \{\alpha_{\mathsf{Int}} \doteq \mathsf{Int},\ z_7 \doteq y_7,\ z_8 \doteq y_8\}$$
$$\Psi_2 = \Delta^{\mathsf{Bool}} \cup \{y_5 = \mathsf{snd}\, y_3,\ x_6 = \mathsf{snd}\, y_5,\ z_7 = \mathsf{fst}\, y_5,\ y_2 = \langle x_6, \mathbb{f}\rangle\}$$
$$\Gamma_2 = \{\alpha_{\omega\times\mathsf{Bool}} \doteq \omega \times \mathsf{Bool},\ z_2 \doteq y_2;\ \alpha_{\omega\times(\mathsf{Int}\times\omega)} \doteq \omega \times (\alpha_{\mathsf{Int}} \times \omega),\ z_3 \doteq y_3;$$
$$\alpha_{\mathsf{Int}\times\omega} \doteq \alpha_{\mathsf{Int}} \times \omega,\ z_5 \doteq y_5\}$$

where each type constraint $\alpha_\sigma \doteq \sigma^\circ$ in Γ_i is followed by variable constraints $z_j \doteq y_j$ with $z_j \colon \alpha_\sigma$ and $y_j \colon \sigma^\circ$.

5.2 The Combination Theorem

Lemma 3 implies that the combined solver is sound: the input $\Phi_{\mathrm{in}}, \Gamma$ is unsatisfiable if the solver says so. Completeness is less clear because it requires that a $\mathcal{T}$-model be assembled from a collection of $\mathcal{T}_i$-models. When the theories satisfy a flexibility condition *à la* Löwenheim-Skolem, completeness follows immediately from the following theorem.

Theorem 1. *Assume the notation is as in the previous section and that the theories $\mathcal{T}_1, \ldots, \mathcal{T}_n$ are flexible for reduced algebraic queries. Then: Φ, Γ is $\mathcal{T}$-satisfiable if and only if $\Psi_i, \Gamma_i^{\mathrm{card}}$ is $\mathcal{T}_i$-satisfiable for every $i = 0, \ldots, N$.*

Here are the requisite definitions. An environment $\langle \iota, \rho\rangle$ is SEPARATING if ρ maps all variables of the same type to distinct elements. A theory is FLEXIBLE for a fragment $\mathcal{F}$ if for every separating model $\langle \iota, \rho\rangle$ for an $\mathcal{F}$-query Ψ and every $\alpha \in \mathtt{dom}(\iota)$, there exist separating models $\langle \iota^{\mathrm{up}(\kappa)}, \rho^{\mathrm{up}(\kappa)}\rangle$ and $\langle \iota^{\mathrm{down}}, \rho^{\mathrm{down}}\rangle$ for Ψ such that $\iota^{\mathrm{up}(\kappa)}(\beta) = \iota(\beta) = \iota^{\mathrm{down}}(\beta)$ for every $\beta \neq \alpha$, and

1. [UP-FLEXIBILITY] $\iota^{\mathrm{up}(\kappa)}(\alpha)$ has any prescribed cardinality κ greater than the cardinality of $\iota(\alpha)$
2. [DOWN-FLEXIBILITY] $\iota^{\mathrm{down}}(\alpha)$ is countable

Lemma 4. *Every fully parametric structure is up-flexible for reduced algebraic queries. It is also down-flexible for this fragment if it satisfies the following condition: for every type operator F and every element $a \in [F](A_1, \ldots, A_n)$, there exist countable subsets A_i' of A_i such that $a \in [F](A_1', \ldots, A_n')$.*

We have proved that $\mathcal{T}_{\mathsf{Eq}}$ is flexible for reduced queries [10]. Also, by Lemma 4, the theories of common datatypes mentioned in Lemma 1 all qualify for complete Nelson-Oppen cooperation. The mild condition in Lemma 4 required for down-flexibility is probably unnecessary. We conjecture (but are unable to prove without informal reference to the downward Löwenheim-Skolem Theorem) that down-flexibility for algebraic queries holds for all fully parametric theories.

The lemma below follows from parametricity theorems [10] and is central for the proof of Theorem 1. We use it to incrementally modify the members of a given family of $\mathcal{T}_i$-models so that at each step they agree more on the intersections of their domains; at the end, a $\mathcal{T}$-model is obtained by amalgamating the modified $\mathcal{T}_i$-models.

Lemma 5 (Remodeling). *Suppose $\langle \iota, \rho \rangle$ is a separating model for an algebraic query Ψ in a fully parametric structure, and $f\colon \iota(\alpha) \to \iota(\alpha)$ is a bijection for some $\alpha \in \mathtt{dom}(\iota)$. Then there exists a separating model $\langle \iota, \rho' \rangle$ for Ψ such that*

(a) $\rho'(x) = f(\rho(x))$ for every variable $x \in \mathtt{dom}(\rho)$ of type α
(b) $\rho'(y) = \rho(y)$ for every $y \in \mathtt{dom}(\rho)$ whose type does not depend on α

Example 9. To illustrate the proof of Theorem 1, let us continue with Example 8. Starting with $\mathcal{T}_i$-models $\langle \iota_i, \rho_i \rangle$ for Ψ_i $(i = 0, 1, 2)$, we build a model $\langle \iota, \rho \rangle$ for the union of the Ψ_i and Γ_i. Let us order the types in T with respect to their complexity as in the first row of the table below. Let ι be a type environment that maps the original type variable ω and the proxy type variables α_σ for $\sigma \in T$ to sets in the second row of the table. Here $\mathbb{I} = \{\star, \dagger, \ddagger, \ldots\}$ is an arbitrary infinite set. Using the up- or down-flexibility of $\mathcal{T}_i$ and a simple consequence of parametricity ("permutational invariance"), we first modify the given models to achieve $\iota_0 = \iota_1 = \iota_2 = \iota$; this will satisfy all type constraints too. Then we modify the environments ρ_i in six steps, corresponding to the six types in T, so that after the step related to $\sigma \in T$, the ρ_i agree on their variables associated with σ and all types preceding σ. (For each $x_m \in X^\sigma$, one of the ρ_i has y_m in its domain, while the others have z_m.) These changes are possible by Lemma 5. The top half of the table shows the ρ_i's after the second step, where we have agreement on variables associated with ω and $\omega \times \mathsf{Bool}$ (the shaded area). Turning to the type Int, the pivot values $4, 0$ (underlined) are picked from the "owner" model ρ_1, and ρ_0, ρ_2 adjust to it, with appropriate changes at "higher" types. The table also shows the pivot value $\langle 4, \dagger \rangle$ for the next step .

σ	ω	$\omega \times \mathsf{Bool}$	Int		$\mathsf{Int} \times \omega$	$\omega \times (\mathsf{Int} \times \omega)$	$\omega \times \mathsf{Bool} \Rightarrow \mathsf{Bool}$
$[\![\sigma]\!]\,\iota$	$\mathbb{I}$	$\mathbb{I} \times \mathbb{B}$	$\mathbb{Z}$		$\mathbb{Z} \times \mathbb{I}$	$\mathbb{I} \times (\mathbb{Z} \times \mathbb{I})$	$\mathbb{I} \times \mathbb{B} \Rightarrow \mathbb{B}$
	x_6	y_2 or z_2	y_7 or z_7	y_8 or z_8	y_5 or z_5	y_3 or z_3	y_1 or z_1
ρ_0	$\dagger$	$\langle \dagger, \mathsf{false} \rangle$	1	5	$\langle 10, \ddagger \rangle$	$\langle \dagger, \langle 11, \star \rangle \rangle$	$\boldsymbol{\lambda} u.\,\mathsf{false}$
ρ_1	$\dagger$	$\langle \dagger, \mathsf{false} \rangle$	$\underline{4}$	$\underline{0}$	$\langle 12, \star \rangle$	$\langle \star, \langle 13, \dagger \rangle \rangle$	$\boldsymbol{\lambda} u.\,\mathsf{true}$
ρ_2	$\dagger$	$\langle \dagger, \mathsf{false} \rangle$	3	7	$\langle 3, \dagger \rangle$	$\langle \ddagger, \langle 3, \dagger \rangle \rangle$	$\boldsymbol{\lambda} u.\,\mathsf{true}$
ρ'_0	$\dagger$	$\langle \dagger, \mathsf{false} \rangle$	4	0	$\langle 10, \ddagger \rangle$	$\langle \dagger, \langle 11, \star \rangle \rangle$	$\boldsymbol{\lambda} u.\,\mathsf{false}$
ρ'_1	$\dagger$	$\langle \dagger, \mathsf{false} \rangle$	4	0	$\langle 12, \star \rangle$	$\langle \star, \langle 13, \dagger \rangle \rangle$	$\boldsymbol{\lambda} u.\,\mathsf{true}$
ρ'_2	$\dagger$	$\langle \dagger, \mathsf{false} \rangle$	4	0	$\underline{\langle 4, \dagger \rangle}$	$\langle \ddagger, \langle 4, \dagger \rangle \rangle$	$\boldsymbol{\lambda} u.\,\mathsf{true}$

6 Conclusion and Future Work

We contend that the base logic for SMT should have parametric types and polymorphic functions. These features make it possible to easily model typical datatypes by single parametric structures and to model (unbounded) combinations of several datatypes by simple parameter instantiation. of several datatypes by simple parameter instantiation.

Our revision of the Nelson-Oppen method relies just on the parametricity of the datatypes modeled by the component theories and on the existence of *strong solvers* for them. Parametricity requirements hold for virtually all datatypes of interest, so to make our method widely applicable it remains to enhance the existing satisfiability procedures into efficient strong solvers. This can likely be done in ways similar to [15], and is the subject of future work.

Acknowledgments. Thanks to John O'Leary for discussions on *HOL* semantics, and to Levent Erkök, John Harrison, John Matthews, Albert Oliveras, and Mark Tuttle for reading parts of the manuscript and commenting on it.

References

1. N. Ayache and J.-C. Filliâtre. Combining the Coq proof assistant with first-order decision procedures. (unpublished), 2006.
2. C. Barrett and S. Berezin. CVC Lite: A new implementation of the cooperating validity checker. In *Computer Aided Verification (CAV)*, vol. 3114 of *LNCS* , pp. 515–518. 2004.
3. C. Barrett, I. Shikanian, and C. Tinelli. An abstract decision procedure for satisfiability in the theory of recursive data types. In *Pragmatics of Decision Procedures in Automated Deduction (PDPAR)*, 2006.
4. P. Fontaine and E. P. Gribomont. Combining non-stably infinite, non-first order theories. In *Pragmatics of Decision Procedures in Automated Deduction*, 2004.
5. P. Fontaine et al. Expressiveness + automation + soundness: Towards combining SMT solvers and interactive proof assistants. In *Tools and Algorithms for the Construction and Analysis of Systems)*, vol. 3920 of *LNCS*, pp. 167–181. 2006.
6. S. Ghilardi, E. Nicolini, and D. Zucchelli. A comprehensive combination framework. *ACM Transactions on Computational Logic*, 2007. (to appear).
7. M. J. C. Gordon and T. F. Melham, editors. *Introduction to HOL: A Theorem Proving Environment for Higher Order Logic*. Cambridge University Press, 1993.
8. J. Grundy et al. Tool building requirements for an API to first-order solvers. *ENTCS*, 144(2):15–26, 2006.
9. J. Harrison. Towards self-verification in HOL Light. In *Automated Reasoning (IJCAR)*, vol. 4130 of *LNAI*. 2006.
10. S. Krstić et al. Combined satisfiability modulo parametric theories. Tech. report, Oct. 2006. (`ftp://ftp.cs.uiowa.edu/pub/tinelli/papers/KrsGGT-RR-06.pdf`).
11. G. Nelson and D. C. Oppen. Simplification by cooperating decision procedures. *ACM Transactions on Programming Languages and Systems*, 1(2):245–257, 1979.
12. R. Nieuwenhuis and A. Oliveras. Congruence closure with integer offsets. In *Logic for Programming, AI and Reasoning (LPAR)*, vol. 2850 of *LNCS*, pp. 78–90. 2003.
13. R. Nieuwenhuis, A. Oliveras, and C. Tinelli. Solving SAT and SAT Modulo Theories: From an abstract Davis-Putnam-Logemann-Loveland procedure to DPLL(T). *Journal of the ACM*, 2006. (to appear).
14. T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*, vol. 2283 of *LNCS*. 2002.
15. S. Ranise, C. Ringeissen, and C. G. Zarba. Combining data structures with nonstably infinite theories using many-sorted logic. In *Frontiers of Combining Systems (FroCoS)*, vol. 3717 of *LNCS*, pp. 48–64. 2005.

16. S. Ranise and C. Tinelli. The SMT-LIB standard: Version 1.2. Technical report.
17. J. C. Reynolds. Types, abstraction and parametric polymorphism. In *Information Processing: 9th World Computer Congress*, pp. 513–523. North-Holland, 1983.
18. C. Ringeissen. Cooperation of decision procedures for the satisfiability problem. In *Frontiers of Combining Systems (FroCoS)*, vol. 3 of *Applied Logic*, pp. 121–140.
19. N. Shankar. Using decision procedures with a higher-order logic. In *Theorem Proving in Higher Order Logics (TPHOLS)*, vol. 2152 of *LNCS*, pp. 5–26, 2001.
20. C. Tinelli and M. Harandi. A new correctness proof of the Nelson-Oppen combination procedure. In *Frontiers of Combining Systems (FroCoS)*, vol. 3 of *Applied Logic*, pp. 103–120. Kluwer, 1996.
21. C. Tinelli and C. Zarba. Combining decision procedures for sorted theories. In *Logic in Artificial Intelligence (JELIA)*, vol. 3229 of *LNAI*, pp. 641–653. 2004.
22. C. Tinelli and C. Zarba. Combining nonstably infinite theories. *Journal of Automated Reasoning*, 34(3):209–238, 2005.
23. P. Wadler. Theorems for free! In *Functional Programming Languages and Computer Architecture (FPCA)*, pp. 347–359. ACM Press, 1989.
24. C. G. Zarba. Combining sets with elements. In *Verification: Theory and Practice*, vol. 2772 of *LNCS*, pp. 762–782. 2004.