Universidad de Buenos Aires
Facultad de Ciencias Exactas y Naturales
Departamento de Computación

A study of the Combination Problem: dealing with multiple theories in SMT solving

Tesis presentada para optar al título de
Licenciada en Ciencias de la Computación

Paula Chocrón

Director: Carlos Gustavo López Pombo

Buenos Aires, 2014

Un estudio del problema de Combinación: soluciones a SMT módulo varias teorías

Hoy en día, las aplicaciones de software son ubicuas en nuestras vidas, y sus fallas pueden, en muchos casos, producir enormes pérdidas. Esto hace esencial el desarrollo de métodos de análisis de sofware. Para esto, usualmente se asume que se cuenta con una especificación formal del sistema a analizar; entonces es posible verificar si cierta propiedad de interés se deduce de la especificación. La lógica es frecuentemente utilizada como sistema formal para la especificación y verificación, lo cual vuelve un problema crucial el decidir si una propiedad es satisfacible módulo determinadas teorías.

Un problema de satisfaccibilidad se expresa frecuentemente en una combinación de varias teorías, y un enfoque natural consiste en resolverlo combinando los procedimientos de decisión con los que se cuenta para cada una de las teorías. En esta tesis estudiamos el problema de combinación de procedimientos para teorías de primer orden sobre signaturas que comparten símbolos. Presentamos dos enfoques diferentes para resolver este problema. En el primero, extendemos el método de combinación clásico para teorías sobre signaturas disjuntas de Nelson y Oppen al caso en el cual sólo se comparten predicados unarios. Teorías relevantes se pueden analizar desde este marco, como por ejemplo la clase de teorías de Löwenheim. El segundo enfoque estudia un caso particular, en el cual el fragmento de signatura compartida resulta de definir funciones que conectan elementos de dos teorías sobre signaturas disjuntas. Un ejemplo clásico es el de las listas extendidas con una función de longitud que las relaciona con la aritmética. Presentamos un procedimiento para resolver este caso, y mostramos cómo se puede adaptarlo para considerar distintas clases de teorías y funciones.

Este trabajo fue parcialmente desarrollado durante una pasantía en LORIA-INRIA, bajo la supevisión de Pascal Fontaine y Christophe Ringeissen, a quienes agradecemos especialmente.

Palabras claves: Satisfactibilidad Módulo Teorías, Análisis de Software, Modularidad, Combinación de procedimientos de decisión.

A study of the Combination Problem: dealing with multiple theories in SMT solving

Nowadays software artifacts are ubiquitous in our lives, and failures may, in many cases, produce enormous losses, making essential the development of methods to perform analysis over software. In order to do this, usually it is assumed that a formal specification of the system is available. Then, it is possible to check whether a certain property of interest follows from the specification. Logics have often been used as formal systems suitable for the specification and verification of software artifacts, making the problem of deciding if a property is satisfiable modulo some background theory crucial in verification.

A satisfiability problem is very often expressed in a combination of theories, and a natural approach consists in solving the problem by combining the decision procedures available for the component theories. In this thesis, we study the problem of combining decision procedures for first-order theories over signatures sharing symbols. We present two different approaches. In the first one, we extend a classical combination method by Nelson and Oppen for theories over disjoint signatures to the case when only unary predicates are shared. Some well-know classes of theories fit in our framework, for example the Löwenheim class. The secod approach focuses on an specific case, in which the non-disjointness arises from defining functions connecting terms in two theories with disjoint signatures. A classical example is the one of lists endowed with a lenght function that relates them with arithmetic. We present a procedure to solve this case, and discuss how it should be adapted to different classes of theories and functions.

This work was partially developed during an internship at LORIA-INRIA, under the supervision of Pascal Fontaine and Christophe Ringeissen, to whom we thank specially.

Keywords: Satisfiability Modulo Theories, Sofware Analysis, Modularity, Combination of Decision procedures.

Contents

1 Introduction

Nowadays, software artefacts are ubiquitous in our lives being an essential part of home appliances, cars, mobile phones, and even in more critical activities like aeronautics and health sciences. In this context software failures may produce enormous losses, either economical or, in the worst case, in human lives, and the satisfaction of certain standards of quality ensuring that certain errors do not occur are a requirement when producing software for these critical activities. Software analysis is an area of software engineering concerned with the application of rigorous techniques in order to prove the absence of these errors in software pieces. To perform analysis over software, usually it is assumed that a formal description of the behaviour of the system, i.e. a specification, is available. Then, it is possible to check whether a certain property of interest follows from the specification.

Logics have often been used as formal systems suitable for the specification of software artefacts. Moreover, logical specifications, being formal, might contribute towards the application of sound verification techniques. Several formalisms have been developed to cope with these aspects and most of them are effective in describing some particular characteristics of software systems. Techniques applied to these formal characterisations of software artefacts are usually referred to as formal methods.

Formal methods are usually divided into two categories: heavyweight and lightweight. These names refer to the amount of mathematical expertise needed during the process of proving a given property. For many specification languages a lower degree of involvement equates to a higher degree of automatization and, consequently, to less certainty regarding the satisfaction of the property. Thus lightweight formal methods, like Alloy [26], cannot usually be entirely trusted when dealing with critical models. An alternative is the use of heavyweight formal methods, as for instance semi-automatic theorem provers [35, 14, 33]. Theorem provers also exhibit limitations, for instance, they require an expertise from the user that many times discourages their use.

As we mentioned before, both lightweight and heavyweight formal methods rely on the formal description of a system and the desired properties. The concepts of satisfiability and validity of a formula are the basic elements regarding reasoning in a logic. These ideas can already be found in Aristotle's *categorical propositions*,

and the relationships between these concepts is explained through the *square of opposition*. The interested reader is pointed to [19] for a detailed historical exposition of the satisfiability problem.

The problem of determining if a formula is satisfiable can be expressed in different ways and has different solutions depending on the logic. In the case of propositional logic, the satisfiability problem has been largely studied with several implications in mathematics and computer science. Named SAT it was proved to be decidable and the first known example of an NP-complete problem [13]. There exist several, and different, implementations of SAT-solvers [29, 24], software systems devoted to solve instances of this problem. In first-order predicate logic, the general satisfiability problem is undecidable and it was proved by Church and Turing in response to the *Entscheidungsproblem* as stated by David Hilbert in 1928 [12]. Consequently, there is no hope in finding an algorithm to determine whether a first-order sentence is satisfiable or not. However, there exist decidable fragments of first-order predicate logic. A typical example is C_2, the fragment of first-order predicate logic with two variables and the counting quantifiers $\exists_{\leq n}$ and $\exists_{\geq n}$ ("there exists at least n" and "there exists at most n", respectively). There exist several examples of decidable first-order theories; this means that in some cases there exists a **decision procedure** for testing satisfiability of a given formula depending on the axioms considered in the theory.

Luckily for most applications we are not interested in deciding general satisfiability of a first-order formula. More commonly, the problem reduces to find if a formula is satisfiable under certain assumptions. For example, we usually do not want to consider any possible interpretation of the symbols 0 and $<$ that makes the "$x < 0$" satisfiable; we only regard on an specific interpretation, for example linear arithmetic.

The problem of determining if there exists an interpretation that satisfies a given *quantifier-free formula* φ modulo some first order theories is called **Satisfiability Modulo Theories** [6], and it is the base of the verification tools known as SMT-solvers [7]. SMT-solvers are essentially constraint solvers used to determine the validity of a formula under a certain set of axioms written in a logical language. Thus, we can think of them as the tool support required to answer the question: does the property φ hold for the system specified by the set of axioms Γ (this is usually denoted as $\Gamma \models \varphi$).

The key aspect in the development of an SMT-solver is to consider a reasonable set of decidable theories. There are many interesting theories meeting this condition for which specialized reasoning methods provide a decision procedure. Examples are: (1) The theory of equality (2) The theory of linear arithmetic (3) The theory of linear arithmetic over reals (4) Recursively defined data structures. A deeper explanation of each of these theories is given in Chapter 2.

The fact that these methods are theory-specific means that they fail to decide

satisfiability under combinations of theories. This introduces what is called the *Combination problem*:

> Given n theories $T_1, \ldots T_n$, is it possible to combine the available decision procedures for them into a single decision procedure for the satisfiability of a formula in the union theory $T_1 \cup \cdots \cup T_n$?

This problem is not simple; in fact, the combination of two decidable theories is not necessarily decidable, (see [8]), so having procedures for some theories does not mean having one for the combination. The need for decision procedures for combinations of theories is especially critical in software verification applications. Since the foundational work of Parnas [36, 37], practitioners build software artefacts (and particularly software specifications) modularly, as it produces better quality software description by profiting re-usability of modules and better separation of concerns. A direct consequence of this is that one of the key problems to be solved in SMT-solving is how to modularly apply ad-hoc decision procedures to partial models in a way that conclusions drawn for them can be promoted to the theory resulting from their composition.

The problem of combining decision procedures was addressed for the first time more than 30 years ago by Nelson and Oppen in [30] and Shostak in [43]. In that work, Nelson and Oppen proposed a general method for combining decision procedures that is still used by actual SMT-solvers.

It was in [43] that Shostak proposed a more restricted method based on congruence closure for combining the quantifier-free theory of equality with certain kind of theories. His method is also used in several verification systems like ICS [17] and PVS [35].

These two methods, specially Nelson and Oppen's, are currently considered state-of-the-art in combining decision procedures. However, they both make strong assumptions on the theories being combined. In its initial presentation, the Nelson-Oppen combination method requires the theories in the combination to be: (1) signature-disjoint and (2) stably infinite (to guarantee the existence of an infinite model). These are strong limitations, and many recent advances aim to go beyond disjointness.

In this work we focus on combining theories over non-disjoint signatures. This problem is much harder and evidence leads to suppose it is not possible to obtain general decidability results for non-disjoint theories. However, it is possible to combine decision procedures for theories whose signatures do not need to be disjoint into a semi-decision procedure for the unsatisfiability of formulae modulo the union of the theories. With the rewrite-based approach initiated in [4], the problem reduces to proving the termination of this calculus in particular (combinations of) theories. A general criteria has been proposed to get modular termination results

for superposition, when T_1 and T_2 are either disjoint [3] or non-disjoint [41].
As another example of a general approach to the problem, in [23], Ghilardi proposed a very general model-theoretic combination framework to obtain a combination method à la Nelson-Oppen when T_1 and T_2 are two *compatible* extensions of the same shared theory T_0 (satisfying some properties). This framework relies on an application of the Robinson Joint Consistency Theorem (roughly speaking, the union of theories is consistent if the intersection is complete). In this framework, several requirements have to be satisfied but useful examples of shared theories T_0 have been successfully studied, e.g. some significant fragments of arithmetic [23, 31, 32].

Due to the difficulty of generalization, and specially of finding concrete cases of application that fulfil the requirements for general model-theoretic frameworks, the research in obtaining concrete decision procedures for unions of theories has been mostly focused in solving specific cases, for which the problem is easier. We here explore two different approaches.

Overview of the document

The logical framework used in this document is first-order logic with equality. In Section 4 we will need to introduce sorts, so we will use many-sorted first-order logic as well. Definitions of these two systems and notations are given in Chapter 2. In Section 2.3, we provide some examples of decidable first order theories that we will use frequently, and we briefly discuss the methods their decision procedures are based on. To conclude, in Section 2.4 we present the combination problem in Satisfiability Modulo Theories. We prove the Combination Theorem, which is essential for the rest of the sections, and the classical Nelson-Oppen's procedure for combining disjoint and stably infinite theories. In this chapter, we also introduce some of the structures we will use often.

Chapters 3 and 4 contain the contributions of the thesis. We present two approaches for solving particular cases of non-disjoint combination. In Chapter 3 we introduce a framework to solve combination that extends Nelson and Oppen's procedure in the case when the signatures of the theories to be combined share only a finite set of unary predicates. In this case, the cardinality constrains the theories must satisfy in order to be combinable are more complicated than for disjoint signatures. We extend the ideas for the disjoint case [18] to introduce a class of theories, called $\mathcal{P}$-gentle, that fulfil these requirements. This framework also extends beyond stably infiniteness. In Section 3.2, we show that a major class of theories, the Löwenheim class, is $\mathcal{P}$-gentle, as well as the Bernays-Schönfinkel-Ramsey (BSR) class. Most of this section are part of an article that is submitted to the conference IJCAR'14.

In Chapter 4, we present a different approach. There we focus on a special case in which the non-disjointness arises from defining *bridging functions* between two

structures that belong to disjoint theories, say T_1 and T_2. Our aim is to reduce the satisfiability problem in the theory that extends $T_1 \cup T_2$ with these functions to the disjoint combination problem in T_1 and T_2. This framework is specially useful when defining functions over data structures, for example the length over lists, a frequent need in software specifications. In Section 4.1, we propose a procedure that generalizes existent ones. In the following sections we show cases, analysing the instantiation of the method for different classes of theories. In Section 4.4, we discuss issues about applicability.

Finally, in Chapter 5, we draw some conclusions and discuss future work, both improvements of what was done and possible lines for further research.

2 Preliminaries

We introduce here the logical framework and notations used throughout this work. We assume familiarity with First-Order Logic, as we only fix notation and terminology here. The interested reader is pointed to [16]. As an extension, we introduce many-sorted First-Order Logic.

We later present some other definitions we will use frequently. Basic notions about Abstract Data Structures and their axiomatization in First-Order Logic is assumed. In the last section, we give a brief survey of methods for solving the disjoint Combination Problem, namely Nelson-Oppen's procedure. In this last section we also include some definitions that are useful to approach this kind of problems.

2.1 First-order Logic

We introduce basic concepts of first-order logic with equality (FOL$^{=}$ for short). We define the main syntactic and semantic concepts and fix the notation we will use.

Definition 1. *A first-order signature is a tuple* $\Sigma = \langle C, F, P \rangle$*, where:*

- *C is a countable set of constant symbols.*
- *F is a countably infinite set of function symbols. An arity r is associated to each function symbol.*
- *P is a countably infinite set of predicate symbols. An arity r is associated to each function symbol.*

Definition 2. *Given a signature* Σ*,* Σ-terms *are:*

- *the variables $x \in V$,*
- *the constants $c \in C$, and*
- *$f(t_1, \ldots t_r)$, for each function symbol $f \in F$, if $t_1, \ldots, t_r$ are terms.*

Definition 3. *Given a signature* Σ*, a* Σ-atom *is:*

- *an expression of the form* $p(t_1, \ldots, t_r)$*, where* $p \in P$ *and* $t_1, \ldots, t_r$ *are terms respectively or*
- $t = t'$*, for each pair* t, t' *of terms is an atomic formula.*

A flat atom is one that does not have chained applications of functions in the terms.

Σ*-literals are* Σ*-atoms or expressions of the form* $\neg A$*, where* A *is a* Σ*-atom. A set of* Σ*-literals is called a* Σ*-constraint.*

Definition 4. *Let* V *be an enumerable set of variables. The set of* Σ*-formulas over* V *is defined recursively as follows:*

- *every* Σ*-atom is a* Σ*-formula;*
- *for each* Σ*-formula* ψ*,* $\neg\psi$ *is a* Σ*-formula;*
- *for each* Σ*-formula* ψ *and each variable* $x \in V$*,* $(\forall x)\psi$*,* $(\exists x)\psi$ *are* Σ*-formulas;*
- *for each* Σ*-formulas* ψ_1 *and* ψ_2*,* $(\psi_1 \star \psi_2)$ *is a* Σ*-formula if* $\star \in \{\wedge, \vee, \rightarrow\}$*.*

Whenever no ambiguity arises, we will assume an arbitrary but fixed set V of variable symbols. Also, a pair of parentheses can be omitted when there is no ambiguity. When Σ is irrelevant or clear from the context, we will simply omit it, and refer to terms, atoms, literals, constraints and formulas.

Among all formulas, some forms are of special interest: a *clause* is a disjunction ($\vee$) of literals, and a *conjunctive normal form* is a conjunction ($\wedge$) of clauses. A *disjunctive normal form* is a disjunction ($\vee$) of conjunctions ($\wedge$). Many proof methods assume formulas are in some of these forms.

The set of variables of a formula (or term) ψ is the subset $Var(\psi)$ of V used in the formula (term). The set of free variables of a formula ψ is $Free(\psi)$ defined inductively on the structure of the formula:

- the free variables of an atom ψ are $Var(\psi)$
- the free variables of a Boolean combination of formulas is the union of the free variables of the formulas in the combination,
- the free variables for $\exists x.\psi$ and $\forall x.\psi$ contains all free variables in ψ but x.

A formula is *quantifier-free* if it does not contain the symbols $\exists$ and $\forall$. A formula (or term) ψ is ground if it does not contain any variable. Ground formulas are always quantifier-free. A formula ψ is closed if it does not contain free variables.

A formula is *universal* if it is of the form $\forall x_1 \ldots \forall x_n.\varphi$ where φ is quantifier-free. It is always possible to transform a given formula into an equisatisfiable universal formula, using Skolemization. We refer to [5] for Skolemization.

Two signatures are disjoint if they share no elements.

The following definitions concern the semantics of First-Order Logic.

Definition 5. *An interpretation $\mathcal{I}$ for a first-order signature $\Sigma = \langle C, F, P\rangle$ with domain D over a set of variables V provides:*

- *an element $\mathcal{I}[x] \in D$ for every variable $x \in V$*
- *an element $\mathcal{I}[c] \in D$ for every constant $c \in C$*
- *a function $\mathcal{I}[f] : D^r \to D$ for every function symbol $f \in F$ of arity r ,*
- *a predicate $\mathcal{I}[p] \subseteq D^r$ for every predicate symbol $p \in P$ of arity r,*

The interpretation $\mathcal{I}_{x_1/d_1,\ldots,x_n/d_n}$ for pairwise different variables $x_1, \ldots, x_n$ stands for the interpretation that agrees with $\mathcal{I}$, except that it interprets each variable x_i as d_i.

An interpretation $\mathcal{I}$ assigns a value $\mathcal{I}[t]$ in D to each term t. Similarly, $\mathcal{I}$ assigns a truth-value $\mathcal{I}[\psi]$ to each formula ψ. An interpretation $\mathcal{I}$ is a model of formula ψ (noted $\mathcal{I} \models \psi$) if $I[\psi]$ evaluates to true, and it is a model of the set of formulas T (noted $\mathcal{I} \models T$) if it is a model of every formula in T.

Definition 6. *A formula φ is*

- *valid, if it evaluates to true under all interpretations;*
- *satisfiable, if it evaluates to true under some interpretation;*
- *unsatisfiable, if it evaluates to false under all interpretations.*

A set $\{\varphi_1, \ldots, \varphi_n\}$ of formulae is valid, satisfiable, unsatisfiable if so is the conjunction $\varphi \wedge \cdots \wedge \varphi_n$. We say that two formulae φ and ψ are

- equivalent, if φ and ψ have the same truth-value under all interpretations;
- equisatisfiable, if φ is satisfiable if and only if so is ψ.

Let Ω be a signature and let $\mathcal{I}$ be an Ω-interpretation. For a subset Σ of Ω, we denote with $\mathcal{I}^\Sigma$ the Σ-interpretation obtained by restricting $\mathcal{I}$ to interpret only the symbols in Σ.

Given an interpretation $\mathcal{I}$ on domain D, the restriction $\mathcal{I}'$ of $\mathcal{I}$ on $D' \subseteq D$ is the unique interpretation on D' such that $\mathcal{I}$ and $\mathcal{I}'$ interpret predicates, functions and variables the same way on D'. An extension $\mathcal{I}'$ of $\mathcal{I}$ is an interpretation on a D' domain including D such that $\mathcal{I}'$ restricted to D is exactly $\mathcal{I}$.

Definition 7. *Let Σ be a signature, and let $\mathcal{A}$ and $\mathcal{B}$ be Σ-interpretations with domains $D_{\mathcal{A}}$ and $D_{\mathcal{B}}$. A map $h : D_{\mathcal{A}} \to D_{\mathcal{B}}$ is an isomorphism of $\mathcal{A}$ into $\mathcal{B}$ if the following conditions hold*

- *h is bijective;*
- *$h(\mathcal{A}[x]) = \mathcal{B}[x]$ for each variable $x \in V$;*
- *$h(\mathcal{A}[f](a_1, \ldots, a_n)) = \mathcal{B}[f](h(a_1), \ldots, h(a_n))$, for each n-ary function symbol $f \in F$ and $a_1, \ldots, a_n \in D_{\mathcal{A}}$;*
- *$(a_1, \ldots, a_n) \in \mathcal{A}[p]$ if and only if $(h(a_1), \ldots, h(a_n)) \in \mathcal{B}[p]$, for each n-ary predicate symbol $p \in P$ and $a_1, \ldots, a_n \in D_{\mathcal{A}}$.*

We write $\mathcal{A} \simeq \mathcal{B}$ to indicate that there exists an isomorphism of $\mathcal{A}$ into $\mathcal{B}$.

We finally introduce some definitions regarding theories.

Definition 8. *Let Σ be a signature. A Σ-theory is any set of closed Σ-formulae. Given a Σ-theory T, a T-interpretation is a Σ-interpretation $\mathcal{I}$ such that all Σ-formulae in T evaluate to true under $\mathcal{I}$.*

Definition 9. *Given a Σ-theory T, a Σ-formula φ is*

- *T-valid, if it evaluates to true under all T-interpretations;*
- *T-satisfiable, if it evaluates to true under some T-interpretation (this is the same as $T \cup \{\varphi\}$ being satisfiable);*
- *T-unsatisfiable, if it evaluates to false under all T-interpretations.*

T-validity, T-satisfiability, and T-unsatisfiability, T-equivalence and T-equisatisfiability are straightforward extensions of the previous definitions.

A *decision problem* is a question in some formal system with a yes-or-no answer. A problem of this kind is said to be *decidable* if there exists a procedure such that for any input, it yields an answer in a finite number of computational steps. (see Section 2.3 for examples).

Given a theory T, we can define several types of decision problems. In particular:

- the validity problem for T: decide, for each formula φ of the language, whether or not φ is T-valid;
- the satisfiability problem for T: decide, for each formula φ of the language, whether or not φ is T-satisfiable;

The quantifier-free validity problem and the quantifier-free satisfiability problem are equivalent, but restricting φ to be quantifier-free.

The cardinality of an interpretation is the cardinality of its domain D. In this work, we will consider cardinalities in order to analyse if there exists an isomorphism between two interpretations, and particularly between models of two theories. The following version of the classical Löwenheim-Skolem theorem will be useful for this. Details about this theorem and about cardinalities of interpretations can be found in [10]:

Theorem 1. *Let Σ be a signature and T a Σ-theory. If T has an infinite model, then T has models of all infinite cardinalities.*

This means that, if two theories have infinite models, they will for sure have models of equal cardinality. If a theory has only finite models, then we must find two with the same number of elements in order for an isomorphism to be possible.

2.1.1 Many-sorted first-order logic

In a computer science context, it is natural to think in term of different kind of variables, usually known as *types*. In order to support this, we need to use a variant of $\mathrm{FOL}^{=}$: many-sorted first-order logic. Contrary to standard first-order signatures, in many-sorted first order ones the arguments of function and predicate symbols may have different sorts, as well as variables. Many-sorted first-order logic can be translated easily to unsorted first-order logic, and many results for unsorted first-order logic can be transferred to many-sorted logic. We here give the basic ideas on how to introduce sorts. For further detail on this kind of logic, we refer to [16].

Definition 10. *A many-sorted first-order signature is a tuple $\Sigma = \langle S, C, F, P\rangle$, where:*

- *S is a countable non-empty set of sorts (or types);*
- *C is a countable set of constant symbols. A sort is associated to each constant.*
- *F is a countably infinite set of function symbols. An arity r and a sort $\tau_1, \ldots, \tau_r, \tau$, where $r \geq 1$ and $\tau_1 \ldots \tau_r, \tau \in S$, are associated to each function symbol.*
- *P is a countably infinite set of predicate symbols. An arity r and a sort $\tau_1, \ldots, \tau_n$, where $r \geq 1$ and $\tau_1 \ldots \tau_n \in S$, are associated to each predicate symbol.*

In many sorted logics, we are provided with an equality symbol and quantifiers for each sort $\tau \in S$, in addition to the classical operators $\wedge, \vee, \rightarrow, \neg$, and the parenthesis.

Terms are defined as in unsorted first-order logic, but considering the sorts. A variable has its own sort, while a term built by applying a function f of sort $\tau_1, \ldots, \tau_n, \tau$ to terms of sorts $\tau_1, \ldots, \tau_n$ has sort τ. To extend the definition of atoms for this kind of logic, we only need to take care of applying predicates to terms of the corresponding sorts. Formulae are defined in the same way, and all the other definitions are trivially extendible.

We also need to redefine the semantic part:

Definition 11. *An interpretation $\mathcal{I}$ for a many-sorted first-order signature Σ, over the set of variables $\bigcup_{\tau \in S} V_\tau$ provides:*

- *a non empty domain D_τ for each $\tau \in S$,*
- *a function $\mathcal{I}[f] : D_{\tau_1} \times \cdots \times D_{\tau_r} \to D_\tau$ for every function symbol f of arity r and sort $\tau_1, \ldots, \tau_r, \tau$*
- *a predicate $\mathcal{I}[p] \subseteq D_{\tau_1}, \times \cdots \times, D_{\tau_r}$ for every predicate symbol p of arity r and sort $\tau_1, \ldots, \tau_r$,*
- *an element $\mathcal{I}[x] \in D_\tau$ for every variable $x \in V_\tau$.*
- *an element $\mathcal{I}[c] \in D_\tau$ for every constant c of sort τ.*

We let $D = \bigcup_{\tau \in S} D_\tau$

All other definitions are similar to the ones for unsorted FOL.

2.2 Absolutely free data structures

Recursively defined data structures are ubiquitous in computer science, as they abstract the idea of a container. This kind of structures are built with functions called constructors, that adds elements of some sort to build a new structure from another one. In this section we formally define a class of theories of recursively defined data structures that satisfy that an element can be built in only one way from the constructors. To this aim, we add a set of axioms representing injectivity, acyclicity and disjointness of constructors. Similar definitions can be found in [45]. From now on, we will refer to Absolutely Free Data Structures as AFDS.

Let Σ_S be a many-sorted signature with at least two sorts, `struct` and `elem`, and the following elements:

- A certain number function symbols called $Cons_{\Sigma_S}$, or colloquially, *constructors*. This can be of two kinds:

  * constants of sort `struct`. We will refer to these constructors as *atomic constructors*.
  * n-ary functions symbols that take as input (at least) a number of elements of sort `struct` and a number of sort `elem` and return an element of sort `struct`.

- A certain number of function symbols, called *selectors*, which are axiomatized by pattern matching over the constructors, having in the right side of the definition only constructors and variables (they are not recursive).

- A certain number of function symbols, called *defined functions*, with arity $f : struct \to struct$, axiomatized by pattern matching over the constructors, and defined recursively.

The Σ_S-theory T_{struct} contains the axiomatization of all the *defined functions*, defining them over $Cons_{\Sigma_S}$. We only consider constructors that are injective, acyclic, and disjoint, by adding to T_{struct} the theory $AbsFree_{\Sigma_S}$, where

$$AbsFree_{\Sigma_S} = \bigcup_{c \in Cons_{\Sigma_S}} ((Inj_c) \cup (Acyc_c)) \cup \bigcup_{c,d \in Cons_{\Sigma_S}, c \neq d} Disjoint(c,d), \text{ with}$$

$$\begin{array}{ll} (Inj_c) & (\forall x_1,\ldots,x_n,y_1,\ldots,y_n)\left(c(x_1,\ldots,x_n) = c(y_1,\ldots,y_n) \to \bigwedge_{i=1}^{n} x_i = y_i\right) \\ (Acyc_c) & (\forall x_1,\ldots,x_n,a)\,(c(x_1,\ldots,x_n) \neq a) \text{ if } a \text{ occurs in some } x_i \\ \left(Disjoint_{c,d}\right) & (\forall x_1,\ldots,x_n,y_1,\ldots,y_n)\,(c(x_1,\ldots,x_n) \neq d(y_1,\ldots,y_n)) \end{array}$$

Let us illustrate this theory with an example to which we will come back frequently:

T_L: The theory of lists.

The signature of T_L is the set of function symbols $\{nil : list, car : list \to elem, cdr : list \to list, cons : elem \times list \to list\}$. The axioms of T_L are:

- $car(cons(x,y)) = x$
- $cdr(cons(x,y)) = y$

2.2.1 Bridging Functions on Absolutely Free Data Structures

In the following, we describe a way of relating an AFDS with elements of another kind, with a class of recursively defined *bridging functions*.

Consider an AFDS defined over Σ_S by the theory $AFDS_{\Sigma_S}$ and, on the other hand, a sort `t`, different or not from the ones in Σ_S, and a theory T_T over a signature Σ_T

with the sort `t`. We consider an extension of the disjoint combination of $AFDS_{\Sigma_S}$ and T_T.
Let $\Sigma = \Sigma_S \cup \Sigma_T \cup \{f\}$, and Rec_f a Σ-theory defining a function f over the data structure that returns elements of sort `t`. If $x_1, \ldots, x_n$ are `struct` variables, and $y_1, \ldots, y_m$ are `t`-variables, the recursive definition of the functions is as follows:

$$Rec_f = \begin{cases} f(k) = f_k \\ f(c(x_1, \ldots, x_n, y_1, \ldots, y_m)) = g^{c,f}(f(x_1), \ldots, f(x_n), y_1, \ldots, y_m) \end{cases}$$

for each k constant in $Cons_{\Sigma_S}$ and c functions of arity $n+m$ in $Cons_{\Sigma_S}$, and $g^{c,f}$ is a Σ_T-function.

An example extending the Lists theory

We first let the sort `t` be $\mathcal{Z}_0^+$, the theory of linear arithmetic over the non-negative integers. We now define T_{LS}, a theory of lists endowed with length. The many-sorted signature of T_{LS} is $\Sigma_L \cup \Sigma_{\mathcal{Z}}$, plus the function symbol $l : lists \to \mathcal{Z}_0^+$. The theory T_{LS} is the union of T_L and $\mathcal{Z}_0^+$, plus Rec_l:

- $l(nil) = 0$
- $l(cons(a, y)) = l(y) + 1$

2.3 Some decidable first order theories

A decidable theory T is a theory such that the T-satisfiability problem for sets of literals in the language of T is decidable, this is, an algorithm can be written in order to let a computer decide if a finite set of literals is T-satisfiable. We here show some examples of decidable theories and explain briefly the existing methods.

- Equality with uninterpreted functions ($T_{\mathcal{E}}$): it is the empty theory with no axioms, that is, $T_{\mathcal{E}} = \emptyset$. Of course, due to the undecidability of first-order logic, the validity problem for $T_{\mathcal{E}}$ is undecidable for quantifier free formulae. However, Ackermann [1] proved that the quantifier-free validity problem for $T_{\mathcal{E}}$ is decidable. There exist efficient decision procedures based on congruence closure.
- The theory of linear arithmetic ($\mathcal{Z}$): Defined over the signature $\Sigma_{\mathcal{Z}} = \langle +, -, \leq \rangle$ extended with a constant symbol c_n for each integer n, it is the set of closed formulas in $\mathcal{Z}$ that are true in the interpretation $\mathcal{A}$ whose domain A is the set $\mathcal{Z} = \{0, \pm 1, \pm 2, \ldots\}$ of integers, and interpreting the symbols in $\mathcal{Z}$ according to their standard meaning over $\mathcal{Z}$. This theory was proven decidable by Presburger in 1929, using quantifier elimination [38]. This technique consists in

converting a closed formula φ into a $\mathcal{Z}$ -equivalent closed formula ψ without quantifiers, which can be effectively computed as it is a boolean combination of ground atoms. However, if we add multiplication, we obtain an undecidable theory [12]. [1]

- The theory of linear arithmetic over reals ($\mathcal{R}$): Defined over the same $\Sigma_{\mathcal{Z}}$, plus a constant c_r for each rational number $r \in \mathcal{Q}$. The theory is defined analogously as for $T_{\mathcal{Z}}$, but for the meaning of symbols over $\mathcal{R}$. The validity problem for $\mathcal{R}$ can also be proved to be decidable using quantifier elimination [20]. In this case, if we add the multiplication symbol $\times$ to $\Sigma_{\mathcal{R}}$, then the validity problem for the resulting theory is decidable. This result was proved by Tarski [48] using quantifier elimination.

- Recursively defined data structures. Recursively defined data structures are defined over signatures which include function symbols called *constructors*, and some other ones, for which there are axioms that define them. An example of recursively defined data structures is the Absolutely Free Data Structures presented in Section 2.2, with lists as a more specific example. There exist general decision procedures for this kind of theories. For efficiency, theory-specific methods are often used. To give an example, the problem of deciding the satisfiability of conjunctions of literals under the theory of lists is solvable in linear time [34].

- Löwenheim theories: finite sets of closed formulas in a relational language containing only unary predicates, and no functions except constants). This class is also known as first-order relational monadic logic. The decidability of this first-order fragment was established by Leopold Löwenheim in 1915 [27].

2.4 Combination of Satisfiability Procedures

Until now, we have always defined decision problems under just one theory. However, in practice we often need to check satisfiability of a formula under more than one theory. This is especially common when verifying software: specifications are built using many different structures, each of them having its own theory.

Checking satisfiability under T_1 and T_2 would be as easy as checking it under $T_1 \cup T_2$ if we had a general method, but unfortunately we do not. Recalling that we use theory-specific methods to decide satisfiability, the question is whether we can combine them to get a decision procedure for the union. If we know φ is satisfiable under T_1 and under T_2, can we conclude it is for the union? Not in the general case.

For example, we could be interested in checking

$$\varphi : x - y \neq 0 \wedge x = f(a) \wedge y = f(b) \wedge a = b$$

[1] Notice that this makes $\mathcal{Z}_0^+$ also decidable.

modulo $\mathcal{Z} \cup T_{\mathcal{E}}$. We can divide φ in two formulas, each of them corresponding to one theory's language, and check satisfiability separately. In this case, we have

- $\varphi_{\mathcal{Z}} : x - y \neq 0$. It is satisfiable in Z, taking $x \neq y$
- $\varphi_{T_{\mathcal{E}}} : x = f(a) \wedge y = f(b) \wedge a = b$. It is satisfiable in the theory of equality, taking $x = y$.

But clearly they are not satisfiable in the union of theories, as they require opposite truth values for $x = y$.

The question is how we can deal with modularity in decision procedures. Having a decision procedure P_1 for a theory T_1, and another one, P_2, for a theory T_2, when and how can we obtain a decision procedure P for $T_1 \cup T_2$? The following theorem provides an answer to this question.

Theorem 2. ***(Combination Theorem)***
Let Σ_1 and Σ_2 be signatures. Let Γ_i be a set of Σ_i-formulae and $\mathcal{V}_i = Var(\Gamma_i)$ for $i = 1, 2$. Then $\Gamma_1 \cup \Gamma_2$ is satisfiable if and only if there exists a Σ_1-interpretation $\mathcal{A}$ over $\mathcal{V}_1$ satisfying Γ_1 and a Σ_2-interpretation $\mathcal{B}$ over $\mathcal{V}_2$ satisfying Γ_2 such that $\mathcal{A}^{\Sigma_1 \cap \Sigma_2, \mathcal{V}_1 \cap \mathcal{V}_2} \simeq \mathcal{B}^{\Sigma_1 \cap \Sigma_2, \mathcal{V}_1 \cap \mathcal{V}_2}$.

The proof of this theorem uses the isomorphism to build a model of $\Gamma_1 \cup \Gamma_2$ from one of each theory separately. Details can be found in [28].

Theorem 2 is useful to analyse the cases we are interested in. As the approach is modular, we can suppose we have two theories, and solve the case for multiple ones by applying the theorem repeatedly.

To explain how we can use this theorem to solve a satisfiability problem modulo a number of theories, first let us introduce a definition.

Definition 12. *Let $\Sigma = \bigcup_{i=1}^{n} \Sigma_i$ be a union of signatures. A* pure *literal, formula, or term is one that has only symbols from one signature Σ_i, and variables.*

We can also define purity for literals respecting sorts, for many-sorted signatures. Let Σ be a many-sorted first-order signature with sorts $S = \{\tau_1, \ldots, \tau_n\}$. Then a Σ-literal is *pure* if it contains only elements from one sort, that is, if it is an equalities of τ_i-terms or variables, or a predicate applied to τ_i-terms or variables. If a literal (or anything else) is not pure, it is called *mixed*. A set of literals is pure if all the literals in it are pure, and from the same sort.

Notice that we can always translate a mixed set of literals to sets of pure literals maintaining satisfiability. We introduce the procedure for separating sorts, of course the one for signatures is equivalent considering each signature of a different sort. The idea is to introduce fresh variables whenever a literal has terms of different

sorts, in order to replace one of the terms. We later add an equality between the new variable and the replaced term, which is also a pure literal. As an example, consider t_1 is a τ_1-term, t_2 is a τ_2-term, and f is a τ_1, τ_2 function symbol. If we find the literal $f(t_1) = t_2$, we can change it for $f(x) = t_2$ (which fits in φ_{τ_2}) and add $x = t_1$ (which fits in φ_{τ_1}). A detailed explanation of a procedure like this (but for an specific case) can be found in [51]. We will refer to this step as *purification*.

With this procedure in hand, we see that if we want to check satisfiability of φ under $T_1 \cup T_2$, we can always purify φ, dividing it into an equisatisfiable conjunction of two pure formula: the Σ_1-formula φ_1, and the Σ_2-formula φ_2. Then we can use the Combination Theorem applied to $\{\varphi_1\} \cup T_1$ and $\{\varphi_2\} \cup T_2$, provided the conditions are met. This will depend on what the reduct of the models look like; that is, what is shared by the signatures of both theories.

2.4.1 Disjoint Combination: The Nelson-Oppen procedure

When the signatures of the theories we want to combine are disjoint, that is, they share only the equality and finitely many variables, we only need to prove there exist models such that $\mathcal{A}^{\emptyset,\mathcal{V}_1\cap\mathcal{V}_2} \simeq \mathcal{B}^{\emptyset,\mathcal{V}_1\cap\mathcal{V}_2}$, in the nomenclature of Theorem 2. The following theorem sets the conditions for this, but first let us introduce a definition.

Definition 13. *An arrangement Arr for a set of variable symbols S is a maximal satisfiable set of equalities and inequalities $a = b$ or $a \neq b$, with $a, b \in S$*

Theorem 3. *Let T_1 and T_2 be theories over the disjoint signatures Σ_1 and Σ_2 respectively. Let L_i $(i = 1, 2)$ be a set of literals in Σ_i, and call $\mathcal{V}$ the set of shared variables between these sets. Then $L_1 \cup L_2$ is $T_1 \cup T_2$-satisfiable if and only if there exist an arrangement Arr of $\mathcal{V}$, a cardinality k, and a T_i-model $\mathcal{M}_i$ of $Arr \cup L_i$ with cardinality k for $i = 1, 2$.*

Proof. We prove the *if* and the *only if* direction.

- *If direction:* Let $\mathcal{M}_i$ be a model with domain M_i of $Arr \cup L_i$ under T_i for $i = 1, 2$. Then $\mathcal{M}_i$ is a model of $T_i \cup L_i$. As the signatures are disjoint, in order to apply Theorem 2 we need to prove there exists an isomorphism between $\mathcal{M}_1^{\mathcal{V}}$ and $\mathcal{M}_2^{\mathcal{V}}$. We can define an isomorphism between the elements in $\mathcal{V}$ as both models satisfy the arrangement Arr, and it can be extended to the entire domains as they have the same cardinality. As the models with isomorphic reducts exist, Theorem 2 can be used to obtain that, $T_1 \cup L_1 \cup T_2 \cup L_2$ is satisfiable, that is $L_1 \cup L_2$ is $T_1 \cup T_2$-satisfiable.

- *Only if direction:* If $\mathcal{M}$ is a $T_1 \cup T_2$-model of $L_1 \cup L_2$, then we can build an arrangement Arr of $\mathcal{V}$ just by reproducing the equivalence relation in $\mathcal{M}$. Then $\mathcal{M}_i = \mathcal{M}^{\Sigma_i}$ is a T_i-model of $L_i \cup Arr$ for $i = 1, 2$.

□

Following this idea, we can present the classical procedure developed by Nelson and Oppen to solve the satisfiability problem for combinations of theories with disjoint signatures. Before, we define a class of theories for which the cardinality assumption of Theorem 3 holds.

Definition 14. *A theory T is **stably infinite** if, whenever φ is T-satisfiable, φ is satisfiable in an infinite model of T.*

Nelson-Oppen procedure Let T_i be a stably infinite Σ_i-theory for $i = 1, 2$, and $\Sigma_1 \cap \Sigma_2 = \emptyset$. Let $\Sigma = \Sigma_1 \cup \Sigma_2$, and let φ be a quantifier-free Σ-formula. The following procedure checks satisfiability of φ under $T_1 \cup T_2$.

1. **Purify input** by dividing φ into φ_1 and φ_2
2. **Guess an arrangement** Arr between the shared variables in φ_1 and φ_2.
3. **Check satisfiability** of $\varphi_1 \cup Arr$ modulo T_1 and of $\varphi_2 \cup Arr$ modulo T_2
   - If they are both satisfiable, return `satisfiable`
   - Otherwise return `unsatisfiable`

It is worth including a remark about the second step of the algorithm. We will frequently use guessings of arrangements of predicates over a set of variables. This means that if there exists a valuation of the predicate for each combination of variables that makes the algorithm return `sat`, the procedure will find it. This method can always be used safely if the we make finite guessings. In this case it is guaranteed to terminate, as it is equivalent to trying all possible combinations.

Proof. On one side, if we can find a model $\mathcal{M}$ of $T_1 \cup T_2 \cup \{\varphi\}$, then we consider the equivalence relation for the shared variables in φ_1 and φ_2 to build an arrangement Arr. Now, $\mathcal{M}$ must model $T_1 \cup \varphi_1 \cup Arr$ and $T_2 \cup \{\varphi_2\} \cup Arr$ as well.

For the other direction, Theorem 3 can be applied; if the procedure returns `satisfiable`, then the second condition of the theorem holds, as there exist two models for the theories agreeing in an equivalence relation for the shared variables. In addition, as T_1 and T_2 are stably infinite, $T_1 \cup \{\varphi_1\}$ and $T_1 \cup \{\varphi_2\}$ both have infinite models. Recalling the Löwenheim–Skolem theorem, we obtain that they are satisfiable in an infinite model of any cardinality, so the first condition holds as well. □

Limitations Nelson-Oppen's method imposes two conditions:

1. the theories have to be stably infinite
2. the signatures of both theories have to be disjoint

These are strong limitations, and many recent advances aim to go beyond disjointness and stable infiniteness.

In this work, we analyse specially how to deal with combinations of theories in signatures sharing some symbols. If there are shared function or predicate symbols, Nelson-Oppen procedure is not sufficient to decide if the conditions to apply the Combination Theorem hold. In the following sections we discuss two different approaches to this problem.

3 Non-Disjointness via Unary Predicates

As a first approach we will analyse how the Nelson-Oppen method can be extended to consider theories where only certain symbols are shared. The Nelson-Oppen combination method requires the theories in the combination to be (1) signature-disjoint and (2) stably infinite. The efforts to go beyond these limitations should not be opposed, as in both cases we know, from the Combination Theorem, that we need to find out if there exists an isomorphism between the restrictions of two models of the theories to the shared signature.

In the particular case of disjoint theories, the isomorphism can be obtained if the domains of the models have the same cardinality, for instance infinite; several classes of *kind* theories (shiny [49], polite [40], gentle [18]) have been introduced to enforce a (same) domain cardinality on both sides of the combination. For extensions of Nelson-Oppen to non-disjoint cases, e.g. in [42, 53], cardinality constraints also arise. In this chapter, we focus on non-disjoint combinations for which the isomorphism can be simply constructed by satisfying some cardinality constraints.

The idea is to introduce a class of $\mathcal{P}$-*gentle* theories, to combine theories sharing a finite set of unary predicates symbols $\mathcal{P}$. The notion of $\mathcal{P}$-gentle theory extends the one introduced for the disjoint case [18], where it was initially presented as a tool to combine non-stably infinite disjoint theories.

Roughly speaking, a $\mathcal{P}$-gentle theory has nice cardinality properties not only for domains of models but also more locally for all Venn regions of shared unary predicates. We introduce this notion and present a combination method for unions of $\mathcal{P}$-gentle theories sharing $\mathcal{P}$ in Section 3.1. The proposed method can also be used to combine a $\mathcal{P}$-gentle theory with another arbitrary theory for which we assume the decidability of satisfiability problems with cardinality constraints (entailed by the $\mathcal{P}$-gentle theory). This is a natural extension of previous works on combining non-stably infinite theories, in the straight line of combination methods à la Nelson-Oppen. In Section 3.2, we show that all the theories in the Löwenheim class are $\mathcal{P}$-gentle. Our combination framework is a way to combine theories with sets. The relation between (monadic) logic and sets is as old as logic itself, and this relation is particularly clear for instance considering Aristotle Syllogisms. It is however useful to again study monadic logic, and more particularly the Löwenheim class, specially with the recent advances in combinations with non-disjoint and non-stably infinite

theories. Theories in the BSR class are also $\mathcal{P}$-gentle. For a detailed proof of this, refer to the article where we expose the contributions in this chapter [11]. A simple example is given in Section 3.3, and in Section 3.4 we discuss limitations and possibilities of extending the approach for theories sharing other symbols.

3.1 Gentle Theories Sharing Unary Predicates

First of all, we will introduce some useful definitions. Consider an interpretation $\mathcal{I}$ on a language with unary predicates $p_1, \ldots, p_n$ and some elements D in the domain of this interpretation. Every element $d \in D$ belongs to a Venn region[1] $v(d) = v_1 \ldots v_n \in \{\top, \bot\}^n$ where $v_i = \mathcal{I}[p_i](d)$. We denote by $D_v \subseteq D$ the set of elements of D in the Venn region v. Notice also that, for a language with n unary predicates, there are 2^n Venn regions.

Given an interpretation $\mathcal{I}$, we will also denote by D^c the subset of elements in D associated to constants by $\mathcal{I}$. Naturally, D^c_v denotes the set of elements associated to constants that are in the Venn region v.

From now on, we assume that $\mathcal{P}$ is a non-empty finite set of unary predicates. A *$\mathcal{P}$-union* of two theories T_1 and $\mathcal{T}_2$ is a union sharing only $\mathcal{P}$ from the signature, in addition to a set of variables and the equality.

Definition 15. *A $\mathcal{P}$-arrangement $\mathcal{A}$ for finite sets of constant symbols S and unary predicates $\mathcal{P}$ is a maximal satisfiable set of equalities and inequalities $a = b$ or $a \neq b$ and literals $p(a)$ or $\neg p(a)$, with $a, b \in S$, $p \in \mathcal{P}$.*

There are only a finite number of $\mathcal{P}$-arrangements for given sets S and $\mathcal{P}$.

Given a theory $\mathcal{T}$ whose signature includes $\mathcal{P}$ and a model of $\mathcal{T}$ on domain D, the *$\mathcal{P}$-cardinality* $\vec{\kappa}$ is the tuple of cardinalities of all Venn regions of $\mathcal{P}$ (κ_v will denote the cardinality of the Venn region v). The following theorem, which is an specialization of general combination lemmas in Section 2.4, states the completeness of the combination procedure for $\mathcal{P}$-unions of theories:

Theorem 4. *Consider a $\mathcal{P}$-union of theories T_1 and T_2 whose respective signatures Σ_1 and Σ_2 share a finite set S of constants, and let L_1 and L_2 be sets of literals, respectively in Σ_1 and Σ_2. Then $L_1 \cup L_2$ is $T_1 \cup T_2$-satisfiable if and only if there exist a $\mathcal{P}$-arrangement $\mathcal{A}$ for S and $\mathcal{P}$, and a T_i-model $\mathcal{M}_i$ of $\mathcal{A} \cup L_i$ with the same $\mathcal{P}$-cardinality for $i = 1, 2$.*

The *spectrum* of a theory $\mathcal{T}$ is the set of $\mathcal{P}$-cardinalities of its models. The above theorem can thus be restated as:

[1] Other works refer to this notion as the set of elements with a same 1-table, 1-type, or color.

Corollary 1. *The $T_1 \cup T_2$-satisfiability problem for sets of literals is decidable if, for any sets of literals L_1 and L_2 it is possible to decide if the intersection of the spectrums of $T_1 \cup \mathcal{A} \cup L_1$ and of $T_2 \cup \mathcal{A} \cup L_2$ is non-empty.*

To characterize the spectrum of the decidable classes considered in this chapter, we introduce the notion of *cardinality constraint*. A *finite* cardinality constraint is simply a $\mathcal{P}$-cardinality with only finite cardinalities. An *infinite* cardinality constraint is given by a $\mathcal{P}$-cardinality $\vec{\kappa}$ with only finite cardinalities and a non-empty set of Venn regions V, and stands for all the $\mathcal{P}$-cardinalities $\vec{\kappa}'$ such that $\kappa'_v \geq \kappa_v$ if $v \in V$, and $\kappa'_v = \kappa_v$ otherwise. The *spectrum* of a finite set of cardinality constraints is the union of all $\mathcal{P}$-cardinalities represented by each cardinality constraint. It is now easy to define the class of theories we are interested in:

Definition 16. *A theory T is* $\mathcal{P}$-gentle *if for every set L of literals in the signature of T, the spectrum of $T \cup L$ is the spectrum of a computable set of cardinality constraints.*

Notice that a $\mathcal{P}$-gentle theory is (by definition) decidable. To relate the above notion with the gentleness in the disjoint case [18], observe that if p is a unary predicate symbol not occurring in the signature of the theory T, then $T \cup \{(\forall x)p(x)\}$ is $\{p\}$-gentle if and only if $\mathcal{T}$ is gentle.

If a theory is $\mathcal{P}$-gentle, then it is $\mathcal{P}'$-gentle for any non-empty subset $\mathcal{P}'$ of $\mathcal{P}$. It is thus interesting to have $\mathcal{P}$-gentleness for the largest possible $\mathcal{P}$. Hence, when $\mathcal{P}$ is not explicitly given for a theory, we assume that $\mathcal{P}$ denotes the set of unary predicates symbols occurring in its signature. In the following sections we show that the Löwenheim theories and the BSR theories are $\mathcal{P}$-gentle.

The union of two $\mathcal{P}$-gentle theories is decidable, as a corollary of the following modularity result:

Theorem 5. *The class of $\mathcal{P}$-gentle theories is closed under $\mathcal{P}$-union.*

Proof. If we consider the $\mathcal{P}$-union of two $\mathcal{P}$-gentle theories with respective spectrums $\mathcal{S}_1$ and $\mathcal{S}_2$, then we can build some finite set of cardinality constraints whose spectrum is $\mathcal{S}_1 \cap \mathcal{S}_2$ □

Some very useful theories are not $\mathcal{P}$-gentle, but in practical cases they can be combined with $\mathcal{P}$-gentle theories. To define more precisely the class of theories T' that can be combined with a $\mathcal{P}$-gentle one, let us introduce the *T'-satisfiability problem with cardinality constraints*: given a formula and a finite set of cardinality constraints, the problem amounts to check whether the formula is satisfiable in a model of T whose $\mathcal{P}$-cardinality is in the spectrum of the cardinality constraints. As a direct consequence of Corollary 1, $T \cup T'$-satisfiability is decidable if T is $\mathcal{P}$-gentle and T'-satisfiability with cardinality constraints is decidable. This latter requirement is satisfied by the theories we can usually find in SMT solvers. This

gives the theoretical ground to add to the SMT solvers any number of $\mathcal{P}$-gentle theories sharing unary predicates.

It follows that the non-disjoint union (sharing unary predicates) of BSR and Löwenheim theories, with one decidable theory accepting further constraints of the form $(\forall x)((\neg)p_1(x) \wedge \cdots \wedge (\neg)p_n(x)) \Rightarrow (x = a_1 \vee \cdots \vee x = a_m)$ is decidable. For instance, the guarded fragment with equality accepts such further constraints and the superposition calculus provides a decision procedure [22]. Thus any theory in the guarded fragment can be combined with Löwenheim and BSR theories sharing unary predicates.

In the disjoint case, any decidable theory expressed as a finite set of first-order axioms can be combined with a gentle theory [18]. Here this is not the case anymore. Indeed, consider the theory $\psi = \varphi \vee \exists x\, p(x)$ where p does not occur in φ; any set of literals is satisfiable in the theory ψ if and only if it is satisfiable in the theory of equality. If the satisfiability problem of literals in the theory φ is undecidable, the $\mathcal{P}$-union of ψ and the Löwenheim theory $\forall x\, \neg p(x)$ will also be undecidable.

3.2 Classes of gentle theories: The Löwenheim Class and the BSR Class

3.2.1 The Löwenheim Class

A Löwenheim theory is a finite set of closed formulas in a relational language containing only unary predicates (and no functions except constants). This class is also known as first-order relational monadic logic. Usually one distinguishes the Löwenheim class with and without equality. The Löwenheim class has the finite model property (and is thus decidable) even with equality. Full monadic logic *without equality*, i.e. the class of finite theories over a signature containing symbols (predicates and functions) of arity at most 1, also has the finite model property. Considering monadic logic with equality, the class of finite theories over a signature containing only unary predicates and just two unary functions is already undecidable. With only one unary function, however, the class remains decidable [9], but does not have the finite model property anymore. Since the spectrum for this last class is significantly more complicated [25] than for the Löwenheim class we will here only focus on the Löwenheim class with equality (only classes with equality are relevant in our context), that is, without functions. More can be found about monadic first-order logic in [9, 15]. In particular, a weaker version of Corollary 2 (given below) can be found in [15].

Previously [18, 2], combining theories with non-stably infinite theories took advantage of "pumping" lemmas, allowing — for many decidable fragments — to build models of arbitrary large cardinalities. The following theorem is such a pumping lemma, but it considers the cardinalities of the Venn regions and not only the global

cardinality.

Lemma 1. *Assume T is a Löwenheim theory with equality. Let q be the number of variables in T. If there exists a model $\mathcal{M}$ on domain D with $|D_v \setminus D^c| \geq q$, then, for each cardinality $q' \geq q$, there is a model extension or restriction $\mathcal{M}'$ of $\mathcal{M}$ on domain D' such that $|D'_v \setminus D^c| = q'$ and $D'_{v'} = D_{v'}$ for all $v' \neq v$.*

Proof. Two interpretations $\mathcal{I}$ (on domain D) and $\mathcal{I}'$ (on domain D') for a formula ψ are *similar* if

- $|(D_v \cap D'_v) \setminus D^c| \geq q$;
- $D_{v'} = D'_{v'}$ for each Venn region v' distinct from v;
- $\mathcal{I}[a] = \mathcal{I}'[a]$ for each constant in ψ;
- $\mathcal{I}[x] = \mathcal{I}'[x]$ for each variable free in ψ.

Notice that $\mathcal{M}$ and $\mathcal{M}'$ above are similar. We prove that, given a Löwenheim formula ψ (or a set of formulas), two similar interpretations for ψ give the same truth value to ψ and to each sub-formula of ψ. This will suffice to prove the lemma.

The proof is by induction on the structure of the (sub-)formula ψ. It is obvious if ψ is atomic, since similar interpretations assign the same value to variables and constants. If ψ is $\neg\varphi_1$, $\varphi_1 \vee \varphi_2$, $\varphi_1 \wedge \varphi_2$ or $\varphi_1 \Rightarrow \varphi_2$, the result holds if it also holds for φ_1 and φ_2.

Assume $\mathcal{I}$ makes true the formula $\psi = \exists x\, \varphi(x)$. Then there exists some $d \in D$ such that $\mathcal{I}_{x/d}$ is a model of $\varphi(x)$. If $d \in D'$, then $\mathcal{I}'_{x/d}$ is similar to $\mathcal{I}_{x/d}$ and, by the induction hypothesis, it is a model of $\varphi(x)$; $\mathcal{I}'$ is thus a model of ψ. If $d \notin D'$, then $d \in D_v$ and $|(D_v \cap D'_v) \setminus D^c| \geq q$. Furthermore, since the whole formula contains at most q variables, $\varphi(x)$ contains at most $q-1$ free variables besides x. Let $x_1, \ldots, x_m$ be those variables. There exists some $d' \in (D_v \cap D'_v) \setminus D^c$ such that $d' \neq \mathcal{I}[x_i]$ for all $i \in \{1, \ldots, m\}$. By structural induction, it is easy to show that $\mathcal{I}_{x/d}$ and $\mathcal{I}_{x/d'}$ give the same truth value to $\varphi(x)$. Furthermore $\mathcal{I}_{x/d'}$ and $\mathcal{I}'_{x/d'}$ are similar. $\mathcal{I}'$ is thus a model of ψ. To summarize, if $\mathcal{I}$ is a model of ψ, $\mathcal{I}'$ is also a model of ψ. By symmetry, if $\mathcal{I}'$ is a model of ψ, $\mathcal{I}$ is also a model of ψ. The proof for formulas of the form $\forall x\, \varphi(x)$ is analogous. □

The above lemma has the following consequence on the acceptable cardinalities for the models of a Löwenheim theory:

Corollary 2. *Assume T is a Löwenheim theory with equality with n distinct unary predicates. Let r and q be respectively the number of constants and variables in T. If T has a model of some cardinality κ strictly larger than $r + 2^n \max(0, q-1)$, then T has models of each cardinality equal or larger than $\min(\kappa, r + q\, 2^n)$.*

Proof. If a model with such a cardinality exists, then there are Venn regions v such that $|D_v \setminus D^c| \geq q$. Then the number of elements in these Venn regions can be increased to any arbitrary larger cardinality, thanks to the previous Lemma. If $\kappa > r + q\,2^n$, it means some Venn regions v are such that $|D_v \setminus D^c| > q$, and by eliminating elements in such Venn regions (using again the previous Lemma), it is possible to obtain a model of cardinality $r + q\,2^n$. □

To conclude, Lemma 2 implies the decidability of Löwenheim theories, but also directly entails the $\mathcal{P}$-gentleness:

Theorem 6. *Löwenheim theories on a signature with unary predicates in $\mathcal{P}$ are $\mathcal{P}$-gentle.*

3.2.2 The Bernays-Schönfinkel-Ramsey Class

A Bernays-Schönfinkel-Ramsey (BSR for short) theory is a finite set of formulas of the form $\exists^*\forall^*\varphi$, where φ is a first-order formula which is function-free (but constants are allowed) and quantifier-free. Bernays and Schönfinkel first proved the decidability of this class without equality; Ramsey later proved that it remains decidable with equality. More can be found about BSR theories in [9]. Ramsey also gave some (less known) results about the spectrum of BSR theories [39].

BSR theories can be proved to be $\mathcal{P}$-gentle. The idea of the proof is to show that, for each Venn region, there is a computable number k such that if $\mathcal{T}$ has a model of cardinality greater or equal to k, then it has a model of any cardinality larger than k. This was already proved for the general cardinality of BSR theories in [39, 18]. To show it for each Venn region, it is necessary to define very technical instrumental notions and to prove a non-trivial extension of Ramsey's Theorem. This contribution can be found in detail in [11].

3.3 Example: Non-Disjoint Combination of Order and Sets

To illustrate the kind of theories that can be handled in our framework, consider a very simple yet informative example with a BSR theory defining an ordering $<$ and augmented with clauses connecting the ordering $<$ and the sets p and q (we do not distinguish sets and their related predicates):

$$\mathcal{T}_1 = \begin{cases} (\forall x)\ \neg(x < x) \\ (\forall x, y)\ (x < y \wedge y < z) \Rightarrow x < z \\ (\forall x, y)\ (p(x) \wedge \neg p(y)) \Rightarrow x < y \\ (\forall x, y)\ (q(x) \wedge \neg q(y)) \Rightarrow x < y \end{cases}$$

and a Löwenheim theory

$$\mathcal{T}_2 = \{\forall x.\ (p(x) \wedge q(x)) \equiv x = c\}$$

stating that the intersection of the sets p and q is the singleton $\{c\}$.

The first theory imposes either $p \cap \overline{q}$ or $\overline{p} \cap q$ to be empty (we will assume that the domain is non-empty and simplify the cardinality constraints accordingly). The second theory obviously imposes the cardinality of $p \cap q$ to be exactly 1. Notice that both theories are actually $\mathcal{P}$-gentle. The following table collects the cardinality constraints:

	$\mathcal{T}_1$		$\mathcal{T}_2$
$\overline{p} \cap \overline{q}$	≥ 0	≥ 0	≥ 0
$\overline{p} \cap q$	0	≥ 0	≥ 0
$p \cap \overline{q}$	≥ 0	0	≥ 0
$p \cap q$	≥ 0	≥ 0	1

Assume now that we have two literals $p(a)$, $q(b)$ (these can again be considered as a further non-disjoint theory). Since either $p \cap \overline{q}$ or $\overline{p} \cap q$ is empty, either a or b belongs to the intersection $p \cap q$. Hence, the set

$$\mathcal{T}_1 \cup \mathcal{T}_2 \cup \{p(a), q(b), a \neq c, b \neq c\}$$

is unsatisfiable.

As a final comment, there could be theories using directly the Venn cardinalities as integer variables. Consider again $\mathcal{T}_1 \cup \mathcal{T}_2$, one could imagine a further constraint in another theory including linear arithmetic on integers that would state $|p| > 1$ and $|q| > 1$. This would of course be unsatisfiable with $\mathcal{T}_1 \cup \mathcal{T}_2$.

3.4 Discussion

This approach to solving non-disjointness needs to take advantage of the properties of the shared symbols in order to find a way to decide if the theories are combinable or not, regarding cardinality constraints. Therefore, allowed shared symbols are very restricted. Our combination method is limited to shared unary predicates. Unfortunately, the theoretical limitations are strong for a framework sharing predicates with larger arities: for instance even the guarded fragment with two variables and transitivity constraints is undecidable [21], although the guarded fragment (or first-order logic with two variables) is decidable, and transitivity constraints can be expressed in BSR. When considering a general shared binary predicate, the restriction in the theories is that there exist a model for each of them in which the "graphs" induced by the binary relation are isomorphic. This is much more difficult to decide, and it seems not easy to find a general way of expressing the

conditions that makes it true. For some specific topologies of relations, however, the problem is easier. For example, the problem of combining theories with only a shared dense order has been successfully solved [23, 28]. In that specific case, there is again an implicit infiniteness argument that could be possibly expressed as a form of extended gentleness, to reduce the isomorphism construction problem into solving some appropriate extension of cardinality constraints. A clearly challenging problem that deserves being explored is to identify an appropriate extended notion of gentleness for some particular binary predicates.

When the shared signature includes function symbols, the outlook is even less promising. New fundamental problems appear, since it is not possible to perform a finite guess in the involved variables anymore, as we can apply a function an arbitrary number of times. Therefore we may have to exchange infinitely many literals between the theories to find if an isomorphism is possible. Briefly, we need to check if the shared possible terms match. Suppose we have a shared unary function f and a variable x. It could happen that $f(x) = x$, or that $f(f(x)) = x$, and so on, meaning that the arrangements are not finite anymore. This is exactly why a Nelson-Oppen-like method for sharing functions is not guaranteed to always terminate.

4 Non-Disjointness via Bridging Functions

Obtaining a general Nelson-Oppen-like combination method for the non-disjoint case seems to be at least complicated. Some examples exist, as we mention in the introduction, but they make use of very theoretical notions to define the framework, and it becomes difficult to find examples of applications. For this reason, it is worth exploring specific classes of non-disjoint combinations of theories that appear frequently in software specification, and for which it would be useful to have a procedure. In this section we analyse one of these cases.

We will study non-disjointness when it arises from connecting elements in one theory with elements in another one via some functions, that we will call *bridging functions*. Let T_1 and T_2 be two disjoint theories over many-sorted signatures with some sorts τ_1 and τ_2 respectively, for which we have decision procedures. Suppose we define some function from elements of sort τ_1 to elements of sort τ_2; now we have a theory T, built from T_1, T_2 and the definition of the function. The question is if we can reuse somehow the available decision procedures for T_1 and T_2 to build one for T. This is a very commonly seen case, specially when bridging functions are defined from a data structure to some other one, which is very usual in software specifications. In this case, defining bridging functions is a natural way of extending the data structure and relating it to other sorts. In this section we will use many-sorted first-order logic, as we will allow the signatures to have other sorts besides the ones that are connected by the function.

A new Combination Problem Let Σ_s and Σ_t be many-sorted first-order logic signatures containing the sorts `source` and `target` respectively. Consider a Σ_s-theory T_s and a Σ_t-theory T_t, which are disjoint.

Let us define a function f : `source` $\rightarrow$ `target` relating elements from both sorts. This function is defined with some axioms, which are in the signature $\Sigma = \Sigma_s \cup \Sigma_t \cup \{f\}$. The theory containing the axioms that define f will be called T_f. Now consider the Σ-theory $T = T_s \cup T_f \cup T_t$, that extends the previous two disjoint theories with the definition of this new function.

To illustrate this bridging function idea, recall the functions over absolutely free data structures that we presented in Section 2.2.1, and specifically the example of length defined over lists. In this case, T_L is the source theory and $\mathcal{Z}_0^+$ the target

one, and the two axioms defining the length conform T_f. T_{LS} would be T, the theory that results from the union of these three theories.

Suppose we have decision procedures for T_s and T_t respectively, and we want to check satisfiability of a Σ-formula φ under T. We can express this as checking satisfiability under $T_s \cup T_f$, the source theory with the function definitions, and T_t simultaneously.

If we call the shared variables $\mathcal{V}$, the Combination Theorem states that the conjunction will be satisfiable if we can find a model $\mathcal{A}$ of $T_s \cup T_f \cup \varphi$ and a model $\mathcal{B}$ of T_t such that their reducts in the shared signature are isomorphic. Notice that the signature of $T_s \cup T_f \cup \varphi$ is $\Sigma = \Sigma_s \cup \Sigma_t \cup \{f\}$, and the signature of T_t is Σ_t. So we need:

$$\mathcal{A}^{(\Sigma_s \cup \Sigma_t \cup \{f\}) \cap \Sigma_t, \mathcal{V}} \simeq \mathcal{B}^{(\Sigma_s \cup \{f\}) \cap \Sigma_t, \mathcal{V}}$$

which is clearly

$$\mathcal{A}^{\Sigma_t, \mathcal{V}_{target}} \simeq \mathcal{B}^{\Sigma_t, \mathcal{V}_{target}}$$

since we know Σ_s and Σ_t are disjoint. In the variables, the shared part can be only of sort `target`.

Since $\mathcal{B}$ is already a Σ_t-model, this is actually $\mathcal{A}^{\Sigma_t, \mathcal{V}_{target}} \simeq \mathcal{B}$ Notice that this means that it is enough to find a model of T_t that is also a model of the part of $T_s \cup T_f \cup \{\varphi\}$ modelled by $\mathcal{A}^{\Sigma_t, \mathcal{V}_{target}}$. If we could isolate it this part, we could solve the problem easily.

Our goal will be to "extract" the Σ_t part of $T_s \cup T_f \cup \varphi$. Then, we will check satisfiability for this part under T_t; if it is satisfiable, then it clearly has an isomorphic model to one of T_t. Our aim in the following sections will be to analyse when we can divide $T_s \cup T_f \cup \{\varphi\}$ into a Σ_s set of literals and a Σ_t set of literals preserving global satisfiability.

A very simple approach: just replace. The difficulty of this problem depends on many factors, for example, how f is defined. In some cases there exists a very simple solution: as we are dealing with first order logic, we know f must appear in φ applied to some term, so we can just find each occurrence of f and replace it with the definition applied to that term.

This replacement procedure towards a disjoint problem is possible when the function f is defined over a variable in T_s. That is, T_f looks like $f(x) = c(x)$, for some expression $c \in \Sigma_t$. Sadly, this approach will not work for more complicated definitions: we cannot simply replace when considering functions defined by cases, or over some specific terms.

We introduce a procedure that proves itself useful in this case. The idea is to eliminate the function symbols, expressing all the information in them using just $\Sigma_s \cup \Sigma_t$. If we can do that maintaining satisfiability, we have reduced our problem to a disjoint one, and any of the methods we know for this problem can be used.

Structure of this chapter In Section 4.1 we present a general approach to translate a satisfiability problem under two theories connected by a bridging function into the problems for those two theories. This procedure is incomplete, as each specific class of theories will need its own adaptations, but it is useful as a general reference. For this reason, correctness is not proven here.

In Section 4.2 we provide an adaptation of the procedure for the classes of theories of Absolutely Free Data Structures that we introduced in 2.2 with a surjective bridging function as we defined it in 2.2.1. Zarba presented procedures for checking satisfiability of lists with length using just the procedure for arithmetic [52], as well as for multisets with multiplicity [50]. These procedures were aimed to relax the stably-infiniteness assumption in Nelson-Oppen procedure, as it is possible to use it with, for example, multisets with a finite set of elements. We generalize these approaches to consider general AFDSs. We use the AFDSs framework as is done in [44], where Sofronie-Stokkermans uses a locality property to show that the definition of the function connecting the theories can be eliminated. We show explicitly the procedure to do this, which could be implemented in an SMT solver.

In his work, Zarba deletes completely the theory of the data structure, as he considers it as just a container. In our approach we maintain it, making it possible to extend it to richer theories where, for example, we define functions between the data structure elements.

In Section 4.3.1, we consider the problem of restricting the models we are using to the standard interpretation of the data structures. Sofronie-Stokkermans already considers this problem, but her solution is very restrictive, as it discards the problematic cases in which cardinality problems might arise from not having enough elements to build structures that must be different. We face exactly that problem and find two solutions for different cases. In the first one, we consider a class of bridging functions called *stable*, for which the problem is solved easily. We present the extension of the procedure and the proof of correctness. This last one is made for the theory of lists with length, and we discuss how to extend it. The second case is for functions which are not stable. Here we present an algorithm which finds the problematic cases and decides if they are or not satisfiable. Again, it is presented for lists, and we clearly identify the theories for which we could extend it. This algorithm simplifies notably the approach to solve cardinality problems presented by [54].

In [46], the authors present a procedure to solve cardinality problems that is also based on a procedure for reducing bridging functions, and that works in the same

classes of theories as ours. It is worth saying that we developed this section without having notice of that article. In the conclusions, the authors suggest relations between their work and the one in [44] should be studied; Section 4.3.1 can be seen as an approach to that study.

Finally, in Section 4.4, we discuss the possibilities of application of our approach. We face two practical problems; one is to consider non-surjective functions, and the second one is how it could be applied modularly, which would be a very interesting and desirable property for these kind of procedures.

4.1 A procedure towards a disjoint problem

In this section we describe a general combination procedure for two theories connected via a bridging function. This procedure is aimed for cases when the function is not defined over simple variables but in more complex ways. We here present the general idea of the method; some parts will change with different classes of theories and functions to ensure correctness.

Consider two disjoint theories T_s and T_t over the disjoint signatures Σ_s and Σ_t containing the sorts `source` and `target` respectively, and a third theory T built from them and the axioms defining a bridging function $f : \texttt{source} \to \texttt{target}$, over $\Sigma = \Sigma_s \cup \Sigma_t \cup \{f\}$. We describe a decision procedure for checking the T-satisfiability of a quantifier-free Σ-formula φ. We can restrict ourselves without loss of generality to consider just conjunctions of literals. Indeed, as we can always translate a formula into its disjunctive normal form, and then solve the satisfiability problem for each disjunction separately.

First phase: Variable Abstraction and Partition The first phase of our decision procedure takes as input a set of mixed literals φ , and converts it into pure sets of flat literals. The output of the variable abstraction phase is a *separated* equisatisfiable formula $\varphi_s \cup \varphi_t \cup \varphi_f$ such that:

- φ_s contains only flat and pure Σ_s-literals. If Σ_s has many sorts $\tau_1, \ldots, \tau_n$, we will have one φ_{τ_i} for each one (one of which will be `source`), with pure τ_i literals, and $\varphi_s = \bigcup_{i=1}^{n} \varphi_{\tau_i}$.
- φ_t contains only flat Σ_t-literals. The same division by sorts holds here.
- φ_f contains only literals of the form $u = f(x)$

Note that flat literals can be obtained with a procedure similar to purification. The idea in this step is to have a separate variable for each element of sort `source` that is named in φ. That is, if we have the literal $f(f(x)) = y$, we will want to be able to name $f(f(x))$ and $f(x)$ separately, so we will replace it with $f(x) = z$ and $f(z) = y$.

We denote by V_τ the collection of τ-variables that either occur in φ or are generated in this phase. For each `source`-variable x, generate a new t-variable f_x.

Second phase: Decomposition In this phase, we build two sets of literals Γ_{source} and Γ_{target}.

We need to add in Γ_{source} the information of equivalence between elements of sort `source`. To do this, non-deterministically guess an arrangement Arr in V_{source}, and add it to Γ_{source}.

In Γ_{target}, add the collection of literals obtained by replacing all literals in $\varphi_s \cup \varphi_f \cup \Gamma_s$ with the following formulae for $x, y \in V_{source}$:

1. $x = y \rightarrow f_x = f_y$
2. $u = f(x) \rightarrow u = f_x$
3. for other literals, it will be defined later

This is the basic content of Γ_{source} and Γ_{target}. For different classes of theories, it may be necessary to add some other literals, which we will analyse in particular, for the procedure to be correct.

Third phase: Check The check consists in:

1. checking $\varphi_s \cup \Gamma_{source}$ for T_s-satisfiability
2. checking $\varphi_t \cup \Gamma_{target}$ for T_t-satisfiability

If both sets are satisfiable, φ is T-satisfiable.

The following step is to prove that the procedure is correct. The proof will of course vary with each particular case, as Γ_{source} and Γ_{target} will be different. Our goal will be, in all cases, to prove that the algorithm returns `sat` if and only if φ is T-satisfiable.

4.2 Bridging functions on absolutely-free data structures

Our procedure will be specially useful when bridging functions are not defined over variables, but over more complex constructions. A typical and extremely common example of this is recursive definitions. This is the natural way of defining functions over recursively defined data structures. We will focus in this specific case on bridging functions, which are useful in software specifications. In this section, we will work over the Absolutely Free Data Structures we defined in Section 2.2, when they are extended with a bridging function defined recursively over them like in Section 2.2.1 that connects them with another sort. We analyse how we can modify the procedure described in Section 4.1 to make it produce a set of safe literals, introducing some changes to the general procedure.

To simplify the analysis, we will suppose that the bridging function is surjective and the theory of the AFDS contains only constructors and selectors; no recursively defined functions. We will see how we can extend the procedure to relax these conditions in Section 4.4.

The procedure for AFDSs

Let $AFDS_{\Sigma_s}$ be the theory for an Absolutely Free Data Structures (Section 2.2) over a signature Σ_s with sorts `struct` and `elem` with no defined functions, T_t another theory over a signature Σ_t of sort `t`, and T a theory built adding to $AFDS_{\Sigma_s} \cup T_t$ the axioms to define a surjective bridging function $f : struct \to t$ (Section 2.2.1). In what follows we summarize the changes in each of the phases.

In the first phase, the original procedure will give us sets of pure literals φ_{struct}, φ_{elem}, φ_t and φ_f we need to restrict φ_{struct} to a few specific kind of literals. We will admit only literals of the form:

- $x = y$
- $x \neq y$
- $x = k$ with k an atom constructor in $Cons_{\Sigma_s}$
- $x = c(x_1, \ldots, x_n, a_1, \ldots, a_m)$, with c a function in $Cons_{\Sigma_s}$

Notice that we are not allowing literals including the *selectors* that can exist on an absolutely free data structure. This will simplify significantly the procedure. However, we need to provide a way to eliminate them from φ. This can always be done, as selectors are defined by pattern matching over the constructors, and in the right part of the definition there are only constructors or variables. The procedure

to eliminate them should be executed when the literals are already separated and flattened, and before adding the new variables for the function f in Γ_t, and would consist in:

- Take a literal in which a selector $sel(x)$ appears applied to some x.
- x must be a variable, as the literals are flattened. Then, for each definition of sel over a constructed list $c(x_1, \ldots, x_n, a_1, \ldots, a_m)$, create a new set of literals replacing $sel(x)$ in the literal by the right side of the definition and adding the new literal $x = c(x_1, \ldots, x_n, a_1, \ldots, a_m)$. This creates a new set of literals for each definition, for which we will continue with the procedure. Again, we will consider the original satisfiable if any of those are.

As another remark, we do not allow inequalities between a variable and a constructor, but this can be easily enforced by introducing a fresh variable.

In the second phase, we compute Γ_{struct} and Γ_t as in the procedure and then add in Γ_t the following replacements for the k atomic constructors in $Cons_{\Sigma_s}$ and c functions in $Cons_{\Sigma_s}$:

- $x = k \rightarrow f_x = f_k$
- $x = c(x_1, \ldots, x_n, a_1, \ldots, a_m) \rightarrow f_x = g^{c,f}(f_{x_1}, \ldots, f_{x_n}, a_1, \ldots, a_m)$

Correctness of the extended procedure

We will show the procedure extended like this returns `sat` if and only if φ is satisfiable modulo T. To that aim, let us introduce two lemmas.

Lemma 2. *Let $\varphi = \varphi_{elem} \cup \varphi_{struct} \cup \varphi_t \cup \varphi_f$ be a set of literals in separate form. The procedure computes Γ_{struct} and Γ_t such that if:*

- *$\varphi_{struct} \cup \varphi_{elem} \cup \Gamma_{struct}$ is $AFDS_{\Sigma_s}$-satisfiable*
- *$\varphi_t \cup \Gamma_t$ is T_t-satisfiable*

then φ is T-satisfiable.

Proof. Let $\mathcal{A}$ be a $AFDS_{\Sigma_s}$-interpretation satisfying $\varphi_{struct} \cup \varphi_{elem} \cup \Gamma_{struct}$, and let $\mathcal{B}$ be a T_t-interpretation satisfying $\varphi_t \cup \Gamma_t$.
We now define an interpretation $\mathcal{M}$. First, we specify the domains. We let $M_{elem} = A_{elem}$ and $M_t = B$.

M_{struct} will be built by adding one element for each `struct`-variables appearing in the formula, plus the ones built from those with the constructors in $Cons_{\Sigma_S}$ and the elements in M_{elem}, considering the equivalence relations that are implied by φ_{struct}. More formally, consider $L = \bigcup_{n\geq 0} L_n$, where L_i is defined recursively as follows:

$$L_0 = \{l_x : x \in V_{struct}\} \cup \{k : k \text{ is an atomic constructor in } Cons_{\Sigma_s}\}$$
$$L_{n+1} = L_n \cup \{c(x_1, \ldots, x_n, a_1, \ldots, a_m) : x_1, \ldots, x_n, \in L_n, a_1, \ldots, a_m \in V_{elem}\}$$

then M_{struct} is the quotient of L by the weakest equivalence relation implied from the axioms in φ_{struct}. We consider the following interpretation in $\mathcal{M}$:

- for each `t`-variable $u \in V_t$, $\mathcal{M}[u] = \mathcal{B}[u]$
- for each `elem`-variable $a \in V_{elem}$, $\mathcal{M}[a] = \mathcal{A}[a]$
- for each `struct`-variable $x \in V_{struct}$, $\mathcal{M}[x] = [\![x]\!]$ [1].
- the interpretation in $\mathcal{M}$ of the symbols in Σ_s is the same as the one in $\mathcal{A}$, and similarly with the symbols in Σ_t and $\mathcal{B}$.
- The interpretation of the function f is defined recursively over the elements in M_{struct} as follows:
  * If the equivalence class has an element $x \in V_{struct}$, then $\mathcal{M}[f](\mathcal{M}[x]) = \mathcal{B}[f_x]$.
  * If it does not, then the equivalence class must consist of just one constructed element, as $AbsFree_{\Sigma_S}$ ensures all elements built with the constructors are different. Let $t_1, \ldots, t_n$ be terms such that there exist $d_1, \ldots, d_n \in M_{struct}$ such that $\mathcal{M}[f](\mathcal{M}[t_i]) = d_i$, then $\mathcal{M}[f](c(t_1, \ldots, t_n, a_1, \ldots, a_m)) = \mathcal{M}[g^{c,f}](d_1, \ldots, d_n, a_1, \ldots, a_m)$

Now we need to show that $\mathcal{M}$ satisfies $\varphi_t \cup \varphi_{elem} \cup \varphi_{struct} \cup \varphi_f \cup T$. $\varphi_t \cup \varphi_{elem}$ is clearly satisfied, as it was already satisfied by $\mathcal{A}$ and $\mathcal{B}$ and we preserved these interpretations. φ_{struct} must be satisfied as we enforced the right equivalence relation in the universe.

We need to prove $\mathcal{M} \models T$. $\mathcal{M} \models T_s$ and $\mathcal{M} \models T_t$ are trivially deduced from the definitions of the symbols in their signatures. It remains to prove that the function $\mathcal{M}[f]$ satisfies the axioms in Rec_f. This is done by separating constructed terms depending on how we defined the function for them.

- Terms which share an equivalence relation with one or more elements in L_0 that are associated with a variable $x \in V_{struct}$. For this variable, we added to Γ_{struct} one of the following literals :

[1] $[\![x]\!]$ denotes the equivalence class of x

* $f_x = f^k$ if the term is an atomic constructor
* $f_x = g^{c,f}(f(t_1), \ldots, f(t_n))$ if it is built from $f(t_1), \ldots, f(t_n)$.

This is satisfied by $\mathcal{B}$, and since $\mathcal{M}[f(x)] = \mathcal{B}[f_x]$ the axiom is valid.

- Terms which do not share an equivalence class with an element in L_0. In those cases, we defined $\mathcal{M}[f]$ by resorting to the definition deduced from the recursive axioms, so they must hold.

□

Lemma 3. *Let $\varphi = \varphi_{elem} \cup \varphi_{struct} \cup \varphi_t \cup \varphi_f$ be a conjunction of literals in separate form. If φ is T-satisfiable, then the procedure computes Γ_{struct} and Γ_t such that if:*

(a) $\varphi_{struct} \cup \varphi_{elem} \cup \Gamma_{struct}$ is $AFDS_{\Sigma_s}$-satisfiable

(b) $\varphi_t \cup \Gamma_t$ is T_t-satisfiable.

Proof. Let $\mathcal{M}$ be a T-interpretation satisfying φ. We define an arrangement Arr to add in Γ_{struct} as follows:

$$x = y \in Arr \text{ if and only if } \mathcal{M}[x] = \mathcal{M}[y] \text{ for each } x, y \in V_{struct}$$

Clearly, property (a) holds since $\mathcal{M}$ satisfies φ_{struct}, φ_{elem} and $AFDS_{\Sigma_s}$, and by construction also the arrangement.
To check the validity of (b), notice that $\mathcal{M}$ satisfies φ_t. In addition, since the variables f_x do not occur in φ, we can safely modify $\mathcal{M}$ by letting $\mathcal{M}[f_x] = f(\mathcal{M}[x])$ for each $x \in V_{struct}$. The modified $\mathcal{M}$ satisfies T_t and inspecting the replacements in the *decomposition phase* (Section 4.1), it is easy to verify it also satisfies all literals in Γ_t. □

Theorem 7. *The described procedure returns* `sat` *if and only if φ is T-satisfiable.*

Proof. It follows from Lemmas 2 and 3 and considering the guessing of the right arrangement. □

An example with lists

Recall the theory T_{LS} of lists enlarged with a length function we defined in Section 2.2.1, and suppose we want to check T_{LS}-satisfiability of the formula $\varphi : x = cons(a, cons(b, z)) \wedge l(x) + 1 = l(z)$.

1. **Variable Abstraction and Partition** φ will be divided in:

- $\varphi_{list} : y = cons(b, z) \wedge x = cons(a, y)$
- $\varphi_{elem} : true$
- $\varphi_z : c + 1 = d$
- $\varphi_l : l(x) = c \wedge l(z) = d$, where c and d are new constants.

2. **Decomposition** We create the variables l_x, l_y and l_z, and set:

   - Γ_{list}. We need to guess an arrangement between the list variables. Let us choose the one in which they are all different, so we will add to Γ_{list} the set of literals: $\{x \neq y, y \neq z, z \neq x\}$. This is the only relation that is satisfiable together with φ_{list}, so it is the only choice that may lead to satisfiability.
   - Γ_z. After performing the replacements, we will have the set of literals: $\{l_y = l_z + 1, l_x = l_y + 1, l_x = c, l_z = d\}$.

3. **Check**

   - We check $\varphi_{list} \cup \varphi_{elem} \cup \Gamma_{list}$ under T_L. It is satisfiable.
   - We check $\varphi_z \cup \Gamma_z$ under $\mathcal{Z}_0^+$. It is not satisfiable.

   We conclude φ is not satisfiable under T_{LS}.

4.3 Restricting to standard interpretations

The algorithm we presented can lead to some strange situations. Indeed, consider $\varphi : l(x) = 0 \wedge l(y) = 0 \wedge x \neq y$. After phases 1 and 2, we have:

- $\varphi_{list} = \{x \neq y\}$
- $\Gamma_{list} = \{x \neq y\}$
- $\varphi_z = \{l_x = 0, l_y = 0\}$
- $\Gamma_z = \{l_x \geq 0, l_y \geq 0\}$

Clearly $\varphi_{list} \cup \Gamma_{list}$ is satisfiable under T_L, and $\varphi_z \cup \Gamma_z$ is satisfiable under $\mathcal{Z}_0^+$, so we should that conclude φ is satisfiable under T_{LS}. It is, in fact, in our theory of lists, where nothing prevents us from having two different lists of length zero.

We could, however be interested in considering just standard models of lists, meaning only the interpretations we are used to having one zero length list (nil), and all the other ones built from that one by adding elements. This extends to other data structures.

In order to formalize this notion, we need to find the way to ensure that all lists in the universe of a model of our theory will be equal to a list built with the

constructors. As usual, a model can be seen as an homomorphism between the term algebra and another structure of the same algebraic type. We are looking for this homomorphism to be surjective; all elements are related to one in the term algebra. Considering the axioms in $AbsFree$, we know it will also be injective, so we conclude the models we look for are isomorphic to the term algebra for the lists. For the theory of Absolutely Free Data Structures (2.2), we can add the following axiom:

$$(\forall x : struct) \bigvee_c ((\exists \bar{t} : struct, \bar{a} : elem)\ x = c(\bar{t}, \bar{a})) \vee \bigvee_k (x = k)$$

with c a function in $Cons_{\Sigma_s}$, k an atomic constructor in $Cons_{\Sigma_s}$, and $\bar{t}, \bar{a}$ vectors of the length corresponding to the arity.

That is, every element must be built with the constructors. We will call this sentence the *extensionality axiom*, and the theory of Absolutely Free Data Structures with this axiom will be denoted as AFDS^{e}. From now on, we will consider our decision procedures for data structures are for this theory.

Notice that changing the theory makes the procedure incorrect. Under this assumption, fixing the case in the example only requires to ensure we will assign the value 0 to only one integer variable (modulo equivalence).

Let us consider the following formula: $l(x) = 1 \wedge l(y) = 1 \wedge x = y$, and a theory of elements admitting only models whose carrier is a singleton set. The formula should not be satisfiable. Notice that it is the theory of elements which prevents us from building suitable models.

A good starting point is to analyse when a cardinality problem might arise for lists. The problem appears when we do not have enough elements to build different lists. Consider a set of lists which are all distinct pairwise. In this case, if we can assign a different length variable to each of them, the problem is solved, as even with one element we will be able to build them. The difficult case is when the length variables for them cannot be all different, because of constraints in the arithmetic part. In this case, the length must be large enough as to be able to build them all. For example, if we have four different lists of equal length, and only two elements, they must have at least length two to be constructable.

We will analyse in which cases and how we can deal with this new difficulty.

4.3.1 Stable functions

As a first approach, we will consider a case when all the cardinality constraints can be expressed easily in the target theory, so we can add them and see if they are satisfiable. To this aim, let us define a new class of functions.

Definition 17. *A function is* `stable` *if all elements in the image have a preimage of equal cardinality, except maybe for some previously defined elements. When*

considering bridging functions over term-constructed algebras, these constants which may not satisfy the condition are the image of the atomic constructors. This is, if f is function with domain D and image I, over an absolutely free data structure with atomic constructors $k_1, \ldots, k_n$, and $P(a) = \{x \in D | f(x) = a\}$, f is stable if $P(a)$ has the same cardinality for each $a \in D$, except maybe for $f(k_1), \ldots, f(k_n)$.

Extension of the procedure for AFDS^e with stable functions

Let $AFDS_{\Sigma_s}$ be an AFDS^e theory over signature Σ_s with sorts `struct` and `elem`, T_t another theory over a signature Σ_t with sort `t`, and T a theory built adding to $AFDS_{\Sigma_s} \cup T_t$ the axioms to define a surjective and stable bridging function $f : struct \to r$. We will show how to extend the procedure in Section 4.2 to compute Γ_{struct} and Γ_t that make the procedure correct.

The changes take place in the second phase. We start with Γ_{struct} and Γ_t as computed in the procedure for AFDSs, and add some new elements.

On one side, we will need to know which elements are atoms and which are not. To this aim, we guess a new unary relation $K_i(x)$ on V_{struct} for each atom constructor k_i. For each one of these relations, we will add to Γ_{struct} the following set of literals:

$$\{x = k_i | x \in V_{struct}, K_i(x)\} \cup \{x \neq k_i | x \in V_{struct}, \neg K_i(x)\}$$

For Γ_t, the replacements are the same as before but now we also need to provide information about the cardinality of the preimages. This is translated as how many of the added f_x variables can be equal. For example, if we know a function is bijective, we do not want the arithmetic part to assign the same value to f_x and f_y if $x \neq y$, as this would never match with the source part. We have two cases:

- If the function has infinite preimage, there is nothing to do.
- If it has a preimage of constant cardinality k, we do:
    1. Identify sets of lists which are different pairwise. We can do this by finding the equivalence classes in our arrangement of lists Arr and taking one representative from each one, as these lists are for sure different with each other.
    2. Add to the target part a predicate saying these variables can be equal in groups of at most k elements. This can always be done, as it is expressible in first order logic. It will, however, inevitably introduce disjunctions, so we will have to divide the problem into many ones and continue the procedure for each of them.

Correctness of the extended procedure

Let us consider T^e_L and T^e_{LS}, the theories of lists and lists with length but seen as an AFDS^e, that is, with extensionality. We will prove correctness of the extended procedure for the theory T^e_{LS} and discuss a generalization later. Our aim is to prove that the procedure returns `sat` if and only if φ is satisfiable in T^e_{LS}, that is, if it has a standad model. We will, again, introduce two lemmas.

We will use the same notation as in the examples with lists, where:

- Γ_{struct} and φ_{struct} are Γ_{list} and φ_{list}
- Γ_t and φ_t are Γ_z and φ_z
- $AFDS_{\Sigma_s}$ is T^e_L
- T_t is $\mathcal{Z}^+_0$

Lemma 4. *Let $\varphi = \varphi_{elem} \cup \varphi_{list} \cup \varphi_z \cup \varphi_l$ be a conjunction of literals in separate form. If the procedure computes Γ_{list} and Γ_z such that if:*

(a) $\varphi_{list} \cup \varphi_{elem} \cup \Gamma_{list}$ is T^e_L-satisfiable

(b) $\varphi_z \cup \Gamma_z$ is satisfiable in $\mathcal{Z}^+_0$.

then φ is satisfiable in T^e_{LS}.

Proof. Let $\mathcal{A}$ be a T_L-interpretation satisfying $\varphi_L = \varphi_{list} \cup \varphi_{elem} \cup \Gamma_{list}$, which will be isomorphic to the standard interpretation for lists. $\mathcal{Z}^+_0$ stands for the integers. We will build an interpretation $\mathcal{M}$ satisfying φ.

We need to fix the interpretation for l, and a value for each variable in V_{list}, V_{elem}, V_z. For V_{elem} and V_z, these will be the ones in $\mathcal{A}$ and in $\mathcal{Z}^+_0$. We will do something different for V_{list}, but first let us interpret l over the universe of lists, in the standard way.

Now we can interpret the list variables. Let x be a variable in V_{list}, and k be the interpretation for l_x in the arithmetic part. We claim that we can choose an interpretation for x in such a way that $\mathcal{M}[l](x) = k$, without losing the satisfiability in the list part.

First, recall that φ_{list} only has literals of the form:

- $x = cons(a, y)$
- $x = nil$
- $x = y$

- $x \neq y$

Now we will order the values of the length variables in the arithmetic part (the l_x variables) increasingly, and we will prove that we can assign values to each of them in the same order.

- Base case: consider any $x \in V_{list}$ such that $l_x = 0$ in the interpretation. We claim we can always choose $x = nil$ maintaining satisfiability. If we cannot, one of the following happens:
  * The literal $x = cons(a, y)$ appears in φ_{list}. This is not possible, as analysing the translations we can see there would be a literal saying $l_x = l_y + 1$ in Γ_z.
  * The literal $x = y$ appears in φ_{list}, and $l_y \neq 0$. This is also not possible, $l_x = l_y$ should appear in Γ_z.
  * The literal $x \neq y$ appears in φ_{list}, and $l_y = 0$. But recall that we have made a guess for the list variables equal to nil and the ones that are not, and translated this to the arithmetic part. So this case is not possible either.
- Inductive step. Let $k > 0$. By the induction hypothesis, there are lists for all x' such that $l_{x'} < k$. We will show that we can find a list for any x such that $l_x = k$. Let use analyse all the possible problematic cases:
  * The literal $x = cons(a, y)$ appears in φ_{list}, and $l_y \notin [0, k-1]$. This is not possible, as $l_x = l_y + 1$ appears in Γ_z.
  * The literal $x = nil$ appears in φ_{list}. This is not possible, as $l_x = 0$ appears in Γ_z.
  * The literal $x = y$ appears in φ_{list}, and $l_y \neq k$. This is also not possible, $l_x = l_y$ should appear in Γ_z.
  * The literal $x \neq y$ appears in φ_{list}, and $l_y = k$, but we do not have enough elements to build different lists of length k for x and y. In the case of a stable function with a finite number of preimages, this cannot happen, as we already added the restrictions to have only as many equal length variables as possible. In the case of a stable function with infinitely many preimages, we can easily find finitely many distinct lists having the same length.

Then we can conclude that φ can be satisfied in T^e_{LS}. □

Lemma 5. *Let $\varphi = \varphi_{elem} \cup \varphi_{list} \cup \varphi_z \cup \varphi_l$ be a conjunction of literals in separate form. If φ is satisfiable in T^e_{LS}, then the procedure computes Γ_{list} and Γ_z such that:*

(a) $\varphi_{list} \cup \varphi_{elem} \cup \Gamma_{list}$ *is* T_L^e*-satisfiable*

(b) $\varphi_z \cup \Gamma_z$ *is satisfiable in* $\mathcal{Z}_0^+$.

Proof. If φ is satisfied in standard interpretation of T_{LS}^e, then we can build Γ_{list} and Γ_z in such a way that (a) and (b) hold, following the reasoning in Lemma 3. □

We have proved Lemmas 4 and 5 only for the theory of lists for simplicity. This proof can be extended to other Absolutely Free Data Structures in a simple way, considering that φ_{struct} always has literals of the same form. There is one assumption in the proof that the new data structure should also meet. It should be possible to order the values of sort `t` in such a way that it also fixes an order in the preimage of the bridging function. That is, if $t_1 < t_2$, then the data structures in the preimage of t_1 are structurally less complex than the ones in the preimage of t_2. This is a strong requirement, although it is not difficult to find cases like this in data structures. The general proof without this restriction is left for future work.

Theorem 8. *The described procedure returns* `sat` *if and only if the input is* T_{LS}^e*-satisfiable.*

Proof. It follows from Lemmas 4 and 5 and considering the guessing of the right arrangement. □

Applications and examples

Examples of stable functions are not difficult to find. For example, the length for lists over a stably infinite theory of elements is stable, as there are infinitely (countable) many lists for each length, except for zero. This includes, for example, lists of integers. Lists over a theory of elements admitting only one element (chains of an element) makes length also stable, as there is only one list for each length. Any bijective function follows this last case.

4.3.2 Non-stable functions

Although we can find examples of stable functions, this is not always the case, as functions with preimages of different cardinalities are extremely common. Consider for example the `inorder` function mapping trees to lists; depending on the length of the lists, we will have more or less trees that are mapped to it. We can also consider length for lists with a theory of elements admitting only models with a domain of cardinality 2. In this case, there is no easy way to add information to Γ_t to solve the cardinality problems.

In this section we present an algorithm for solving cardinality problems for functions that might not be stable. The idea is to find the model for the target theory that imposes the least cardinality restrictions in the source theory. We present the algorithm for lists, and then discuss when and how it can be extended for other structures.

The algorithm for lists

The following algorithm finds sets of different lists with equal length, which is the problematic case, and it decides if it is possible to build all of them. If it is not possible, it returns `unsat`. This algorithm should be executed after the procedure in Section 4.2 computed Γ_{list} and Γ_z, and add literals with them. We assume that the original procedure returns `sat`, this is, that the computed Γ_{list} and Γ_z make each part of the formula satisfiable under the corresponding theory. If this is not the case, we return `unsat` directly.

1. Identify the set of lists which are all different pairwise. This can be done using Arr (the arrangement of lists guessed in the second step of our procedure), and taking one representative from each equivalence class.

2. Guess an equivalence relation between the length variables for the lists in this set. For each equivalence class of length variables, choose one as a class representative, and call it len_i. LEN will be the set of all len_i, and c_i the cardinality of each class cardinality.

3. If $c_i = 0$ for each i, return `sat`, as there are no different lists with equal length. If not, add to Γ_{list} the formula to check if the theory of elements admits models with at least two different elements. If it is not the case, it will be `unsat`, as there is no way to satisfy two different list with equal length with only one element.

4. Find the largest values the len_i variables can take. To do this, let $P(len_i) : len_i > log_2(c_i)$. If this property holds for a length variable, the lists are long enough to be built from 2 elements. Guess an arrangement on the truth value of $P(len_i)$, for each $len_i \in LEN$.

5. Let $LEN' = \{len_i \in LEN | \neg P(len_i)\}$. If $LEN' = \emptyset$, return `sat`. If not, find the largest possible value for the variables in it, by performing a countdown in the sum. Let $k = \sum_{len_i \in LEN'} log_2(c_i)$.
   Add to Γ_z the constraint $\sum_{len_i \in LEN'} len_i = k$. If it is not satisfiable, decrease k in one, and check again. Repeat the process until a satisfiable constraint is found [2].

[2] This will happen eventually, as we have guessed that this variables have lengths lower than $log_2(c_i)$.

6. When the first satisfiable k is obtained, guess a combination of length values in LEN' that satisfies it.

7. Let d_x be the value for the variable l_x in the model for $\varphi_z \cup \Gamma_z$, for all length variables. Then add to Γ_{list} the formula $x = cons(a_1, cons(a_2, \ldots, cons(a_{d_x}, nil)\ldots))$. If $\varphi_{list} \cup \varphi_{elem} \cup \Gamma_{list}$ is T^e_L-satisfiable, return `sat`. If satisfiability is not obtained for any of these cases, return `unsat`.

Correctness of the procedure with the algorithm

In what follows, we will prove that, for a scenario in which cardinality problems can arise, the procedure in 4.2 complemented with this algorithm is correct.

Lemma 6. *Suppose we want to check φ under T^e_{LS}. If the algorithm returns* `sat`*, then φ is T^e_{LS}-satisfiable.*

Proof. We suppose the procedure for AFDSs in Section 4.2 returned `sat`, because if not the algorithm returns `unsat`. Then, because of Lemma 2, it computed Γ_{list} and Γ_z, such that:

- $\varphi_{list} \cup \varphi_{elem} \cup \Gamma_{list}$ is T^e_L-satisfiable
- $\varphi_z \cup \Gamma_z$ is satisfiable in $\mathcal{Z}_0^+$.

The algorithm returns `sat` in two steps:

- In step 3, if all the sets of different lists with equal lengths are empty. But then there are no cardinality problems, so we can trust the correctness of procedure in sec. 4.2.

- In step 7, after the translation of each length value to the lists, if $\varphi_{list} \cup \varphi_{elem} \cup \Gamma_{list}$ is T^e_L-satisfiable. But then, let $\mathcal{A}$ be the T^e_L-interpretation that satisfies it. $\mathcal{Z}_0^+$ stands for the integers. To build an interpretation $\mathcal{M}$ satisfying φ, we can directly consider A and $\mathcal{Z}_0^+$ as universes, and the interpretations given by $\mathcal{A}$ for V_{list} and V_{elem} and the symbols in Σ_L, and the interpretations in $\mathcal{Z}_0^+$. The function l will be interpreted following exactly its definition.
  Defined this way, $\mathcal{M}$ satisfies $T^e_{LS} \cup \varphi_{list} \cup \varphi_{elem} \cup \varphi_z$. To see it also satisfies φ_l (the original literals involving the function l), notice the length for each variable $x \in V_{list}$ is exactly l_x, as this is enforced by the translation in step 7. Considering that each nameable list is associated to a variable and the translations performed, it is clear that $\mathcal{M} \models \varphi_l$.

□

Lemma 7. *Suppose we want to check φ under T^e_{LS}. If the algorithm returns* `unsat`*, then φ is unsatisfiable under T^e_{LS}.*

Proof. We suppose the procedure for AFDSs in Section 4.2 returned `sat`, because if not the algorithm returns `unsat` directly so the proof is trivial considering the previous lemmas. Then, because of Lemma 2, it computed Γ_{list} and Γ_z, such that:

- $\varphi_{list} \cup \varphi_{elem} \cup \Gamma_{list}$ is T_L-satisfiable
- $\varphi_z \cup \Gamma_z$ is satisfiable in $\mathcal{Z}_0^+$.

We need to prove that, if the additions to Γ_{list} and Γ_z in the algorithm made it return `unsat`, it is because there are not enough elements to build the lists. We assume we made the right guesses. The algorithm returns `unsat` in two cases:

- In step 3, if the theory of elements admits only one element and there are different lists with equal length. This means there are not enough lists, so φ cannot be satisfiable.
- In step 7, if we found a satisfiable value of k that does not leads to any configuration satisfiable in the lists. Suppose this is the case, but there exists a model $\mathcal{M}$ of $T_{LS}^e \cup \varphi$. Consider the quotient of the carrier of $\mathcal{M}$ as outputted from the algorithm, and the representatives which would be in LEN'; those for who P does not hold. Let k' be the sum of the lengths of these lists in $\mathcal{M}$, and let us compare it with k. If $k' \geq k$, the algorithm should have found the right combination of lists that is satisfied by $\mathcal{M}$ and returned `sat`, which it did not, so it is an absurd. On the other hand, it cannot be $k' < k$, as if there are elements to build a combination that sums k', there should be enough elements to build a combination of lists that sums k, and we do not, as we obtained `unsat`. Therefore, there cannot exist a model of $T_{LS}^e \cup \varphi$ because there is no possible value for this k'.

□

Theorem 9. *The algorithm returns* `sat` *if and only if the input is* T_{LS}^e*-satisfiable.*

Proof. It follows directly from Lemmas 6 and 7 and from Theorem 7 for the correctness of the procedure in Section 4.2. □

We have presented the algorithm only for lists. It can be adapted easily to other structures, but the assumptions we made on the theory should be considered. In short, these are:

- The value of the bridging function must depend on the structure and not on the elements.

- Reverse translation must be possible. Given a value of the target sort a, its preimage must be finite and computable modulo the elements inside the structure. This means that we need to be able to obtain the set of the applications of constructors for which the application of the bridging function f returns. We need this in order to perform step 7. If there is more than one element in the preimage, then we will consider each possibility separately.

- Monotonicity in the elements of the preimage. There must be a computable order in the target elements, such that, if $a > b$, then the preimage of a has more elements than the preimage of b.

- There must be a bound. We need to be able to compute some a such that if $f(x) = a$, then there are sufficiently many instantiations of x if the theory of elements admits models with carriers greater than some constant. In the lists, this bound is the logarithm for models with at least two elements.

4.4 Applicability of this approach - A discussion

We have shown how to apply our procedure for some different classes of theories and functions. However, these classes are still very restrictive. For example, we are working with surjective functions defined over absolutely free data structures which are themselves a very particular case of recursive data structure. In addition, we are always considering we have only one bridging function defined.

In this section, we analyse how to extend the procedures we introduced in the previous sections to take into account some of these different possibilities. Some others remain as future work, as we will detail in Chapter 5.

Dealing with non-surjective functions

We have been always working with surjective bridging functions. This has an important advantage: the model in the target theory can associate variables to any element, as they all have a preimage. This is not always the case, and for a non-surjective function we will need to restrict the interpretation to ensure all elements are images of something in the source. We will be able to do this when the subset of the image is expressible in the target theory. That is, when there exists a predicate $p \in P$ such that $p(c)$ iff $(\exists x : source) f(x) = c$. If we have this predicate, all we need to do is to add to Γ_t a set of literals $bound(\varphi) = \{P(f_x) | x \in V_{struct}\}$.

Admitting other functions in the source theory

Until now, our AFDS had no functions besides constructors and selectors. This is not always the case; it may be interesting to apply the same procedure for theories

extended with the definition of some other function. The question is then if the procedure remains correct when adding these extra elements.

We now consider how to adapt the procedures we presented to AFDSs with *defined functions*, which are defined recursively by pattern matching in the constructors.

The first thing we need to take in account is what we can deduce in the target theory from these functions. We need to be able to express the relation of these functions with the bridging function as axioms in Γ_t. To this aim, if we want to include the function $g : struct \to struct$, there must exist a function h in Σ_t which "mocks" the implications of g over the bridging theory f. This is, if we have the literal $x = g(y_1, \ldots, y_n)$ in φ_{struct}, we will translate it in Γ_t as $f_x = h(f_{y_1}, \ldots, f_{y_n})$. For example,

- For the function `reverse` in lists with length, the length of a list must be equal to the length of its reverse. So if $x = reverse(y)$, then $f_x = f_y$.
- For concatenation in lists with length, it should be $x = y{+}{+}z \implies f_x = f_y + f_z$

Once we have identified this mocking function (if it exists), we need to translate all literals containing the defined function to Γ_t, and then perform the procedure as usual.

Modularity

One of the main reasons to use this reducing-to-disjoint approach is that it can be applied modularly. If we have several theories with bridging functions between them, we would like to be able to apply the procedure repeatedly and obtain many disjoint theories. In this section we study different situations under which the decision procedure can be modularized.

Several source theories to one target theory The approach works modularly in this case; we can apply it for each source theory once at a time, obtaining several disjoint problems.

This is a common scenario. For example, we can imagine having different data structures, defined by theories over disjoint signatures, with bridging "measure" functions to arithmetic (lists with length, trees with height, multisets with number of occurrences). In that case, the functions can be dealed with one by one.

For example, suppose we have T_{LS}, the theory of lists with length, and T_{TH}, the trees with height, together. Then we can apply the procedure and obtain a disjoint problem between T_L (the theory of lists) and T_{TH}. By applying it again, we can also reduce this last one to the problem between the theory of trees and arithmetic, solving everything separately.

One source theory to several target theories This case is not as simple as the previous one, because of the implicit information that each of the functions can add to the target theory. For example, consider trees with number of nodes (to arithmetic) and inorder (to lists). These functions have different target theories, but are related, as a tree that has a certain inorder list has its number of nodes determined. This case can be solved if we have a "mocking" function between the target theories, making explicit these relations. In the example we have just seen, we should use the length for lists, and add in Γ_{list} the information obtained from the number of nodes and vice versa in Γ_z. Notice that this would make the target theory a disjoint one (lists with length), but then we can apply the procedure modularly.

One source to one target, several functions In this case it is also necessary to consider the relations between the functions. If they are completely disjoint and do not impose restrictions to each other, we can apply the procedure in parallel, creating variables for each of the functions and translating the axioms. If they are not, we need to be able to express the conditions in the target Γ. For example, for trees with height and number of nodes, we should encode the maximum and minimum number of nodes for a tree of a certain height in the arithmetic.

5 Conclusions and future work

We studied two different approaches to the non-disjoint combination of theories and provided procedures to solve the satisfiability problem on combinations of theories for certain cases. In the first one, we introduced a notion of $\mathcal{P}$-gentleness which is well-suited for combining theories sharing (besides constants and the equality) only unary predicates in a set $\mathcal{P}$. We proved that the Löwenheim theories are $\mathcal{P}$-gentle; BSR theories also are.

Our combination method is limited to shared unary predicates, and the extension for other arities seems complicated for theoretical reasons. A possible direction for research is to identify particular predicates or classes of predicates for which the problem can be solved, and analyse how the topology of the predicates impacts in the complexity of the problem.

Also in future works, the reduction approach (BSR theories can be simplified to a subset of Löwenheim) may be useful as a simplification procedure for sets of formulas that can be seen as non-disjoint (sharing unary predicates only) combinations of BSR or Löwenheim theories and an arbitrary first-order theory: this would not provide a decision procedure, but refutational completeness can be preserved.

The results here are certainly too combinatorially expensive to be directly applicable. However, this work paves the theoretical grounds for mandatory further works that would make such combinations practical. There are important incentives since the BSR and Löwenheim fragments are quite expressive: for instance, it is possible to extend the language of SMT solvers with sets and cardinalities. Many formal methods are based on logic languages with sets. Expressive decision procedures (even if they are not efficient) including e.g. sets and cardinalities will help proving the often small but many verification conditions stemming from these applications.

In Chapter 4, we presented another approach, in which we reduce the problem of deciding satisfiability over two disjoint theories connected with a bridging function to the satisfiability problem of each of the theories. Our results are for the theory of Absolutely Free Data Structures that we introduced in the Preliminaries section. A clear direction to explore is the possibility of extending these ideas to other data structures, in which we do not assume that equal structures can be constructed in only one way. This is an interesting problem as many well known data structures

do not fit in the Absolutely Free Data Structures framework, like for example sets. Ideas of how to delete functions over sets were given in [51] and [47]; we leave for future work to develop a framework for more general theories. Ideas of how to go beyond Absolutely Free Data Structures are given in [44]. This work needs to be extended, but the framework Sofronie-Stokkermans proposes (algebras built with constructors that satisfy only commutativity, associativity, idempotence and nilpotence) could be a good initial point from where we could study more general data structures, relating them with the work in this thesis.

Also as future work, it would be interesting to consider bridging functions over Absolutely Free Data Structures defined in other ways than by simple recursion. For example, we could consider conditional definitions like:

$$Rec_f = \begin{cases} f(k) = k_f \\ \\ f(c(x_1, \ldots, x_n, y_1, \ldots, y_m)) = \begin{cases} g_1^{c,f}(f(x_1), \ldots, f(x_n), y_1, \ldots, y_m) \\ \qquad \text{if } p_1(x_1, \ldots, x_n, y_1, \ldots, y_m) \\ \\ g_k^{c,f}(f(x_1), \ldots, f(x_n), y_1, \ldots, y_m) \\ \qquad \text{if } p_k(x_1, \ldots, x_n, y_1, \ldots, y_m) \end{cases} \end{cases}$$

where $p_1, \ldots, p_k$ are predicates in Σ_S. This would introduce a new difficulty for translating the functions.

We gave one general procedure for Absolutely Free Data Structures, and proved it works in the case when we are not restricted to consider only standard models. We saw that this latter case introduces a new difficulty, as cardinality problems can arise when we do not have enough elements to build the lists, and our procedure did not provide a way to express that in the target theory. In fact, these constraints are difficult to translate, as in the general case we do not know the cardinality of our models. We studied solutions for two cases; in one we can express the restrictions directly because the functions have nice cardinality properties, and in the second one we need to perform an algorithm to check if the constraints are satisfiable. The proof for the first case and the algorithm in the second one are presented for the particular theory of lists with length and we explain for which theories generalization is easy. This generalization should be formalized. It would also be interesting to go beyond these restrictions and find a framework to solve cardinality issues in a general way.

In the last section of this chapter we discussed some extensions of the procedure to relax conditions we imposed before. These ideas need to be formalized and extended, since they are crucial in order to be able to actually apply these procedures in an SMT solver. It is specially necessary to study in depth how to accept other functions defined over the data structure, as this is what makes the source theory interesting. We should also provide a formal framework to approach modularity, which is one of the most interesting features of this kind of procedures, and of special relevance for software specification.

In order to make these procedures applicable it would also be necessary to improve their efficiency, as they are currently too combinatorially expensive; ways of making the guessings smarter should definitely be explored. An interesting idea towards more efficient algorithms is to consider also a deductive approach, in which we first extract information that is implicit in φ to decrease the number of cases to test. This is specially suitable for applications in SMT, as they already perform this kind of methods, so the machinery is available. The implementation of these procedures in an SMT solver and the necessary study of how to integrate them is definitely interesting future work.

Finally, a long-term objective would be to analyse the relations between our two approaches; the one extending Nelson-Oppen procedure and the one for bridging functions. Although they seem to refer to different situations, there are similarities, particularly in that they both deal with cardinality constraints. Finding a unified theoretical framework to express these problems and a way to solve them uniformly would be a challenging but definitely interesting task.

Bibliography

[1] W. Ackermann. *Solvable Cases of the Decision Problem.* North-Holland Publishing Company, 1954.

[2] C. Areces and P. Fontaine. Combining theories: The Ackerman and Guarded fragments. volume 6989, pages 40–54. Springer, 2011.

[3] A. Armando, M. P. Bonacina, S. Ranise, and S. Schulz. New results on rewrite-based satisfiability procedures. *ACM Trans. Comput. Log.*, 10(1), 2009.

[4] A. Armando, S. Ranise, and M. Rusinowitch. A rewriting approach to satisfiability procedures. *Inf. Comput.*, 183(2):140–164, 2003.

[5] M. Baaz, U. Egly, and A. Leitsch. Normal form transformations. In J. A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 5, pages 273–333. Elsevier Science B.V., 2001.

[6] C. Barrett, R. Sebastiani, S. A. Seshia, and C. Tinelli. Satisfiability modulo theories. In A. Biere, M. J. H. Heule, H. van Maaren, and T. Walsh, editors, *Handbook of Satisfiability*, volume 185, chapter 26, pages 825–885. IOS Press, Feb. 2009.

[7] L. D. Bjørner. Satisfiability modulo theories: introduction and applications. *Communications of the ACM*, 54(9):69–77, 2011.

[8] M. P. Bonacina, S. Ghilardi, E. Nicolini, S. Ranise, and D. Zucchelli. Decidability and undecidability results for nelson-oppen and rewrite-based decision procedures. In U. Furbach and N. Shankar, editors, *Proceedings of the 3rd International Joint Conference on Auto mated Reasoning (IJCAR 2006)*, volume 4130 of *Lecture Notes in Computer Science*, page 513–527. Springer.

[9] E. Börger, E. Grädel, and Y. Gurevich. *The Classical Decision Problem.* Perspectives in Mathematical Logic. Springer-Verlag, Berlin, 1997.

[10] C. Chang and H. Keisler. *Model Theory.* Studies in Logic and the Foundations of Mathematics. Elsevier Science, 1990.

[11] P. Chocrón, P. Fontaine, and C. Ringeissen. A gentle non-disjoint combination of satisfiability procedures.

[12] A. Church. A note on the Entscheidungsproblem. *J. Symb. Log.*, 1(1):40–41, 1936.

[13] S. A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, STOC '71, pages 151–158, New York, NY, USA, 1971. ACM.

[14] G. Dowek, A. Felty, H. Herbelin, G. Huet, C. Murthy, C. Parent, C. Paulin-mohring, and B. Werner. The coq proof assistant user's guide (version 5.8). Technical Report 154, INRIA, Rocquencourt, France, 1993.

[15] B. Dreben and W. D. Goldfarb. *The Decision Problem: Solvable Classes of Quantificational Formulas.* Addison-Wesley, Reading, Massachusetts, 1979.

[16] H. B. Enderton. *A Mathematical Introduction to Logic.* Academic Press, Inc., Orlando, Florida, 1972.

[17] J.-C. Filliâtre, S. Owre, H. Rue*B, and N. Shankar. Ics: Integrated canonizer and solver? In G. Berry, H. Comon, and A. Finkel, editors, *Computer Aided Verification*, volume 2102 of *Lecture Notes in Computer Science*, pages 246–249. Springer Berlin Heidelberg, 2001.

[18] P. Fontaine. Combinations of theories for decidable fragments of first-order logic. In *Frontiers of Combining Systems, 7th International Symposium, FroCoS 2009, Trento, Italy, September 16-18, 2009. Proceedings*, volume 5749 of *Lecture Notes in Computer Science*, pages 263–278. Springer, 2009.

[19] J. Franco and J. Martin. A history of satisfiability. In A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors, *Handbook of Satisfiability.* IOS Press, 2009.

[20] J. L. K. G. Kreisel. *Elements of Mathematical Logic. Studies in Logic and the Foundations of Mathematics.* North-Holland Publishing Company, 1967.

[21] H. Ganzinger, C. Meyer, and M. Veanes. The two-variable guarded fragment with transitive relations. pages 24–34. IEEE Computer Society, 1999.

[22] H. Ganzinger and H. D. Nivelle. A superposition decision procedure for the guarded fragment with equality. In *In Proc. LICS'99*, pages 295–303. IEEE Computer Society Press, 1999.

[23] S. Ghilardi. Model-theoretic methods in combined constraint satisfiability. *Journal of Automated Reasoning*, 33(3-4):221–249, 2004.

[24] E. Goldberg and Y. Novikov. Berkmin: A fast and robust sat-solver. *Discrete Appl. Math.*, 155(12):1549–1561, June 2007.

[25] Y. Gurevich and S. Shelah. Spectra of monadic second-order formulas with one unary function. In *LICS '03: Proceedings of the 18th Annual IEEE Symposium on Logic in Computer Science*, pages 291–300, Washington, DC, USA, 2003. IEEE Computer Society.

[26] D. Jackson. Alloy: a lightweight object modelling notation. *ACM Transactions on Software Engineering and Methodology*, 11(2):256–290, 2002.

[27] L. Löwenheim. Über möglichkeiten im relativkalkül. *Mathematische Annalen*, 76(4):447–470, 1915.

[28] Z. Manna and C. G. Zarba. Combining decision procedures. In *Formal Methods at the Crossroads. From Panacea to Foundational Support, 10th Anniversary Colloquium of UNU/IIST, Revised Papers*, volume 2757 of *Lecture Notes in Computer Science*, pages 381–422. Springer, 2003.

[29] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient sat-solver. In *Proceedings of the 38th Annual Design Automation Conference*, DAC '01, pages 530–535, New York, NY, USA, 2001. ACM.

[30] G. Nelson and D. C. Oppen. Simplification by cooperating decision procedures. *ACM Transactions on Programming Languages and Systems*, 2(1):245–257, 1979.

[31] E. Nicolini, C. Ringeissen, and M. Rusinowitch. Combinable extensions of abelian groups. In R. A. Schmidt, editor, *Automated Deduction - CADE-22, 22nd International Conference on Automated Deduction, Montreal, Canada, August 2-7, 2009. Proceedings*, volume 5663 of *Lecture Notes in Computer Science*, pages 51–66. Springer, 2009.

[32] E. Nicolini, C. Ringeissen, and M. Rusinowitch. Combining satisfiability procedures for unions of theories with a shared counting operator. *Fundam. Inform.*, 105(1-2):163–187, 2010.

[33] T. Nipkow, L. Paulson, and M. Eenzel. *Isabelle/HOL – A proof assistant for higher-order logic*, volume 2283 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, Germany, 2002.

[34] D. C. Oppen. Reasoning about recursively defined data structures. *Journal of the Association for Computing Machinery*, 3(27):403–411, 1980.

[35] S. Owre, S. Rajan, J. Rushby, N. Shankar, and M. Srivas. PVS: Combining specification, proof checking, and model checking. pages 411–414.

[36] D. Parnas. On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, 15(12):1053–1058, 1972. See also [**?**].

[37] D. L. Parnas. Designing software for ease of extension and contraction. In *Proceedings of the 3rd International Conference on Software Engineering*, ICSE '78, pages 264–277, Piscataway, NJ, USA, 1978. IEEE Press.

[38] M. Presburger. Uber die vollst ndigkeit eines gewissen systems der arithmetik a ganzer zahlen, in welchen die addition als einzige operation hervortritt. *Comptes Rendus du Premier Congrès des Mathèmaticienes des Pays Slaves*, pages 92–101, 1929.

[39] F. P. Ramsey. On a Problem of Formal Logic. *Proceedings of the London Mathematical Society*, 30:264–286, 1930.

[40] S. Ranise, C. Ringeissen, and C. G. Zarba. Combining data structures with nonstably infinite theories using many-sorted logic. In B. Gramlich, editor, *Frontiers of Combining Systems, 5th International Workshop, FroCoS 2005, Vienna, Austria, September 19-21, 2005, Proceedings*, volume 3717 of *Lecture Notes in Computer Science*, pages 48–64. Springer, 2005.

[41] C. Ringeissen and V. Senni. Modular termination and combinability for superposition modulo counter arithmetic. volume 6989 of *Lecture Notes in Computer Science*, pages 211–226. Springer, 2011.

[42] C. Ringeissen and C. Tinelli. Unions of non-disjoint theories and combinations of satisfiability procedures. *Theoretical Computer Science*, 290(1):291–353, Jan. 2003.

[43] R. E. Shostak. Deciding combination of theories. *Journal of the Association for Computing Machinery*, 1(31):1–12, 1984.

[44] V. Sofronie-Stokkermans. Locality results for certain extensions of theories with bridging functions. In R. A. Schmidt, editor, *Automated Deduction - CADE-22, Proceedings*, volume 5663 of *Lecture Notes in Computer Science*, pages 67–83. Springer, 2009.

[45] V. Sofronie-Stokkermans. On combinations of local theory extensions. In *Programming Logics - Essays in Memory of Harald Ganzinger*, volume 7797 of *Lecture Notes in Computer Science*, pages 392–413. Springer, 2013.

[46] P. Suter, M. Dotta, and V. Kuncak. Decision procedures for algebraic data types with abstractions. *SIGPLAN Not.*, 45(1):199–210, Jan. 2010.

[47] P. Suter, R. Steiger, and V. Kuncak. Sets with cardinality constraints in satisfiability modulo theories. In R. Jhala and D. Schmidt, editors, *Verification, Model Checking, and Abstract Interpretation*, volume 6538 of *Lecture Notes in Computer Science*, pages 403–418. Springer Berlin Heidelberg, 2011.

[48] A. Tarski. A decision method for elementary algebra and geometry. *University of California Press*, pages 92–101, 1951.

[49] C. Tinelli and C. G. Zarba. Combining nonstably infinite theories. *Journal of Automated Reasoning*, 34(3):209–238, Apr. 2005.

[50] C. Zarba. Combining multisets with integers. In A. Voronkov, editor, *Automated Deduction—CADE-18*, volume 2392 of *Lecture Notes in Computer Science*, pages 363–376. Springer Berlin Heidelberg, 2002.

[51] C. Zarba. Combining sets with integers. In A. Armando, editor, *Frontiers of Combining Systems*, volume 2309 of *Lecture Notes in Computer Science*, pages 103–116. Springer Berlin Heidelberg, 2002.

[52] C. G. Zarba. Combining lists with integers. In *International Joint Conference on Automated Reasoning (Short Papers), Technical Report DII 11/01*, pages 170–179. University of, 2001.

[53] C. G. Zarba. Combining sets with cardinals. *J. Autom. Reasoning*, 34(1):1–29, 2005.

[54] T. Zhang, H. B. Sipma, and Z. Manna. Decision procedures for term algebras with integer constraints. *Inf. Comput.*, (10):1526–1574.