SMT Beyond DPLL(T): A New Approach to Theory Solvers and Theory Combination

by

Dejan Jovanović

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
Department of Computer Science
Courant Institute of Mathematical Sciences
New York University
September 2012

Clark W. Barrett

Abstract

Satisifiability modulo theories (SMT) is the problem of deciding whether a given logical formula can be satisifed with respect to a combination of background theories. The past few decades have seen many significant developments in the field, including fast Boolean satisfiability solvers (SAT), efficient decision procedures for a growing number of expressive theories, and frameworks for modular combination of decision procedures. All these improvements, with addition of robust SMT solver implementations, culminated with the acceptance of SMT as a standard tool in the fields of automated reasoning and computer added verification. In this thesis we develop new decision procedures for the theory of linear integer arithmetic and the theory of non-linear real arithmetic, and develop a new general framework for combination of decision procedures. The new decision procedures integrate theory specific reasoning and the Boolean search to provide a more powerful and efficient procedures, and allow a more expressive language for explaining problematic states. The new framework for combination of decision procedures overcomes the complexity limitations and restrictions on the theories imposed by the standard Nelson-Oppen approach.

Acknowledgments

I am in debt to many wonderful people without whose support, help, and advice this thesis would never come to be. I have to start by thanking my parents, Sreten and Smilja, and my brother Vladan, for their unwavering love and support. The guidance of my academic advisers Predrag Janičić, Alexander Leitsch, and especially my PhD adviser Clark Barrett, was invaluable in shaping me as a researcher. I have also benefited greatly from the experiences obtained at research internships during my PhD, and I would like to thank Pankaj Chauhan, Lucas Bordeaux, and Leonardo de Moura for their mentorship. I would also like to thank the members of my thesis committee, Thomas Wies, Benjamin Golberg and Morgan Deters, for their encouragement and helpful advice. Finally, I also have to thank the many friends that have tolerated me in New York City, with distinguished honors going to my roommates Deep Ganguli and Joel Alwen, and my 404 office mates Christopher Conway and Mina Joeng.

Table of Contents

List of Figures

List of Tables

List of Appendices

Introduction

At the International Congress of Mathematics in 1928, David Hilbert introduced the *Entscheidungsproblem* (German for "decision problem"). This decision problem that Hilbert was referring to asks for a precise list of instructions, or in today's terms an algorithm, such that, given a mathematical statement, one can follow these instructions and answer "yes" or "no" depending on the universal unquestionable truth of the statement. The idea of such a mechanical mathematician has been floating about since the times of Leibniz, and Hilbert himself held a strong opinion that such an algorithm should naturally exist. But, a few years later, he was proved wrong with Alonzo Church and Alan Turing independently showing that there can be no algorithm to solve the problem in its full generality. These results shook the world of mathematics but, at the same time, spearheaded a more focused research effort into decision problems where such limits might not apply. Indeed, if one restricts attention to more limited logical fragments, such algorithms (*decision procedures*) do exist. One of the first and most impressive examples of such a *decidable* fragment is the first-order theory of real numbers (real closed fields). In the 1930s, Alfred Tarski [90, 91] famously presented an elegant procedure for deciding any statement in this theory. Since then, there has been an ongoing quest to see how much and which parts of mathematics can be decided automatically, giving rise to the field of *automated theorem proving*.

Tarski's procedure was quite inefficient in terms of computational complexity, and practically not very useful. Ultimately, it was a foundational result that identified an important decidable fragment of mathematics and provided more insight into real closed fields and their properties. But as computing power became a more accessible commodity, some of these procedures made their way into the first available computers, and the field started to take shape with efficiency in mind. The first generation of decision procedures, with actual implementations, started with Gilbert's procedure [49], which marked the field with a concrete running implementation. Procedures that followed were traditionally rooted in Herbrand's results on quantified first-order logic (e.g. the original Davis-Putnam procedure [32]) and the seminal results of Alan Robinson on the resolution principle [81]. Since quantified first-order logic is semi-decidable, these *resolution-based* procedures could in theory show any theorem to be true, and were therefore called *theorem*

provers. Although, in practice, they were limited to deciding only small academic examples, they laid the foundations for most of the modern resolution-based theorem provers.

These first attempts at automation of reasoning were generally aimed at uninterpreted quantified first-order logic. Since quantifiers are a part of the logic, when a particular interpretation (e.g. arithmetic) is needed, one could append to the problem the axioms of interest, try the general procedure, and hope for the best. This was soon shown to be impractical as many trivial problems over interpreted theories are quickly out of reach for this approach. Moreover, this excludes theories that do not have a finite axiomatization, such as the integers. Another possibility is to try and develop procedures with ingrained knowledge about the theory, as is the case with Tarski's procedure, and this is the general topic of this thesis.

One of the simplest interpreted logics is the propositional Boolean logic, where the domain includes two distinct Boolean constants true and false, and operations including conjunction ($\wedge$), disjunction ($\vee$) and negation ($\neg$), which are interpreted as usual. In this logic, proving a statement ϕ over variables $p_1, \ldots, p_n$ (the propositions) to be true amounts to showing that no assignment of variables to Boolean constants can falsify ϕ. In other words, it amounts to checking if there is an assignment of the variables such that $\neg\phi$ evaluates to true. This is what we now know as the classic *propositional satisfiability* (SAT) problem. Although the SAT problem is a canonical NP-complete problem [28], and is therefore believed to be of exponential complexity, there has been an incredible amount of progress in algorithms (SAT solvers) that attack it. In fact, it has become a matter of routine for such solvers to successfully solve problems with hundreds of thousands of variables. Based on the success of SAT, particularly in many applications that are found in areas of formal verification of hardware and software, it became evident that the concept of satisfiability checking, i.e. finding solutions or showing that one doesn't exist, can be used to model many practically important problems where formal reasoning is desirable. For example, showing that two circuits ϕ_1 and ϕ_2 are equivalent, a classic problem in hardware verification, amounts to showing that $\neg(\phi_1 \Leftrightarrow \phi_2)$ is not satisfiable. It can be argued that SAT solvers made it possible for automated theorem proving to make the leap from a purely academic pursuit to an endeavor of industrial importance.

It is important to note that the algorithms underlying modern SAT solvers are conceptually different from the symbolic reasoning employed by most theorem provers. First attempts at

solving SAT were also based on symbolic reasoning [32], relying on the resolution principle at the Boolean level, but suffered from similar drawbacks as those of the general theorem provers. Later algorithms adopted a more natural constructive approach, starting from [31]. They try to construct a satisfying interpretation (a model) of the problem at hand explicitly, by assigning the variables to particular values. But, when the process of model construction fails, they use symbolic reasoning in form of Boolean resolution to learn from the failure, backtrack appropriately and continue from there, trying some other values. It is a basic example of a decision procedure employing a search for a model complemented with a *model-based* symbolic conflict analysis.

Looking beyond propositional logic, the next natural step is to extend the satisfiability problem to allow more expressive languages and semantics, with variables ranging not just over Booleans but over domains such as, for example, the real numbers, integers, and arrays, and include the usual operations over these domains. More precisely, we are talking about the quantifier-free fragment of first-order logic, where interpretations can be restricted to some given background theories. The satisfiability problem in this setting is called *satisfiability modulo theories* [6] (SMT). As an example, consider the formula

$$(x - y = 0) \wedge (f(x) < f(y)) \ ,$$

with the variables x and y ranging over integers, and the symbol f an uninterpreted function over the integers. Checking whether we can assign x and y to integer values and interpret the function f accordingly, in order to make the formula true, is an instance of the SMT problem.[1] This particular instance uses as the background theories the theory of linear integer arithmetic and the theory of uninterpreted functions.

In order to develop a solver for SMT problems, for each particular background theory a different decision procedure must be developed. But, as was the case with SAT solvers, the shift to integration of domain knowledge into the decision procedures provides a more natural and efficient approach to reasoning, and many efficient decision procedures are readily available. Starting from the first solvers such as SVC [4] and ICS [43], SMT provided such a level of automation that nowadays an SMT solver is an essential tool in many industrial labs dealing with formal reasoning. The field of SMT traditionally focuses on the studies of decision procedures for the individual

[1] The answer is no, since to satisfy it we must have that $x = y$ but that implies that $f(x) = f(y)$.

theories [61, 13] and then separately with ways of combining these decision procedures when deciding formulas that include multiple background theories [70, 84]. This thesis will present contributions in both of these areas.

In Chapter 1 and Chapter 2 we will present new decision procedures for two quantifier-free fragments of arithmetic: linear integer arithmetic and non-linear real arithmetic. Both theories have been studied extensively, the quantifier-free fragments are decidable, and there are available decision procedures (e.g. [41, 51, 26, 99, 75]). The procedures we describe, in addition to improvements in practical efficiency, are also unique in using the SAT-style search for a model complemented with model-based conflict analysis. In comparison, most decision procedures in the SMT context are developed to decide conjunctions of assertions. This is sufficient since the Boolean structure of the problem can always be relegated to an efficient SAT solver, in a framework commonly commonly called DPLL(T) [72, 62]. Although this allows reaping the benefits of the constant improvements in SAT solving, being detached from the Boolean search is a restriction that can reflect badly on the proof-theoretic properties of the system – there are seemingly simple problems that can not be solved in sub-exponential time. The procedures we describe, on the other hand, allow for a tight integration of theory-specific reasoning and the Boolean search that overcomes these limitations.

In Chapter 3 we then attack the theory combination problem. Given decision procedures for two distinct theories, one can resort to the classic combination method of Nelson and Oppen [70] to devise a procedure that can decide the combined theory. The Nelson-Oppen method is a remarkable result that in a way started the field of SMT, but it also has some shortcomings. First, it fails to address combinations of certain classes of theories (e.g. the theory of bit-vectors) and, as originally stated, does not scale very well due to inherent complexity issues. In Chapter 3 we develop a new framework for combination of theory solvers. The new combination method is proved correct even for theories that do not satisfy the properties required by the original Nelson-Oppen approach and in addition provides a way to overcome some of the inherent complexity that any combination framework is confronted with.

1
Linear Integer Arithmetic

One of the most impressive success stories of computer science in industrial applications was the advent of linear programming algorithms. Linear programming (LP) became feasible with the introduction of Dantzig's simplex algorithm. Although the original simplex algorithm targets problems over the rational numbers, in 1958 Gomory [50] introduced an elegant extension to the integer case (ILP). He noticed that, whenever the simplex algorithm encounters a non-integer solution, one can eliminate this solution by deriving a plane, that is implied by the original problem, but does not satisfy the current assignment. Incrementally adding these *cutting planes*, until an integer solution is found, yields an algorithm for solving linear systems over the integers. Cutting planes were immediately identified as a powerful general tool and have since been studied thoroughly both as an abstract proof system [24], and as a practical preprocessing step for hard structured problems. For such problems, one can exploit the structure by adding cuts tailored to the problem, such as the clique cuts, or the covering cuts [100], and these cuts can reduce the search space dramatically.

The main idea behind the algorithm of Gomory, i.e to combine a model searching procedure with a conflict resolution procedure – a procedure that can derive new facts in order to eliminate a conflicting candidate solution – is in fact quite general. Somewhat later, for example, in the field of Boolean satisfiability (SAT), there was a similar development with equally impressive end results. Algorithms and solvers for the SAT problem, although dealing with a canonical NP-complete problem, have seen a steady improvement over the years, culminating in thrilling advances in the last decade. Contrarary to what one would expect of an NP-complete problem, it has become a matter of routine to use a SAT solver on problems with millions of variables and constraints. Of course, it would be naive to attribute one single reason to this success, for there are many ingredients that contribute to efficiency of modern SAT solvers. But, one of the most conceptually appealing techniques that these SAT solvers use is a combination of two orthogonal views on how to go about solving a satisfaction problem. One is a backtracking search for a satisfying assignment, as described in the original DPLL [31] algorithm. The other is a search for a proof that there is no solution, in this case a refutation using Boolean resolution, as described

in the DP algorithm [32].

In order to combine these two approaches Silva and Sakallah [85] noticed that, although completely different, they can be used to complement each other in a surprisingly natural manner. If the search for a satisfying assignments encounters a conflicting state, i.e. one in which some clause is falsified by the current candidate assignment, one can use resolution to derive a clause, commonly called *an explanation*, that succinctly describes the conflict. As is the case with Gomory's cutting planes, this explanation clause eliminates the current assignment, so the search is forced to backtrack and consider a different one. Moreover, since this explanation is a valid deduction, it can be kept to ensure that the conflict does not occur again. These explanations can often eliminate a substantial part of the subsequent search tree. The important insight here is that the application of resolution is limited to the cases where it is needed by the search, or in other words the *search is guiding the resolution*. As is usually the case with search algorithms that attack hard problems, the search process can be greatly improved by applying heuristics at the appropriate decision points. In the case of the SAT problem, the decision of which variable to try and assign next is one of the crucial ones. With the above idea of search complemented with conflict resolution in mind, Moskewicz et al. [69] introduced the VSIDS heuristic. This heuristic prefers the variables that were involved in the resolution of recent conflicts, effectively adding the feedback in the other direction, i.e. *the resolution is guiding the search*. This approach to solving SAT problems is commonly called conflict-directed clause learning (CDCL), and is employed by most modern SAT solvers.

Unsurprisingly, the success of SAT solvers has encouraged their adoption in attacking problems from other domains, including some that were traditionally handled by the ILP solvers [8]. These ILP problems are the ones where variables are restricted to the $\{0, 1\}$ domain, and are commonly referred to as *pseudo-Boolean* (PB) problems. Although these problems still operate over Boolean variables, conflict resolution is problematic even at this level [23]. The key problem is to find an analogue conflict resolution principle for integer inequalities, since the Fourier-Moztkin resolution is imprecise for the integers, and the deduced inequalities are often *too weak* to resolve a conflict. For example, consider the inequalities

$$3x_3 + 2x_2 + x_1 \geq 4 \ , \qquad\qquad -3x_3 + x_2 + 2x_1 \geq 1 \ .$$

If we are in a state with an assignment such that $x_1 \mapsto 1$ and $x_2 \mapsto 1$ the left inequality implies that $x_3 \geq 1$, and the right inequality implies that $x_3 \leq 0$. In other words, it is not possible to extend the partial model to x_3 and we are therefore in a conflicting state. We can try to apply a Fourier-Motzkin resolution step to above inequalities in order to explain the conflict. If we do so, we eliminate the variable x_3 and obtain the inequality $3x_2 + 3x_1 \geq 5$, which in the integer domain is equivalent to $x_2 + x_1 \geq 2$. This inequality is not strong enough to explain the conflict, since it is satisfied in the current state.

In this chapter we will resolve this issue and provide an analogue of Boolean resolution not just for PB problems, but for the more general ILP case. We achieve this by introducing a technique for computing *tightly-propagating inequalities*. These inequalities are used to justify every propagation performed by our procedure, and have the property that Fourier-Moztkin resolution is precise for them. Tightly-propagating inequalities guarantee that our conflict resolution can succinctly explain each conflict.

Using the new conflict resolution procedure we then develop a CDCL-like procedure for solving arbitrary ILP problems. The procedure is inspired by recent algorithms for solving linear real arithmetic [67, 60, 30], and has all the important theoretical and practical ingredients that have made CDCL based SAT solvers so successful. As in CDCL, the core of the new procedure consists of a search for an integer model that is complemented with generation of resolvents that explain the conflicts. The search process is aided with simple and efficient propagation rules that enable reduction of the search space and early detection of conflicts. The resolvents that are learned during analysis of conflicts can enable non-chronological backtracking. Additionally, all resolvents generated during the search are valid, i.e. implied by the input formula, and not conditioned by any decisions. Consequently, the resolvents can be removed when not deemed useful, allowing for flexible memory management by keeping the constraint database limited in size. Finally, all decisions (case-splits) during the search are not based on a fixed variable order, thus enabling dynamic reordering heuristics.

Existing ILP solvers can roughly be divided into two main categories: saturation solvers, and cutting-planes solvers. Saturation solvers are based on quantifier elimination procedures such as Cooper's algorithm [29] and the Omega Test [77, 10]. These solvers are essentially searching for a proof, but have the same drawbacks as the DP procedure. On the other hand, the cutting-planes

solvers are model search procedures, complemented with derivation of cutting planes. The main difference with our procedure is that these solvers search for a model in the rational numbers, and use the cutting-planes to eliminate non-integer solutions. Moreover, although it is a well-known fact that for every unsatisfiable ILP problem there exists a cutting-plane proof, to the best of our knowledge, there is no effective way to find this proof. Most systems based on cutting-planes thus rely on heuristics, and termination is not guaranteed. In most cases, the problem with termination is *hidden* behind the assumption that all the problem variables are bounded, which is common in traditional practical applications. Even in theory, this is not an invalid assumption since for any set of inequalities C, there exists an equisatisfiable set C', where every variable in C' is bounded [74]. But these theoretical bounds are of little practical value since even for very small problems (< 10 variables), unless they are of very specific structure [82], the magnitudes of the bounds obtained this way are beyond any practical algorithmic reasoning.

In contrast, our procedure *guarantees termination* directly. We describe two arguments that imply termination. First, we propose a simple heuristic for deciding when a cutting-planes based approach does not terminate, recognizing variables contributing to the divergence. Then, we show that, in such a case, one can isolate a finite number of small *conflicting cores* that are inconsistent with the corresponding current partial models. These cores consist of two inequalities and at most one divisibility constraint. Finally, we apply Cooper's quantifier elimination procedure to derive a resolvent that will *block* a particular core from ever happening again, which in turn implies termination. And, as a matter of practical importance, the resolvents do not involve disjunctions and are expressed only with valid inequalities and divisibility constraints.

1.1 Preliminaries

As usual, we will denote the set of integers as $\mathbb{Z}$. We assume a finite set of variables X ranging over $\mathbb{Z}$ and use x, y, z, k to denote variables, a, b, c, d to denote constants from $\mathbb{Z}$, and p, q, r and s for linear polynomials over X with coefficients in $\mathbb{Z}$. In the following, all polynomials are assumed to be in sum-of-monomials normal form $a_1x_1 + \cdots + a_nx_n + c$. Given a polynomial $p = a_1x_1 + \ldots + a_nx_n + c$, and a constant b, we use bp to denote the polynomial $(a_1b)x_1 + \ldots + (a_nb)x_n + (bc)$.

The main constraints we will be working with in this chapter are linear *inequalities*, which are of the form

$$a_n x_n + \cdots + a_1 x_1 + c \leq 0 \ ,$$

which we will denote with letters I and J. We assume the above form for all inequalities as, in the case of integers, we can rewrite $p < 0$ as $p + 1 \leq 0$, and $p = 0$ as $(p \leq 0) \wedge (-p \leq 0)$. In order to isolate the coefficient of a variable x in a linear polynomial p (inequality I), we will write $\mathsf{coeff}(p, x)$ ($\mathsf{coeff}(I, x)$), and we define $\mathsf{coeff}(p, x) = 0$ if x does not occur in p. We say that an inequality I is *tightly-propagating* for a variable x if $\mathsf{coeff}(I, x) \in \{-1, 1\}$.

In addition to inequalities, we also consider *divisibility constraints* of the form

$$d \ \mid \ a_1 x_1 + \cdots + a_n x_n + c \ ,$$

where d is a non-zero integer constant. We denote divisibility constraints with the (possibly subscripted) letter D.

Finally, given a set of constraints C and a constraint I, we use $C \vdash_{\mathbb{Z}} I$ to denote that I is implied by C in the theory of linear integer arithmetic.

1.2 A Cutting-Planes Proof System

In this section, we introduce a cutting-planes proof system that will be the basis of our procedure. Each rule consists of the premises on the top and derives the conclusion at the bottom of the rule, with the necessary side-conditions presented in the box on the side.

The Combine rule derives a positive linear combination of two integer inequalities.

$$\textsc{Combine} \ \frac{I_1 \qquad I_2}{\lambda_1 I_1 + \lambda_2 I_2} \ \textbf{if} \ \boxed{\lambda_1, \lambda_2 > 0}$$

A special case of the above rule is the resolution step used in the Fourier-Motzkin elimination procedure that eliminates the top variable from a pair of inequalities $-ax + p \leq 0$ and $bx - q \leq 0$, generating the inequality $bp - aq \leq 0$.

In the context of integers the main rule, one that is valid for the integers, but not in the rationals, and is the essence of any cutting-planes proof system, is based on strengthening an

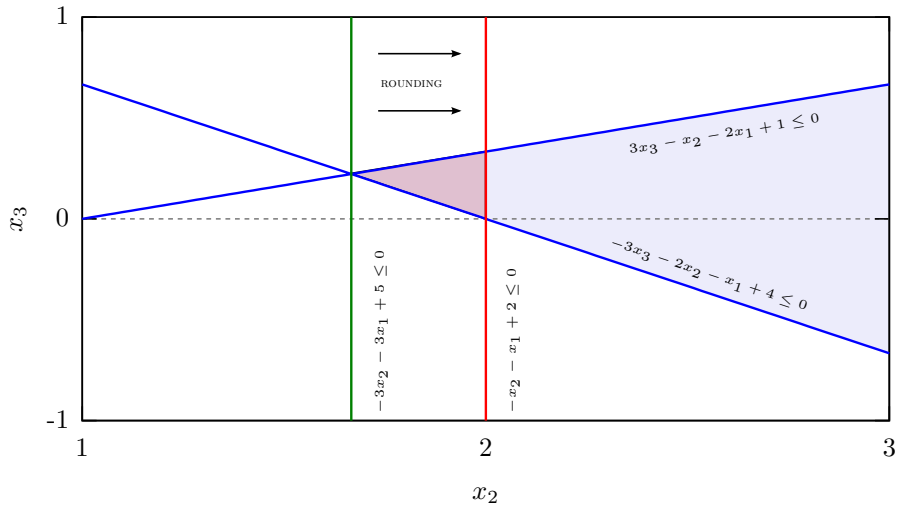


Figure 1.1: Cutting-plane derivation of Example 1.1.

inequality by rounding. The NORMALIZE rule divides an inequality with the greatest common divisor of variable coefficients, while rounding the free constant.

$$\textsc{Normalize}\ \frac{a_1x_1 + \ldots + a_nx_n + c \leq 0}{\frac{a_1}{d}x_1 + \ldots + \frac{a_n}{d}x_n + \lceil \frac{c}{d} \rceil \leq 0}\ \textbf{if}\ \boxed{d = \mathsf{gcd}(a_1, \ldots, a_n)}$$

Example 1.1. Consider the two inequalities $-3x_3 - 2x_2 - x_1 + 4 \leq 0$ and $3x_3 - x_2 - 2x_1 + 1 \leq 0$. We can apply the COMBINE rule with coefficients $\lambda_1 = \lambda_2 = 1$, simulating Fourier-Motzkin elimination, to derive an inequality where the top variable x_3 is eliminated, and then normalizing the result, obtaining the following derivation.

$$\textsc{Combine}\ \frac{\textsc{normalize}\ \dfrac{-3x_3 - 2x_2 - x_1 + 4 \leq 0 \qquad 3x_3 - x_2 - 2x_1 + 1 \leq 0}{-3x_2 - 3x_1 + 5 \leq 0}}{-x_2 - x_1 + 2 \leq 0}$$

This derivation is depicted in Figure 1.1, where it is more evident how the rounding helps eliminate the non-integer parts of the solution space. The shaded part of the figure corresponds to she real solutions of the inequalities at $x_1 = 0$, and the part of this space that does not contain any integer solutions is removed (cut off) by the derived inequality (cutting

plane). Since we are interested only in integer solutions, performing rounding on an inequality corresponds to pushing the hyper-plane that defines the border of the space defined by the inequality, until it touches at least one integer point.

1.3 The Abstract Search Procedure

We describe our procedure as an abstract transition system in the spirit of the Abstract DPLL procedure [62]. The states of the transition system are pairs of the form $\langle M, C\rangle$, where M is a sequence of *bound refinements*, and C is a set of constraints. We use $[\![]\!]$ to denote the empty sequence. In this section we assume that all constraints in C are inequalities. Bound refinements in M can be either *decisions* or *implied bounds*. Decided lower and upper bounds are decisions we make during the search, and we represent them in M as $x \geq b$ and $x \leq b$. On the other hand, lower and upper bounds that are implied in the current state by some inequality I, are represented as $x \geq_I b$ and $x \leq_I b$. We say that a sequence of bound refinements M is *non-redundant* if, for all variables x, the bound refinements in M are monotone, i.e. all the lower (upper) bounds are increasing (decreasing), and M does not contain the same bound for x, decided or implied.

Let $\mathsf{lower}(x, M)$ and $\mathsf{upper}(x, M)$ denote the best, either decided or implied, lower and upper bounds for the variable x in the sequence M, where we assume the usual values of $-\infty$ and ∞ when the corresponding bounds do not exist. We say that a sequence M is *consistent* if there is no variable x such that $\mathsf{lower}(x, M) > \mathsf{upper}(x, M)$. The best lower and upper bound functions can be lifted to linear polynomials using identities such as: $\mathsf{lower}(p+q, M) = \mathsf{lower}(p, M) + \mathsf{lower}(q, M)$, when variables in p and q are disjoint, $\mathsf{lower}(b, M) = b$, and $\mathsf{lower}(ax, M) = a(\mathsf{lower}(x, M))$ if $a > 0$, and $\mathsf{lower}(ax, M) = a(\mathsf{upper}(x, M))$ otherwise.[1]

Given a sequence of bound refinements M, an inequality I containing a variable x can imply a bound on x. To capture this we define the function $\mathsf{bound}(I, x, M)$ representing the implied

[1] In general, when estimating bounds of polynomials, since two polynomials might have variables in common, for a consistent sequence M it holds that, if $\mathsf{lower}(p, M)$ and $\mathsf{lower}(q, M)$ are defined, then $\mathsf{lower}(p + q, M) \geq \mathsf{lower}(p, M) + \mathsf{lower}(q, M)$.

bound as

$$\mathsf{bound}(ax + p \leq 0, x, M) = \begin{cases} -\lceil \frac{\mathsf{lower}(p,M)}{a} \rceil & \text{if } a > 0 \ , \\ -\lfloor \frac{\mathsf{lower}(p,M)}{a} \rfloor & \text{if } a < 0 \ . \end{cases}$$

Definition 1.1 (Well-Formed Sequence). We say a sequence M is *well-formed* with respect to a set of constraints C when M is non-redundant, consistent and M is either an empty sequence or is of the form $M = [\![M', \gamma]\!]$, where the prefix M' is well-formed and the bound refinement γ is either

- $x \geq_I b$, with $I \equiv (-x + q \leq 0)$, $C \vdash_{\mathbb{Z}} I$, and $b \leq \mathsf{lower}(q, M')$; or
- $x \leq_I b$, with $I \equiv (x - q \leq 0)$, $\mathcal{C} \vdash_{\mathbb{Z}} I$, and $b \geq \mathsf{upper}(q, M')$; or
- $x \geq b$, where M' contains $x \leq_I b$; or
- $x \leq b$, where M' contains $x \geq_I b$.

Intuitively, in a well-formed sequence, every decision $x \geq b$ ($x \leq b$) amounts to *deciding* a value for x that is equal to the best upper (lower) bound so far in the sequence. Additionally all the *implied bounds are justified by tight inequalities* that are implied in $\mathbb{Z}$ by the set of constraints C. We say that a state $\langle M, C \rangle$ is well-formed if M is well-formed with respect to C. Note that in the first two properties, when refining a bound, we allow the new bound b to *not necessarily be the most precise one* with respect to I. Although this is unintuitive, the reason for this flexibility will become apparent later.

Given an implied lower (upper) bound refinement $x \geq_I b$ ($x \leq_I b$) and an inequality $ax+p \leq 0$, we define the function $\mathsf{resolve}$ that combines (if possible) the tight inequality $I \equiv \pm x + q \leq 0$ with $ax + p \leq 0$ to eliminate the variable x. If the combination is not applicable, $\mathsf{resolve}$ just returns $ax + p \leq 0$. It is defined as

$$\begin{matrix} \mathsf{resolve}(x \geq_I b, ax + p \leq 0) \\ \mathsf{resolve}(x \leq_I b, ax + p \leq 0) \end{matrix} = \begin{cases} |a|q + p \leq 0 & \text{if } a \times \mathsf{coeff}(I, x) < 0 \ , \\ ax + p \leq 0 & \text{otherwise} \ . \end{cases}$$

The $\mathsf{resolve}$ function will be used in conflict resolution due to the property that it eliminates the variable x if possible, while keeping valid deductions, and the fact that it preserves (or

even improves) the bounds that can be implied. The following lemma states this property more precisely.

Lemma 1.1. *Given a well-formed state $\langle M, C\rangle$, with $M = [\![M', \gamma]\!]$, such that γ is an implied bound, $p \leq 0$ an inequality, and $q \leq 0 \equiv \mathit{resolve}(\gamma, p \leq 0)$ then*

$$C \vdash_{\mathbb{Z}} (p \leq 0) \quad \textit{implies} \quad C \vdash_{\mathbb{Z}} (q \leq 0) \ , \tag{1.1}$$

$$\mathit{lower}(q, M') \geq \mathit{lower}(p, M) \ . \tag{1.2}$$

Proof. Having that γ is an implied bound, we need only consider the following two cases:

1. γ is of the form $x \geq_I b$, where $I \equiv (-x + r \leq 0)$;

2. γ is of the form $x \leq_I b$, where $I \equiv (x - r \leq 0)$;

Let us consider only the first case, as the proof of the second case is similar. Since $\langle M, C\rangle$ is a well-formed state and $M = [\![M', \gamma]\!]$, we have that $b \leq \mathsf{lower}(r, M')$, and that $C \vdash_{\mathbb{Z}} -x + r \leq 0$, by definition. We consider two cases based on the sign of the coefficient of x in p.

- If p is of the form $-ax + s$, for some $a \geq 0$ then by the definition of $\mathsf{resolve}$, we have that $\mathsf{resolve}(\gamma, p \leq 0) = p \leq 0$. Then, $q = p$, and $C \vdash_{\mathbb{Z}} (q \leq 0)$. Moreover $\mathsf{lower}(p, M) = \mathsf{lower}(p, M') = \mathsf{lower}(q, M')$, by the definition of lower.

- If p is of the form $ax + s$, for some $a > 0$. By the definition of $\mathsf{resolve}$, we have that $\mathsf{resolve}(\gamma, p \leq 0) = ar + s \leq 0$. Then, $C \vdash_{\mathbb{Z}} (q \leq 0)$, since $q = ar + s$ is a positive linear combination of the inequalities $p \leq 0$ and $-x + r \leq 0$. Finally, we have that

$$\begin{aligned} \mathsf{lower}(q, M') = \mathsf{lower}(ar + s, M') &\geq a(\mathsf{lower}(r, M')) + \mathsf{lower}(s, M') \\ &\geq a(\mathsf{lower}(x, M)) + \mathsf{lower}(s, M) = \mathsf{lower}(p, M) \ . \end{aligned}$$

Since in both of the cases the statement holds, this concludes the proof. □

Example 1.2. In the statement of Lemma 1.1, we get to keep (or improve) the bound $\mathsf{lower}(q, M') \geq \mathsf{lower}(p, M)$ only because all of the implied bounds were justified by tightly-propagating inequalities. If we would allow non-tight justifications, this might not hold. Consider, for example, a state $\langle M, C\rangle$ where

$$C = \{\overbrace{-x \leq 0}^{I},\ \overbrace{-3y + x + 2 \leq 0}^{J}\}\ , \qquad M = [\![x \geq_I 0,\ y \geq_J 1]\!]\ ,$$

and the inequality $1 + 6y \leq 0$, i.e. the propagation of the bound on y is propagated by a non-tight inequality J . Then, we have that

$$\mathsf{lower}(1 + 6y, M) = 7\ ,$$
$$\mathsf{resolve}(y \geq_J 1,\ 1 + 6y \leq 0) = 2x + 5 \leq 0\ .$$

So, after performing resolution on y using a non-tight inequality J, the inequality became weaker since i.e $\mathsf{lower}(2x + 5, [\![x \geq_I 0]\!]) = 5 \not\geq 7$.

Finally, we define a predicate $\mathsf{improves}(I, x, M)$ as a shorthand for stating that inequality $I \equiv ax + p \leq 0$ implies a better bound for x in M, but does not make M inconsistent. It is defined as

$$\mathsf{improves}(I, x, M) = \begin{cases} \mathsf{lower}(x, M) < \mathsf{bound}(I, x, M) \leq \mathsf{upper}(x, M), & \text{if } a < 0, \\ \mathsf{lower}(x, M) \leq \mathsf{bound}(I, x, M) < \mathsf{upper}(x, M), & \text{if } a > 0, \\ \mathsf{false}, & \text{otherwise.} \end{cases}$$

1.3.1 Deriving tight inequalities

Since we require that all the implied bound refinements in a well-formed sequence M are justified by tightly-propagating inequalities, and we've hinted that this is important for a concise conflict resolution procedure, we will now show how to deduce such tightly-propagating inequalities when needed in bound refinement. Given an inequality $\pm ax + p \leq 0$ such that $\mathsf{improves}(\pm ax + p \leq 0, x, M)$ holds, we show how to deduce a tightly propagating inequality that can justify the

improved bound implied by $\pm ax + p \leq 0$.

The intuition behind the derivation is the following. Starting with an inequality $I \equiv ax + b_1 y_1 + \cdots + b_n y_n \leq 0$, that implies a bound on x, we will transform it using valid deduction steps into an inequality where all coefficients are divisible by a. We can do this since, in order for I to be able to imply a bound on x, the appropriate bounds for the variables $y_1, \ldots, y_n$ have to exist, and moreover these bounds are justified by tightly-propagating inequalities. For example, the bound on variable y_1 might be justified by the inequality $J \equiv -y_1 + q \leq 0$. If so, we can add the inequality J to I as many times as needed to make the coefficient with y_1 divisible by a.

The deduction is described using an auxiliary transition system with the states of this system being tuples of the form

$$\langle M', \pm ax + as \oplus r \rangle \quad ,$$

where $a > 0$, s and r are polynomials, M' is a prefix of the initial M, and we keep the invariant that

$$C \vdash_{\mathbb{Z}} \pm ax + as + r \leq 0, \qquad \mathsf{lower}(as + r, M) \geq \mathsf{lower}(p, M) \quad .$$

The invariant above states that the derived inequality is a valid deduction that implies at least as strong of a bound on x, while the coefficients to the left of the delimiter symbol $\oplus$ are divisible by a.

The initial state for tightening of the inequality $\pm ax + p \leq 0$ is $\langle M, \pm ax \oplus p \rangle$ and the transition rules are listed below.

Consume

$\langle M, \pm ax + as \oplus aky + r \rangle \quad \Longrightarrow \quad \langle M, \pm ax + as + aky \oplus r \rangle$

where $x \neq y$.

Resolve-Implied

$\langle [\![M, \gamma]\!], \pm ax + as \oplus p \rangle \quad \Longrightarrow \quad \langle M, \pm ax + as \oplus q \rangle$

where γ is an implied bound and $q \leq 0 \equiv \mathsf{resolve}(\gamma, p \leq 0)$

Decided-Lower

$\langle [\![M, y \geq b]\!], \pm ax + as \oplus cy + r \rangle \implies \langle M, \pm ax + as + aky \oplus r + (ak - c)q \rangle$

where $y \leq_I b$ in M, with $I \equiv y + q \leq 0$, and $k = \lceil c/a \rceil$.

Decided-Lower-Neg

$\langle [\![M, y \geq b]\!], \pm ax + as \oplus cy + r\rangle \implies \langle M, \pm ax + as \oplus cq + r\rangle$

where $y \leq_I b$ in M, with $I \equiv y - q \leq 0$, and $c < 0$.

Decided-Upper

$\langle [\![M, y \leq b]\!], \pm ax + as \oplus cy + r\rangle \implies \langle M, \pm ax + as + aky \oplus r + (c - ak)q\rangle$

where $y \geq_I b$ in M, with $I \equiv -y + q \leq 0$, and $k = \lfloor c/a \rfloor$.

Decided-Upper-Pos

$\langle [\![M, y \leq b]\!], \pm ax + as \oplus cy + r\rangle \implies \langle M, \pm ax + as \oplus cq + r\rangle$

where $y \geq_I b$ in M, with $I \equiv -y + q \leq 0$, and $c > 0$.

Round (and terminate)

$\langle M, \pm ax + as \oplus b\rangle \implies \pm x + s + \lceil b/a \rceil \leq 0$

We use $\mathsf{tight}(I, x, M)$ to denote the tightly propagating inequalities derived using some strategy for applying the transition rules above.

Example 1.3. Given a well-formed state $\langle M, C\rangle$, where

$$C = \{\underbrace{-y \leq 0}_{I_1}, \underbrace{-x + 2 \leq 0}_{I_2}, \underbrace{-y + 7 + x \leq 0}_{I_3}, \underbrace{-3z + 2y - 5x \leq 0}_{I_4}\}$$

$$M = [\![\ y \geq_{I_1} 0,\ \ x \geq_{I_2} 2,\ \ y \geq_{I_3} 9,\ \ x \leq 2\ ]\!]$$

In this state we have that $\mathsf{bound}(I_4, z, M) = 3$, that is, I_4 is implying a lower bound of z in the current state Since I_4 is not tightly-propagating on z. we now derive a tight inequality that justifies this lower bound by applying the rules as we go backwards in the trail of bound refinements.

$\langle [\![\ y \geq_{I_1} 0,\ \ x \geq_{I_2} 2,\ \ y \geq_{I_3} 9,\ \ x \leq 2\ ]\!], -3z \oplus 2y - 5x\rangle$

$\implies$ Decided-Upper-Pos

$x \leq 2$ is a decided bound, M contains implied bound $x \geq_{I_2} 2$.

We make the coefficient of x divisible by 3 by adding $I_2 \equiv -x + 2 \leq 0$.

$\langle [\![\ y \geq_{I_1} 0,\ \ x \geq_{I_2} 2,\ \ y \geq_{I_3} 9\ ]\!], -3z - 6x \oplus 2y + 2 \rangle$

$\Longrightarrow$ Resolve-Implied

We eliminate y by adding two times $I_3 \equiv -y + 7 + x \leq 0$.

$\langle [\![\ y \geq_{I_1} 0,\ \ x \geq_{I_2} 2\ ]\!], -3z - 6x \oplus 2x + 16 \rangle$

$\Longrightarrow$ Resolve-Implied

We eliminate x in $2x + 16$ by adding two times $I_2 \equiv -x + 2 \leq 0$.

$\langle [\![\ y \geq_{I_1} 0\ ]\!], -3z - 6x \oplus 20 \rangle$

$\Longrightarrow$ Round

$-z - 2x + 7 \leq 0$

The derived tightly propagating inequality $-z - 2x + 7 \leq 0$ implies the same lower bound $\mathsf{bound}(-z - 2x + 7 \leq 0, z, M) = 3$ for z.

The following lemma shows that by deriving tightly propagating inequalities using the system above we do not lose precision in terms of the bounds that the inequality can imply.

Lemma 1.2. *Given a well-formed state $\langle M, C \rangle$ and an implied inequality I, i.e. such that $C \vdash_{\mathbb{Z}} I$, and $\mathsf{improves}(I, x, M)$ the procedure for deriving tightly-propagating inequalities terminates with a tight-inequality J such that $C \vdash_{\mathbb{Z}} J$ and*

- *if I improves the lower bound on x, then $\mathsf{bound}(I, x, M) \leq \mathsf{bound}(J, x, M)$,*
- *if I improves the upper bound on x, then $\mathsf{bound}(I, x, M) \geq \mathsf{bound}(J, x, M)$.*

Proof. Note that for any inequality $I \equiv \pm ax + p \leq 0$ as in the statement of the lemma, i.e. one that improves a bound of x in M, the bounds of all variables from I, except for maybe x, are justified in M. Moreover, since we're in a well-formed state, all of the inequalities that justify these bounds are also tightly-propagating.

For the initial state $\langle M, \pm ax \oplus p \rangle$, all the variables in p have a bound in M. The transition system then keeps the following invariants for any reachable state $\langle M_k, \pm ax + q \oplus r \rangle$:

1. all the variables in r have a bound in M_k;

2. all the variables in q have a bound in M;

3. all the coefficients in q are divisible by a; and

4. $\mathsf{lower}(q + r, M) \geq \mathsf{lower}(p, M)$.

Proving these invariants is an easy exercise, with the interesting and important case being (4), which follows in a manner similar to Lemma 1.1. The cases where the transition rule eliminates a variable follow as in Lemma 1.1. Assume therefore that we are in the case when we don't eliminate the top variable. For example, assume a state where the decided lower bound of y in M_k (and hence in M) is at b.

$$\langle [\![M_{k-1}, y \geq b]\!], \pm ax + q \oplus r \rangle \quad ,$$

Then y must have an implied upper bound $y \leq_I b$ in M_{k-1}, and we add a positive multiple of a tightly-propagating inequality $I \equiv y + t \leq 0$. Note that in M_{k-1}, by property of implied bounds in the definition of the well-formed state, we have that $b \geq \mathsf{upper}(-t, M) = -\mathsf{lower}(t, M)$. Now, for any $\lambda > 0$ we then have

$$\begin{aligned}
\mathsf{lower}(q + r + \lambda(y + t), M) &\geq \mathsf{lower}(q + r, M) + \lambda(\mathsf{lower}(y, M) + \mathsf{lower}(t, M)) && (1.3)\\
&\geq \mathsf{lower}(q + r, M) + \lambda(\mathsf{lower}(y, M) + \mathsf{lower}(t, M_{k-1})) && (1.4)\\
&\geq \mathsf{lower}(q + r, M) + \lambda(\mathsf{lower}(y, M) - b) && (1.5)\\
&= \mathsf{lower}(q + r, M) \ . && (1.6)
\end{aligned}$$

The inequality (1.3) holds just through computation of lower and it is not an equality as some the variables in the terms might be shared, as discussed in the definition of lower. The inequality (1.4) holds as lower bounds on terms can only increase in a well-formed state and M_{k-1} is a subsequence of M. The inequality (1.5) holds since as discussed above we have that $\mathsf{lower}(t, M_{k-1}) \geq -b$. Finally, (1.6) holds simply be definition of lower when a variable has a decided value in a well-formed state.

The improvement of bounds stated in the lemma then easily follows from (4). Termination follows directly, as at least one of the rules is always applicable (using (1)), and each rules either consumes a part of the sequence M or terminates. The length of the derivation is therefore bounded by the length of the sequence M. □

Note that in the statement above, $\mathsf{improves}(J, x, M)$ does not necessarily hold, although the implied bound is the same or better. This is because the $\mathsf{improves}$ predicate requires the new bound to be consistent, and the derived inequality might in fact imply a stronger bound that can be in conflict.

1.3.2 Main procedure

We are now ready to define the main transition system of the decision procedure. In the following system of rules, if a rule can derive a new implied bound $x \geq_I b$ or $x \leq_I b$, the tightly propagating inequality I is written as if computed eagerly. This simplification clarifies the presentation, but we can use them as just placeholders and compute them on demand, which is what we do in our implementation. The transition rules alternate between the *search phase* with states denoted as $\langle M, C\rangle$, where new bounds are propagated and decisions on variables are made, and the *conflict resolution* phase with states denoted as $\langle M, C\rangle \vdash I$, where we try to explain the conflict encountered by the search phase.

Decide

$$\langle M, C\rangle \implies \langle [\![M, x \geq b]\!], C\rangle \quad \textbf{if } \mathsf{lower}(x, M) < b = \mathsf{upper}(x, M)$$

$$\langle M, C\rangle \implies \langle [\![M, x \leq b]\!], C\rangle \quad \textbf{if } \mathsf{lower}(x, M) = b < \mathsf{upper}(x, M)$$

Propagate

$$\langle M, C \cup \{J\}\rangle \implies \langle [\![M, x \geq_I b]\!], C \cup \{J\}\rangle \textbf{ if } \begin{cases} \mathsf{coeff}(J, x) < 0 \\ \mathsf{improves}(J, x, M), \\ I = \mathsf{tight}(J, x, M), \\ b = \mathsf{bound}(J, x, M). \end{cases}$$

$$\langle M, C \cup \{J\}\rangle \implies \langle [\![M, x \leq_I b]\!], C \cup \{J\}\rangle \textbf{ if } \begin{cases} \mathsf{coeff}(J, x) > 0 \\ \mathsf{improves}(J, x, M), \\ I = \mathsf{tight}(J, x, M), \\ b = \mathsf{bound}(J, x, M). \end{cases}$$

Forget

$$\langle M, C \cup \{J\}\rangle \implies \langle M, C\rangle \quad \textbf{if } C \vdash_{\mathbb{Z}} J, \text{ and } J \notin C$$

Conflict

$$\langle M, C\rangle \implies \langle M, C\rangle \vdash p \le 0 \qquad \textbf{if } p \le 0 \in C,\ \mathsf{lower}(p, M) > 0$$

Learn

$$\langle M, C\rangle \vdash I \implies \langle M, C \cup I\rangle \vdash I \qquad \textbf{if } I \notin C$$

Resolve

$$\langle [\![M, \gamma]\!], C\rangle \vdash I \implies \langle M, C\rangle \vdash \mathsf{resolve}(\gamma, I) \quad \textbf{if} \quad \gamma \text{ is an implied bound.}$$

Skip-Decision

$$\langle [\![M, \gamma]\!], C\rangle \vdash p \le 0 \implies \langle M, C\rangle \vdash p \le 0 \qquad \textbf{if} \begin{cases} \gamma \text{ is a decided bound} \\ \mathsf{lower}(p, M) > 0 \end{cases}$$

Unsat

$$\langle [\![M, \gamma]\!], C\rangle \vdash b \le 0 \implies \mathsf{unsat} \qquad \textbf{if } b > 0$$

Backjump

$$\langle [\![M, \gamma, M']\!], C\rangle \vdash J \implies \langle [\![M, x \ge_I b]\!], C\rangle \qquad \textbf{if} \begin{cases} \gamma \text{ is a decided bound} \\ \mathsf{coeff}(J, x) < 0 \\ \mathsf{improves}(J, x, M), \\ I = \mathsf{tight}(J, x, M), \\ b = \mathsf{bound}(J, x, M). \end{cases}$$

$$\langle [\![M, \gamma, M']\!], C\rangle \vdash J \implies \langle [\![M, x \le_I b]\!], C\rangle \qquad \textbf{if} \begin{cases} \gamma \text{ is a decided bound} \\ \mathsf{coeff}(J, x) > 0 \\ \mathsf{improves}(J, x, M), \\ I = \mathsf{tight}(J, x, M), \\ b = \mathsf{bound}(J, x, M). \end{cases}$$

When applying the rules above, the newly introduced inequalities are either tight versions of existing inequalities, or introduced during conflict resolution. In both cases we can see from Lemma 1.1 and Lemma 1.2 and simple inductive reasoning that these new inequalities are implied by the original problem.

Theorem 1.2 (Soundness). *For any derivation sequence* $\langle [\![]\!], C_0\rangle \implies S_1 \implies \cdots \implies S_n$. *If*

S_n is of the form $\langle M_n, C_n \rangle$, then C_0 and C_n are equisatisfiable. If S_n is of the form $\langle M_n, C_n \rangle \vdash I$, then C_0 implies I, and C_0 and C_n are equisatisfiable. Moreover, $\langle M_n, C_n \rangle$ is well-formed.

Example 1.4. Consider the set of inequalities C

$$\{\underbrace{-x \leq 0}_{I_1},\ \underbrace{6x - 3y - 2 \leq 0}_{I_2},\ \underbrace{-6x + 3y + 1 \leq 0}_{I_3}\}$$

Now we show C to be unsatisfiable using our abstract transition system.

$\langle [\![]\!], C \rangle$

$\Longrightarrow$ Propagate x using $I_1 \equiv -x \leq 0$

$\langle [\![x \geq_{I_1} 0]\!], C \rangle$

$\Longrightarrow$ Decide x

$\langle [\![x \geq_{I_1} 0,\ x \leq 0]\!], C \rangle$

$\Longrightarrow$ Propagate y using $I_3 \equiv -6x + 3y + 1 \leq 0$

$\langle \overbrace{[\![x \geq_{I_1} 0,\ x \leq 0,\ y \leq_J -1]\!]}^{M}, C \rangle$, where $J = \mathsf{tight}(I_3, y, [\![x \geq_{I_1} 0, x \leq 0]\!])$

$\langle [\![x \geq_{I_1} 0,\ x \leq 0]\!], 3y \oplus -6x + 1 \rangle$

$\Longrightarrow$ Consume

$\langle [\![x \geq_{I_1} 0,\ x \leq 0]\!], 3y - 6x \oplus 1 \rangle$

$\Longrightarrow$ Round

$J \equiv y - 2x + 1 \leq 0$

$\Longrightarrow$ Conflict using $I_2 \equiv 6x - 3y - 2 \leq 0$, since $\mathsf{lower}(6x - 3y - 2, M) > 0$

$\langle [\![x \geq_{I_1} 0,\ x \leq 0,\ y \leq_J -1]\!], C \rangle \vdash 6x - 3y - 2 \leq 0$

$\Longrightarrow$ Resolve $\quad \mathsf{resolve}(y \leq_J -1, 6x - 3y - 2 \leq 0) = (3(-2x + 1) + 6x - 2 \leq 0)$

$\langle [\![M, x \leq 0]\!], C \rangle \vdash 1 \leq 0$

$\Longrightarrow$ Unsat

unsat

Slack Introduction. One of the obvious drawbacks of the presented system is that it is easy to find a system of constraints where the transition system can not make any progress, for example if the constraints don't contain any explicit variable bounds. This can be fixed by introducing fresh variables that will create artificial bounds that can start-up the computation.

Given a state $S = \langle M, C \rangle$, we say that a variable x is *unbounded* at S if $\mathsf{lower}(x, M) = -\infty$ and $\mathsf{upper}(x, M) = \infty$. We also say that x is *stuck* at state S if it is unbounded and the Propagate rule cannot be used to deduce a lower or upper bound for x. A state S is *stuck* if all undecided variables in S are stuck, and no inequality in C is false in M. That is, there is no possible transition for a stuck state S. Before we describe how we avoid stuck states, we make the observation that for every finite set of inequalities C, there is an equisatisfiable set C' such that for every variable x in C', $(-x \leq 0) \in C'$. The idea is to replace every occurrence of x in C with $x^+ - x^-$, and add the inequalities $-x^+ \leq 0$ and $-x^- \leq 0$. Instead of using this eager preprocessing step, we use a lazy approach, where *slack variables* are dynamically introduced. Suppose, we are in a stuck state $\langle M, C \rangle$, then we simply select an unbounded variable x, add a fresh *slack* variable $x_s \geq 0$, and add new inequalities to C that "bound" x in the interval $[-x_s, x_s]$. This idea is captured by the following rule:

Slack-Intro

$$\langle M, C \rangle \Longrightarrow \langle M, C \cup \{x - x_s \leq 0, -x - x_s \leq 0, -x_s \leq 0\} \rangle \textbf{ if } \begin{cases} \langle M, C \rangle \text{ is stuck} \\ x_s \text{ is fresh} \end{cases}$$

Note that it is sound to reuse a slack variable x_s used for "bounding" x, to bound some other variable y, and this is what we do in our implementation.

1.3.3 Termination

We say a set of inequalities C is a *finite problem* if for every variable x in C, there are two integer constants a and b such that $\{x - a \leq 0, -x + b \leq 0\} \subseteq C$. We say a set of inequalities C is an *infinite problem* if it is not finite. That is, there is a variable x in C such that there are no values a and b such that $\{x - a \leq 0, -x + b \leq 0\} \subseteq C$. We say an inequality is *simple* if it is of the form $x - a \leq 0$ or $-x + b \leq 0$. Let Propagate-Simple be a rule such as Propagate, but with an extra condition requiring J to be a simple inequality. We say a strategy for applying the rules is

reasonable if a rule R different from Propagate-Simple is applied only if Propagate-Simple is not applicable. Informally, a *reasonable* strategy prevents the generation of derivations where simple inequalities are ignored and C is essentially treated as an infinite problem.

Theorem 1.3 (Termination). *Given a finite problem C, and a* reasonable *strategy, there is no infinite derivation sequence starting from $\langle [\![]\!], C_0\rangle$.*

Proof. The proof of the statement is an adaptation of the proof used to show termination of the Abstract DPLL [72]. We say that a state $\langle M_i, C_i\rangle$ is *reachable* if there is a derivation sequence

$$\langle [\![]\!], C\rangle \Longrightarrow \cdots \Longrightarrow \langle M_i, C_i\rangle \ .$$

A sequence M is *bounded* if there is no variable x in C such that $\mathsf{lower}(x, M) = -\infty$ or $\mathsf{upper}(x, M) = \infty$. Given a derivation T starting at $\langle [\![]\!], C\rangle$, let $\langle M_0, C_0\rangle$ be the first state in T where Propagate-Simple is not applicable. Then, M_0 is bounded because C is a finite problem. We say $\langle M_0, C_0\rangle$ is the *actual* initial state of T.

The *level* of a state $\langle M_i, C_i\rangle$ is the number of decided bounds in M_i. The level of any reachable state $\langle M_i, C_i\rangle$ is $\leq n$, where n is the number of variables in C. Let $\mathsf{subseq}_j(M)$ denote the maximal prefix subsequence of M of level $\leq j$. Let V denote the set of variables used in C.

First, we define an auxiliary function $w(M)$ as

$$w(M) = \begin{cases} \infty & \text{if } M \text{ is unbounded,} \\ \sum_{x\in V}(\mathsf{upper}(x, M) - \mathsf{lower}(x, M)) & \text{otherwise.} \end{cases}$$

Now, we define a function weight that maps a sequence M into a $(n+1)$-tuple, where n is the number of variables in C. It is defined as

$$\mathsf{weight}(M) = \langle w(\mathsf{subseq}_0(M)), w(\mathsf{subseq}_1(M)), \ldots, w(\mathsf{subseq}_n(M))\rangle \ .$$

Given two bounded sequences M and M', we say $M \prec\!\!\prec M'$ if $\mathsf{weight}(M) <_{\mathsf{lex}} \mathsf{weight}(M')$, where $<_{\mathsf{lex}}$ is the lexicographical extension of the order $<$ on natural numbers.

For any transition $\langle M_i, C_i\rangle \Longrightarrow \langle M_{i+1}, C_{i+1}\rangle$ performed by Decide, Propagate or Propagate-Simple, as these rules only improve the variable bounds, if M_i is bounded, then M_{i+1} is also bounded and $M_{i+1} \prec\!\!\prec M_i$.

Now let's consider the conflict resolution rules. The conflict resolution process starts from a state $\langle M, C\rangle \vdash I$ and then traverses over the elements of the trail backwards. Since the size of the sequence M is finite, conflict resolution is always a finite sequence of steps. For each conflict resolution step of the transition system

$$\langle M_k, C_k\rangle \vdash p \leq 0 \Longrightarrow \langle M_{k+1}, C_{k+1}\rangle \vdash q \leq 0 \ ,$$

the sequence M_{k+1} is a subsequence of M_k, and the rules keep as invariant the fact that $q \leq 0$ is a valid deduction and $\mathsf{lower}(q, M) > 0$. For applications of the Resolve rule this follows from Lemma 1.1 and Lemma 1.2, and for applications of the Skip-Decision rule this follows from the preconditions of the rule itself.

Let's show that we can not get stuck in conflict analysis, i.e. that a transition using the conflict resolution rules is always possible. Assume that no rule other than possibly Backjump is applicable, i.e. that we are in a state $\langle M_k, C_k\rangle \vdash p \leq 0$ where $M_k = [\![M'_k, \gamma]\!]$, the top trail element γ is a decided bound (Resolve is not applicable), and $\mathsf{lower}(p, M'_k) \leq 0$ (Skip-Decision is not applicable). If $\gamma = x \leq b$, then we know that $\mathsf{lower}(x, M'_k) = b$ and, additionally, that $p = -ax + q$ for some $a > 0$ as otherwise we would have that $\mathsf{lower}(p, M_k) = \mathsf{lower}(p, M'_k) > 0$. We can now compute

$$0 < \mathsf{lower}(-ax + q, M_k) = -a\mathsf{upper}(x, M_k) + \mathsf{lower}(q, M_k) = -ab + \mathsf{lower}(q, M'_k) \ , \qquad (1.7)$$

and therefore $\mathsf{lower}(q, M'_k) > ab$. From here we see that $-ax + q \leq 0$ implies a lower bound $\mathsf{bound}(p \leq 0, x, M'_k) > b$ on x, improving on the current $\mathsf{lower}(x, M'_k) = b$. For the Backjump rule to be applicable we must also show that the new bound does not exceed any existing upper bounds in M'_k. Assume the opposite, i.e. that

$$\mathsf{upper}(x, M'_k) < \mathsf{bound}(p \leq 0, x, M'_k) = \left\lceil \frac{\mathsf{lower}(q, M'_k)}{a} \right\rceil \ .$$

But then we can conclude that

$$\begin{aligned} \mathsf{lower}(-ax + q, M'_k) &= -a\mathsf{upper}(x, M'_k) + \mathsf{lower}(q, M'_k) \\ &> -a\mathsf{upper}(x, M'_k) + a\mathsf{upper}(x, M'_k) = 0 \ . \end{aligned}$$

This contradicts our assumption and therefore the new bound does not exceed the existing bound

and the Backjump rule is applicable. Similarly, if $\gamma = x \geq b$ we can conclude that the Backjump rule is applicable.

The only way to exit the conflict analysis state and get back into the search mode, is by an application of the Backjump rule. But, for any transition $\langle M_i, C_i\rangle \vdash p \leq 0 \Longrightarrow \langle M_{i+1}, C_{i+1}\rangle$ performed by the Backjump rule, since this transition can backtrack at most up to the first decision and will therefore not eliminate the bounded initial state M_0, if $\mathsf{subseq}_0(M_i)$ is bounded, then M_{i+1} is also bounded and $M_{i+1} \prec\!\!\!\prec M_i$. Tough Backjump may eliminate several bounds from M_i, it improves the bound of a variable in some lower level. Since, for finite problems, the *actual* initial state $\langle M_0, C_0\rangle$ is bounded and $\prec\!\!\!\prec$ is well-founded for bounded states, we have that any derivation will eventually terminate. □

As mentioned in the introduction, for infinite problems a termination argument can be constructed using the fact that for any set of inequalities C, there is an equisatisfiable C' where every variable in C' is bounded, but it has little practical value. In Section 1.4, we describe an extra set of rules that guarantee termination even for infinite problems.

1.3.4 Relevant propagations

Unlike in SAT and Pseudo-Boolean solvers, the Propagate rules cannot be applied to exhaustion for infinite problems. If C is unsatisfiable, the propagation rules may remain applicable indefinitely.

Example 1.5. Consider the following set of (unsatisfiable) constraints

$$C = \{\overbrace{-x + y + 1 \leq 0}^{I}, \overbrace{-y + x \leq 0}^{J}, \overbrace{-y \leq 0}^{K}\} \ .$$

Starting from the initial state $\langle [\![]\!], C\rangle$, it is possible to generate the following infinite sequence of states by only applying the Propagate rule.

$$\begin{aligned}\langle [\![]\!], C\rangle \Longrightarrow \langle [\![y \geq_K 0]\!], C\rangle \Longrightarrow \langle [\![y \geq_K 0, x \geq_I 1]\!], C\rangle \\ \Longrightarrow \langle [\![y \geq_K 0, x \geq_I 1, y \geq_J 1]\!], C\rangle \Longrightarrow \\ \langle [\![y \geq_K 0, x \geq_I 1, y \geq_J 1, x \geq_I 2]\!], C\rangle \Longrightarrow \ldots\end{aligned}$$

Theorem 1.4. *The* Propagate *rule cannot be applied indefinitely if the initial set of constraints C is satisfiable.*

The theorem above holds since in the case of integers, the feasible space of the constraints C prevents the bounds on variables to be improved indefinitely. Intuitively, since no propagation can remove a solution (propagations are sound), and a solution exists, we can not increase (decrease) a lower (upper) of a variable indefinitely as this would eventually remove the solution. A theorem similar to the above does not hold for real arithmetic, where due to the density of the domain even for satisfiable problems these infinite propagation sequences are possible.

To try and avoid the infinite loops we adopt a simple heuristic. Let $\mathsf{nb}(x, M)$ denote the number of lower and upper bounds for a variable x in the sequence M. Given a state $S = \langle M, C\rangle$, some $\delta > 0$, and a bound on a number of propagations Max, we say a new lower bound $x \geq_I b$ is δ-*relevant* at S if

1. $\mathsf{upper}(x, M) \neq +\infty$, or
2. $\mathsf{lower}(x, M) = -\infty$, or
3. $\mathsf{lower}(x, M) + \delta|\mathsf{lower}(x, M)| < b$ and $\mathsf{nb}(x, M) < \mathsf{Max}$.

The intuition for the above definition of relevancy is as follows. If x has a upper bound, then any lower bound is δ-relevant because x becomes bounded, and termination is not an issue for bounded variables. If x does not already have a lower bound, then any new lower bound $x \geq_I b$ is relevant. Finally, the third case states that the magnitude of the improvement must be significant and the number of bound improvements for x in M must be smaller than Max. In theory, to prevent non-termination during bound propagation we only need the cutoff Max. The condition $\mathsf{lower}(x, M) + \delta|\mathsf{lower}(x, M)| < b$ is pragmatic, and is inspired by an approach used in [1]. The idea is to block any bound improvement for x that is *insignificant* with respect to the already known bound for x.

Even when only δ-relevant propagations are performed, it is still possible to generate an infinite sequence of transitions. The key observation is that Backjump is essentially a propagation rule, that is, it backtracks M, but it also adds a new improved bound for some variable x. It

is easy to construct non-terminating examples, where Backjump is used to generate an infinite sequence of non δ-relevant bounds.

1.4 Strong Conflict Resolution

In this section, we extend our procedure to be able to handle divisibility constraints, by adding propagation, solving and consistency checking rules specific to divisibility constraints into our system. Then we show how to ensure that our procedure terminates even in cases when some variables are unbounded.

Solving divisibility constraints. We will add one proof rule to the proof system, in order to help us keep the divisibility constraints in a normal form. As Cooper originally noticed in [29], given two divisibility constraints, we can always eliminate a variable from one of them, obtaining equivalent constraints.

$$\textsc{div-solve}\ \frac{d_1 \mid a_1 x + p_1, d_2 \mid a_2 x + p_2}{\begin{array}{c} d_1 d_2 \mid dx + \alpha(d_2 p_1) + \beta(d_1 p_2) \\ d \mid a_2 p_1 - a_1 p_2 \end{array}}\ \textbf{if}\ \boxed{\begin{array}{c} d = \mathsf{gcd}(a_1 d_2, a_2 d_1) \\ \alpha(a_1 d_2) + \beta(a_2 d_1) = d \end{array}}$$

Since we could not find the proof of correctness of the above rule in the literature, we provide the following simple one.

Lemma 1.3. *Consider the following two divisibility constraints*

$$d_1 \mid a_1 x + p_1 \ , \tag{1.8}$$

$$d_2 \mid a_2 x + p_2 \ . \tag{1.9}$$

These are equivalent to the divisibility constraints

$$d_1 d_2 \mid dx + \alpha(d_2 p_1) + \beta(d_1 p_2) \ , \tag{1.10}$$

$$d \mid a_2 p_1 - a_1 p_2 \ , \tag{1.11}$$

where $d = \mathsf{gcd}(a_1 d_2, a_2 d_1)$ *and* $\alpha(a_1 d_2) + \beta(a_2 d_1) = d$.

Proof. ($\Rightarrow$) Assume (1.8) and (1.9). Multiplying them with d_2 and d_1, receptively, we also have that $d_1 d_2 \mid d_2 a_1 x + d_2 p_1$ and $d_1 d_2 \mid d_1 a_2 x + d_1 p_2$. We can add these two together, multiplied with α and β to obtain (1.10).

On the other hand, multiplying (1.8) and (1.9) with a_2 and a_1, respectively, we get that $d_1 a_2 \mid a_1 a_2 x + a_2 p_1$ and $d_2 a_1 \mid a_1 a_2 x + a_1 p_2$. Now, since $d = \mathsf{gcd}(a_1 d_2, a_2 d_1)$ we also know that $d \mid a_1 a_2 x + a_2 p_1$ and $d \mid a_1 a_2 x + a_1 p_2$. Subtracting these two we get (1.11).

($\Leftarrow$) Assume (1.10) and (1.11). Using the assumption that $d = \alpha(a_1 d_2) + \beta(a_2 d_1)$, we can be rewrite them as

$$d_1 d_2 \mid \alpha d_2(a_1 x + p_1) + \beta d_1(a_2 x + p_2) \ , \tag{1.12}$$

$$d \mid a_2(a_1 x + p_1) - a_1(a_2 x + p_2) \ . \tag{1.13}$$

Using the first direction applied to above (taking $a_1 x + p_1$ as the variable) we get that

$$d d_2 \mid \mathsf{gcd}(d_1 d_2 a_2, \alpha d d_2) \mid a_2 \beta d_1(a_2 x + p_2) + \alpha d_2 a_1(a_2 x + p_2) \ ,$$
$$d d_2 \mid d(a_2 x + p_2) \ ,$$

form which we get (1.9). Similarly, assuming $a_2 x + p_2$ is the variable, we get (1.8). □

We use the above proof rule in our transition system to enable normalization of divisibility constraints by variable elimination.

Solve-Div

$$\langle M, C\rangle \quad \Longrightarrow \langle M, C'\rangle \qquad \textbf{if} \begin{cases} D_1, D_2 \in C, \\ (D_1', D_2') = \textsc{div-solve}(D_1, D_2), \\ C' = C \setminus \{D_1, D_2\} \cup \{D_1', D_2'\}. \end{cases}$$

Unsat-Div

$$\langle M, C \cup \{(d \mid a_1 x_1 + \cdots + a_n x_n + c)\}\rangle \implies \mathsf{unsat} \quad \textbf{if } gcd(d, a_1, \ldots, a_n) \nmid c$$

Propagation. With divisibility constraints as part of our problem, we can now achieve even more powerful bound propagation. We say a variable x is *fixed* in the state $S = \langle M, C\rangle$ if $\mathsf{upper}(x, M) = \mathsf{lower}(x, M)$. Similarly a polynomial p is fixed if all of its variables are fixed. To clarify the presentation, for fixed variables and polynomials we write $\mathsf{val}(x, M)$ and $\mathsf{val}(p, M)$ as a shorthand for $\mathsf{lower}(x, M)$ and $\mathsf{lower}(p, M)$.

Let $\langle M, C\rangle$ be a well-formed state, let $D \equiv (d \mid ax + p) \in C$ be a divisibility constraint with $a > 0$, $d > 0$, and assume x has a lower bound $x \geq_I b \in M$ with $I \equiv (-x + q)$. Assume, additionally, that p is fixed, i.e. assume that $\mathsf{val}(p, M) = k$. If the bound b does not satisfy the divisibility constraint, i.e. if $d \nmid ab + k = \mathsf{lower}(ax + p, M)$, we can deduce a better lower bound on x to be the next point that satisfies the divisibility constraint.

Let c be the first such point, i.e. the smallest $c > b$ with $d \mid ac + k$. Since $\langle M, C\rangle$ is a well formed state we know that $b \leq \mathsf{lower}(q, M')$ for some prefix M' of M, and therefore $b \leq \mathsf{lower}(q, M)$. Now, in order to satisfy the divisibility constraint we must have an integer z such that $dz = ax + p$, and therefore $I_1 \equiv -dz + ax + p \leq 0$. Note that b, the lower bound of x, does not satisfy the divisibility constraint, and therefore this inequality implies a bound on z that requires rounding. Since p is fixed, and we have a lower bound on x, we can now use our system for deriving tight inequalities to deduce a tightly propagating inequality $I_2 \equiv -z + r \leq 0$ that, in the state, bounds z from below. Moreover, by using a strategy that never uses the Consume rule on the variable x, we can ensure that r does not include x. From Lemma 1.2 we can now conclude that

$$\mathsf{lower}(r, M) \geq \left\lceil \frac{\mathsf{lower}(ax + p, M)}{d} \right\rceil = \left\lceil \frac{ab + k}{d} \right\rceil = \frac{ac + k}{d} \in \mathbb{Z} \ ,$$

with the last inference resulting by choice of c. Now, we use the new inequality I_2 to derive an inequality I_3 that provides a new bound on x.

$$\textsc{Combine}\ \frac{\dfrac{\overbrace{-z + r \leq 0}^{I_2}}{-dz + dr \leq 0} \qquad \dfrac{\overbrace{dz = ax + p}^{D}}{dz - ax - p \leq 0}}{\underbrace{-ax + dr - p \leq 0}_{I_3}}$$

Since we know that r and p don't include x (and therefore x did not get eliminated) we can compute the bound that this inequality infers on x in the current model

$$\mathsf{bound}(I_3, x, M) = \left\lceil \frac{\mathsf{lower}(dr - p, M)}{a} \right\rceil \geq \left\lceil \frac{d\mathsf{lower}(r, M) - k)}{a} \right\rceil \geq \left\lceil \frac{d\frac{ac+k}{d} - k}{a} \right\rceil = c \ .$$

We can also use our procedure to convert this new constraint into a tightly propagating inequality J. Similar reasoning can be applied for the upper bound inequalities.

We denote, as a shorthand, the result of this whole derivation with $J = \mathsf{div\text{-}derive}(I, D, x, M)$. We can now use the derivation above to empower propagation driven by divisibility constraints, as summarized below.

Propagate-Div

$$\langle M, C\rangle \implies \langle [\![M, x \geq_I c]\!], C \cup \{I\}\rangle \qquad \textbf{if} \begin{cases} D \equiv (d \mid ax + p) \in C, \\ (x \geq_J b) \in M,\ \mathsf{val}(p, M) = k \\ d \nmid ab + k,\ d \mid ac + k,\ c \leq \mathsf{upper}(x, M) \\ I = \mathsf{div\text{-}derive}(J, D, x, M) \end{cases}$$

$$\langle M, C\rangle \implies \langle [\![M, x \leq_I c]\!], C \cup \{I\}\rangle \qquad \textbf{if} \begin{cases} D \equiv (d \mid ax + p) \in C, \\ (x \leq_J b) \in M,\ \mathsf{val}(p, M) = k \\ d \nmid ab + k,\ d \mid ac + k,\ c \geq \mathsf{lower}(x, M) \\ I = \mathsf{div\text{-}derive}(J, D, x, M) \end{cases}$$

Note that, as in the case of propagation with inequalities, we do not need to derive the explanation inequality eagerly, but instead only record the new bound and do the derivation on demand, if needed for conflict analysis.

Eliminating Conflicting Cores. For sets of constraints containing unbounded variables, there is no guarantee that the procedure described in the previous section will terminate. In this section, we describe an extension based on Cooper's quantifier elimination procedure that guarantees termination.

Let U be a subset of the variables in X. We say U is the set of *unbounded* variables. Let $\prec$ be a total order over the variables in X such that for all variables $x \in X \setminus U$ and $y \in U$, $x \prec y$. We say a variable x is *maximal* in a constraint C containing x if for all variables y different from x in C, $y \prec x$. For now, we assume U contains all unbounded variables in the set of constraints C, and $\prec$ is fixed. Later, we describe how to dynamically change U and $\prec$ without compromising termination.

An *interval conflicting core* for variable x at state $S = \langle M, C\rangle$ is a set $\{-ax+p \leq 0,\ bx-q \leq 0\}$ such that p and q are fixed at S, and $\mathsf{bound}(-ax + p \leq 0, x, M) > \mathsf{bound}(bx - q \leq 0, x, M)$. A *divisibility conflicting core* for variable x at state S is a set $\{-ax + p \leq 0,\ bx - q \leq 0,\ (d \mid$

$cx + s)\}$ such that p, q and s are fixed, and for all values k in the interval $[\mathsf{bound}(-ax + p \leq 0, x, M), \mathsf{bound}(bx - q \leq 0, x, M)]$, the divisibility constraint does not hold i.e. $(d \nmid ck + \mathsf{val}(s, M))$. We do not consider cores containing more than one divisibility constraint because rule Solve-Div can be used to eliminate all but one of them. From hereafter, we assume a core is always of the form $\{-ax + p \leq 0,\ bx - q \leq 0,\ (d \mid cx + r)\}$, since we can include the redundant divisibility constraint $(1 \mid x)$ in any interval conflicting core. We say x is a *conflicting variable* at state S if there is an interval or divisibility conflicting core for x. The variable x is the *minimal conflicting variable* at S if there is no $y \prec x$ such that y is also a conflicting variable at S. Let x be a minimal conflicting variable at state $S = \langle M, C\rangle$ and $D = \{-ax + p \leq 0,\ bx - q \leq 0,\ (d \mid cx + r)\}$ be a conflicting core for x, then a *strong resolvent* for D is a set R of inequality and divisibility constraints equivalent to

$$\exists x. -ax + p \leq 0 \wedge bx - q \leq 0 \wedge (d \mid cx + r)$$

The key property of R is that in any state $\langle M', C'\rangle$ such that $R \subset C'$, x is not the minimal conflicting variable or D is not a conflicting core.

We compute the resolvent R using Cooper's left quantifier elimination procedure. It can be summarized by the rule

$$\textsc{Cooper-Left}\ \frac{(d \mid cx + s),\ \ -ax + p \leq 0,\ \ bx - q \leq 0}{\begin{array}{c} 0 \leq k \leq m,\ \ bp - aq + bk \leq 0, \\ a \mid k + p,\ \ ad \mid ck + cp + as \end{array}}$$

where k is a fresh variable and $m = \mathsf{lcm}(a, \frac{ad}{\mathsf{gcd}(ad,c)}) - 1$. The fresh variable k is bounded so it does not need to be included in U. We extend the total order $\prec$ to k by making k the minimal variable. For the special case, where $(d \mid cx + s)$ is $(1 \mid x)$, we get that $m = a - 1$ and the rule above simplifies to

$$\frac{-ax + p \leq 0,\ \ bx - q \leq 0}{0 \leq k < a,\ bp - aq + bk \leq 0,\ a \mid p + k}$$

Lemma 1.4. *The* Cooper-Left *rule is sound and produces a strong resolvent.*

Proof. Multiplying the premises with appropriate coefficients we can obtain new, equivalent constraints that have abc as coefficient with x

$$(ab)d \mid (abc)x + (ab)s \ , \tag{1.14}$$

$$(bc)p \leq (abc)x \ , \quad (abc)x \leq (ac)q \ . \tag{1.15}$$

In order for an integer solution to the inequalities above to exist, from the left inequality we can conclude that there must exist a $k \geq 0$ such that $(abc)x = (bc)p + (bc)k$, and therefore

$$a \mid p + k \ .$$

Additionally, there must be enough room for this solution so, it must be that $(ac)q - (bc)p \geq (bc)k$, i.e

$$bp - aq + bk \leq 0 \ .$$

Now, substituting $(abc)x$ into the divisibility constraint we get that $(ab)d \mid (bc)k + (bc)p + (ab)s$, or equivalently that

$$ad \mid ck + cp + as \ .$$

In order to bound k from above, we note that a sufficient (and necessary) condition for a divisibility constraint $a \mid bx + c$ to have a solution, is to have a solution with $0 \leq x < \frac{a}{\mathsf{gcd}(a,b)}$. We use this and deduce that in our case, since we have two divisibility constraints, it must be that

$$0 \leq k < \mathsf{lcm}\left(a, \frac{ad}{\mathsf{gcd}(ad, c)}\right) \ .$$

The rule Cooper-Left is biased to lower bounds. We may also define the Cooper-Right rule that is based on Cooper's right quantifier elimination procedure and is biased to upper bounds. We use $\mathsf{cooper}(D)$ to denote a procedure that computes the strong resolvent R for a conflicting core D. Now, we extend our procedure with a new rule for introducing resolvents for minimal conflicting variables.

Resolve-Cooper

$$\langle M, C\rangle \implies \langle M, C \cup \mathsf{cooper}(D)\rangle \quad \textbf{if} \begin{cases} x \in U, \\ x \text{ is the minimal conflicting variable}, \\ D \text{ is a conflicting core for } x. \end{cases}$$

Note that in addition to fresh variables, the Resolve-Cooper rule also introduces new constraints without resorting to the Learn rule. We will show that this cannot happen indefinitely, as the rule can only be applied a finite number of times.

Lemma 1.5. *For any initial state* $\langle [\![]\!], C\rangle$*, the* Resolve-Cooper *rule can be applied only a finite number of times, if*

- *it is never applied to cores containing inequalities introduced by the* Learn *rule, and;*
- *the* Forget *rule is never used to eliminate resolvents introduced by* Resolve-Cooper*.*

Proof. First notice that, although the Cooper-Left and Cooper-Right rules introduce fresh variables k, these variables are initially bounded, and are therefore never included in the set U. Consequently, these variables are never considered by Resolve-Cooper and, therefore Resolve-Cooper will only apply to the variables from the initial set of constraints C.

Now, consider a conflicting core

$$D = \{-ax + p \leq 0,\ bx - q \leq 0,\ (d \mid cx + r)\}\ ,$$

and a derivation sequence T satisfying the conditions above. In such a derivation sequence, the Resolve-Cooper rule can only be applied once. This is true because the resolvent $R = \mathsf{cooper}(D)$ is equivalent to $\exists x.D$. Although the resolvent introduces a fresh variable, it is a finite one and therefore smaller than all the variables in U. Therefore, for any state where we could try and apply the strong resolution again, i.e. $\langle M', C'\rangle$ such that $R \subseteq C'$, x is not the minimal conflicting variable or D is not a conflicting core. The rule Resolve-Cooper will therefore not be applicable to the same core, at any state that already contains the resolvent R. Additionally, since we do not eliminate resolvents introduced by Resolve-Cooper using the Forget rule, a resolvent for a core D will be generated at most once.

Now, let U be the set of unbounded variables $\{y_1, \ldots, y_m\}$, such that $y_m \prec \ldots \prec y_1$. Since Resolve-Cooper considers these variables in an ordered fashion, all possible resolvents can be defined by saturation, using the following sequence

$$S_0 = C \qquad S_{i+1} = S_i \cup \{R \mid R \text{ is a resolvent for a core } D \subseteq S_i \text{ for variable } y_{i+1}\}$$

The final set of all possible resolvents will be saturated$(C) = S_{m+1}$. Since Resolve-Cooper can be applied at most once for a core D, and there are a finite number of cores D in each S_i, it follows that the Resolve-Cooper rule can be applied only a finite number of times. □

Now we are ready to present and prove a simple and flexible strategy that will guarantee termination of our procedure even in the unbounded case.

Definition 1.5 (Two-layered strategy). We say a strategy is two-layered for an initial state $\langle [\![], C_0 \rangle$ if

1. it is reasonable (i.e., gives preference to the Propagate-Simple rules);
2. the Propagate rules are limited to δ-relevant bound refinements;
3. the Forget rule is never used to eliminate resolvents introduced by Resolvent-Cooper;
4. it only applies the Conflict rule if Resolve-Cooper is not applicable.

Theorem 1.6 (Termination). *Given a set of constraints C, there is no infinite derivation sequence starting from $S_0 = \langle [\![], C \rangle$ that uses a two-layered strategy when U contains all unbounded variables in C.*

Proof. First we note that, if Conflict rule applies to a non-U-constraint, it must be that Resolve-Cooper is not applicable. Since the strategy prefers Resolve-Cooper this, in effect, splits the procedure into two layers, one dealing with bounded variables, and the other one dealing with the unbounded variables using the strong resolution. And, since the Learn rule will therefore only be able to learn constraint over bounded variables, we will never apply Resolve-Cooper to cores involving those constraints.

The strategy also dictates that we don't remove the strong resolvents introduced by Resolve-Cooper so we know, by Lemma 1.5, that in any derivation sequence

$$T = \langle [\![], C_0 \rangle \Longrightarrow \langle M_1, C_1 \rangle \Longrightarrow \cdots \Longrightarrow \langle M_n, C_n \rangle \Longrightarrow \cdots$$

produced by a two-layered strategy, the Resolve-Cooper rule can only be applied a finite number of times. Consequently, the number of fresh variables introduced in T is bounded.

Then, there must be a state $S_n = \langle M_n, C_n \rangle$ in T such that the Resolve-Cooper rule is not applicable to any state that is reachable from S_n. Therefore no additional fresh variable is created after S_n.

Now, assume that the derivation sequence T is infinite. Then, since the propagation step is limited to the δ-relevant ones, it must be that, after S_n, the Conflict rule is being applied infinitely often. Moreover, since Resolve-Cooper does not apply after the state S_n, it must be that the Conflict rule is applied to only non-U-constraints. But we know, by Theorem 1.3, that if all variables are bounded, this can not happen. □

As an improvement, we note that we do not need to fix the ordering $\prec$ at the beginning. It *can* be modified but, in this case, termination is only guaranteed if we eventually stop modifying it. Moreover, we can start applying the strategy with $U = \varnothing$. Then, for any non-δ-relevant bound refinement $\gamma(x)$, produced by the Backjump rules, we add x to the set U. Moreover, a variable x can be removed from U whenever a lower and upper bound for x can be deduced, and they do not depend on any decided bounds (variable becomes bounded).

1.5 Experimental Evaluation

We implemented the procedure described in a new solver cutsat. The implementation is a straightforward translation of the presented ideas, with very limited propagation, but includes heuristics from the SAT community such as dynamic ordering based on conflict activity, and Luby restarts. When a variable is to be decided, and we have an option to choose between the upper and lower bound, we choose the value that could satisfy the most constraints. The solver source code, binaries used in the experiments, and all the accompanying materials are available at the authors website[2].

In order to evaluate our procedure we took a variety of already available integer problems from the literature, but we also crafted some additional ones. We included the problems that were used in [39] to evaluate their new simplex-based procedure that incorporates a new way of generating cuts to eliminate rational solutions. These problems are generated randomly, with all variables unbounded. This set of problems, which we denote with dillig, was reported hard for modern SMT

[2] `http://cs.nyu.edu/~dejan/cutsat/`

solvers. We also included a reformulation of these problems, so that all the variables are bounded, by introducing slack variables, which we denote as slack. Next, we included the pure integer problems from the MIPLIB 2003 library [2], and we denoted this problem set as miplib2003. The original problems are all very hard optimization instances, but, since we are dealing with the decision problem only, we have removed the optimization constraints and turned them into feasibility problems.[3] We included PB problems from the 2010 pseudo-Boolean competition that were submitted and selected in 2010, marked as pb2010, and problems encoding the pigeonhole principle using cardinality constraints, denoted as pigeons. The pigeonhole problems are known to have no polynomial Boolean resolution proofs, and will therefore be hard for any resolution solver that does not use cutting planes. And finally, we included a group of crafted benchmarks encoding a tight n-dimensional cone around the point whose coordinates are the first n prime numbers, denoted as primes. In these benchmarks all the variables are bounded from below by 0. We included the satisfiable versions, and the unsatisfiable versions which exclude points smaller than the prime solution.

In order to compare to the state-of-the art we compared to three different types of solvers. We compared to the current best integer SMT solvers, i.e yices 1.0.29 [41], z3 2.15 [35], mathsat5 [51] and mathsat5+cfp that simulates the algorithm from [39]. On all 0-1 problems in our benchmark suite, we also compared to the sat4j [11] PB solver, one of the top solvers from the PB competition, and a version sat4j+cp that is based on cutting planes. Finally, we compared with the two top commercial MIP solvers, namely, gurobi 4.0.1 and cplex 12.2, and the open source MIP solver glpk 4.38. The MIP solvers have largely been ignored in the theorem-proving community, as it is claimed that, due to the use of floating point arithmetic, they are not sound.

All tests were conducted on an Intel Pentium E2220 2.4 GHz processor, with individual runs limited to 2GB of memory and 600 seconds. The results of our experimental evaluation are presented in Table 2.1. The rows are associated with the individual solvers, and columns separate the problem sets. For each problem set we write the number of problems that the solver managed to solve within 600 seconds, and the cumulative time for the solved problems. We mark with bold the results that are best in a group of solvers, and we underline the results that are best among all solvers. For the better understanding of the comparison of cutsat with individual

[3] All of the problems have a significant Boolean part, and 13 (out of 16) problems are pure PB problems

Table 1.1: Experimental results.

problems	miplib2003 (16)		pb2010 (81)		dillig (250)		slacks (250)		pigeons (19)		primes (37)	
cutsat	722.78	12	1322.61	46	4012.65	223	2722.19	152	0.15	19	5.08	37
smt solvers	time(s)	solved	time(s)	solved	time(s)	solved	time(s)	solved	time(s)	solved	time(s)	solved
mathsat5+cfp	575.20	11	2295.60	33	**2357.18**	**250**	160.67	98	0.23	19	1.26	37
mathsat5	**89.49**	**11**	1224.91	38	3053.19	245	**3243.77**	**177**	0.30	19	**1.03**	**37**
yices	226.23	8	57.12	37	5707.46	159	7125.60	134	**0.07**	**19**	0.64	32
z3	532.09	9	**168.04**	**38**	885.66	171	589.30	115	0.27	19	11.19	23
pb solvers												
sat4j	**22.34**	**10**	**798.38**	**67**	0.00	0	0.00	0	110.81	8	0.00	0
sat4j+cp	28.56	10	349.15	60	0.00	0	0.00	0	**4.85**	**19**	0.00	0
mip solvers												
glpk	242.67	12	1866.52	46	4.50	248	0.08	10	**0.09**	**19**	**0.44**	**37**
cplex	53.86	15	1512.36	58	8.65	250	8.76	248	0.51	19	3.47	37
gurobi	**28.96**	**15**	**1332.53**	**58**	**5.48**	**250**	**8.12**	**248**	0.21	19	0.80	37

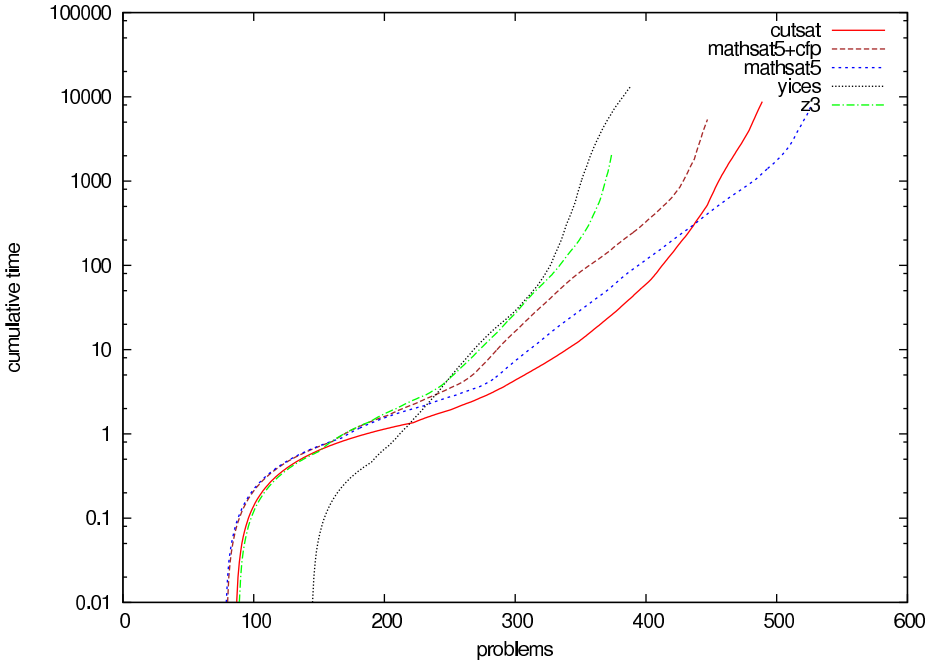


Figure 1.2: Comparison of cutsat with other SMT solvers. The plot presents the number of problems solved against the cumulative time (logarithmic time scale).

SMT solvers we present cumulative solving times in Figure 1.2.

Compared to the SMT solvers, cutsat performs surprisingly strong, given that it is a prototype implementation. It outperforms or is the same as other smt solvers, except mathsat5 on all problem sets. Most importantly, it outperforms even mathsat5 on the real-world miplib2003 and pb2010 problem sets. The random dillig problems seem to be attainable by the solvers that implement the procedure from [39], but the same solvers surprisingly fail to solve the same problems with the slack reformulation (slacks).

Also, very noticeably, the commercial MIP solvers outperform all the SMT solvers and cutsat by a big margin.

2
Non-Linear Arithmetic

From the early beginnings in Persian and Chinese mathematics [54, 55, 101] until the present day, polynomial constraints and the algorithmic ways of solving them have been one of the driving forces in the development of mathematics. Though studied for centuries due to the natural elegance they provide in modeling the real world, from resolving simple taxation arguments to modeling planes and hybrid systems, we are still lacking a practical algorithm for solving a system of polynomial constraints. Throughout the history of mathematics, many brilliant minds have studied and algorithmically solved many of the related problems, such as root finding [37, 98, 89] and factorization of polynomials [71, 46, 47]. But, it was not until Alfred Tarski [90, 91, 97] showed that the theory of real closed fields admits elimination of quantifiers when it became clear that a general decision procedure for solving polynomial constraints was possible. Although certainly a wonderful theoretical result of landmark importance, with its non-elementary complexity, Tarski's procedure was unfortunately totally impractical.

As one would expect, Tarski's procedure consequently has been much improved. Most notably, Collins [26] gave the first relatively effective method of quantifier elimination by cylindrical algebraic decomposition (CAD). The CAD procedure itself has gone through many revisions [65, 52, 27, 15]. However, even with the improvements and various heuristics, its doubly-exponential worst-case behavior has remained as a serious impediment. The CAD algorithm works by decomposing $\mathbb{R}^k$ into connected components such that, in each cell, all of the polynomials from the problem are sign-invariant. To be able to perform such a particular decomposition, CAD first performs a *projection* of the polynomials from the initial problem. This projection includes many new polynomials, derived from the initial ones, and these polynomials carry enough information to ensure that the decomposition is indeed possible. Unfortunately, the size of these projection sets grows exponentially in the number of variables, causing the projection phase, and its consequent impact on the search space, to be a key hurdle to CAD scalability.

We propose a new decision procedure for the existential theory of the reals that tries to alleviate the above problem. As in [58, 67, 60], the new procedure performs a backtracking search for a model in $\mathbb{R}$, where the backtracking is powered by a novel conflict resolution procedure. Our

approach takes advantage of the fact that each conflict encountered during the search is based on the current assignment and generally involves only a few constraints, a *conflicting core*. When in conflict, we project only the polynomials from the conflicting core and explain the conflict in terms of the current model. This means that we use projection conservatively, only for the subsets of polynomials that are involved in the conflict, and even then we reduce it further. As another advantage, the conflict resolution provides the usual benefits of a Conflict-Driven Clause Learning (CDCL)-style [86, 69] search engine, such as non-chronological backtracking and the ability to ignore irrelevant parts of the search space. The projection operators we use as part of the conflict resolution need not be CAD-based and, in fact, one can easily adapt projections based on other algorithms (e.g [66, 9]).

Due to the volume of algorithms and concepts involved, we concentrate on the details of the decision procedure and refer the reader to the existing literature for further information [21, 22, 25, 59, 9]. Acknowledging the importance that the details of a particular implementation play, we include the description of particular algorithms we chose for our implementation in Appendix A.

2.1 Preliminaries

As usual, we denote the ring of integers with $\mathbb{Z}$, the field of rational numbers with $\mathbb{Q}$, and the field of real numbers with $\mathbb{R}$. Additionally, given a vector of variables $\vec{x}$ we denote the set of polynomials with integer (rational, real) coefficients and variables $\vec{x}$ as $\mathbb{Z}[\vec{x}]$ ($\mathbb{Q}[\vec{x}]$, $\mathbb{R}[\vec{x}]$). Unless stated otherwise, we assume all polynomials take integer coefficients, i.e. a polynomial $f \in \mathbb{Z}[\vec{y}, x]$ is of the form

$$f(\vec{y}, x) = a_m \cdot x^{d_m} + a_{m-1} \cdot x^{d_{m-1}} + \cdots + a_1 \cdot x^{d_1} + a_0 \ ,$$

where $0 < d_1 < \cdots < d_m$, and the coefficients a_i are polynomials in $\mathbb{Z}[\vec{y}]$ with $a_m \neq 0$. We call x the *top variable*. The highest power d_m is the *degree* of the polynomial f in variable x, and we denote it with $\mathsf{deg}(f, x)$. The set of coefficients of f is denoted as

$$\mathsf{coeff}(f, x) = \{a_m, \ldots, a_0\} \ .$$

We call a_m the *leading coefficient* in variable x, and denote it with $\mathsf{lc}(f, x)$. If we exclude the first k terms of the polynomial f, we obtain the polynomial

$$\mathsf{R}_k(f, x) = a_{m-k}x^{d_{m-k}} + \cdots + a_0 \ ,$$

called the k-th *reductum* of f. We write $\mathsf{R}^*(f, x)$ for the set $\{\mathsf{R}_0(f, x), \ldots, \mathsf{R}_m(f, x)\}$ containing all reductums.

We denote the set of variables appearing in a polynomial f as $\mathsf{vars}(f)$ and call the polynomial *univariate* if $\mathsf{vars}(f) = \{x\}$ for some variable x. Otherwise the polynomial is *multivariate*, or a constant polynomial (if it contains no variables). Given a set of polynomials $A \subset \mathbb{Z}[x_1, \ldots x_n]$, we denote with A_k the subset of polynomials in A that belong to $\mathbb{Z}[x_1, \ldots, x_k]$[1], or more precisely

$$A_k = A \cap \mathbb{Z}[x_1, \ldots, x_k] \ .$$

A number $\alpha \in \mathbb{R}$ is a *root of the polynomial* $f \in \mathbb{Z}[x]$ iff $f(\alpha) = 0$. We call a real number $\alpha \in \mathbb{R}$ *algebraic* iff it is a root of a univariate polynomial $f \in \mathbb{Z}[x]$, and we denote the field of all real algebraic numbers by $\mathbb{R}_{\mathsf{alg}}$. We can represent any algebraic number α as $(l, u)_f$, with $l, u \in \mathbb{Q}$, where α is a root of a polynomial f, and the only root in the interval (l, u). We denote the number of distinct real roots of a univariate polynomial f as $\mathsf{rootcount}(f)$.

Example 2.1. Consider the univariate polynomial $f_1 = 16x^3 - 8x^2 + x + 16$. This polynomial has only one real root, the irrational number $\alpha_1 \approx -0.840661$ and we can represent it as $(-0.9, -0.8)_{f_1}$.

Given a set of variables $X = \{x_1, \ldots, x_n\}$, we call υ a *variable assignment* if it maps each variable x_k to a real algebraic number $\upsilon(x_k)$, the value of x_k under υ. We overload υ, as usual, to obtain the value of a polynomial $f \in \mathbb{Z}[x_1, \ldots, x_n]$ under υ and write it as $\upsilon(f)$. We say that a polynomial f *vanishes* under υ if $\upsilon(f) = 0$. We can update the assignment υ to map a variable x_k to the value α, and we denote this as $\upsilon[x_k \mapsto \alpha]$. Under a variable assignment υ that interprets the variables $\vec{y}$, some coefficients of a polynomial $f(\vec{y}, x)$ may vanish. If a_k is the

[1] We thus have $A_0 \subseteq A_1 \subseteq \cdots \subseteq A_n$, with A_0 being the constant polynomials of A, and $A_n = A$.

first non-vanishing coefficient of f, i.e., $\upsilon(a_k) \neq 0$, we write $\mathsf{R}(f, x, \upsilon) = a_k x^{d_k} + \cdots + a_0$ for the *reductum of f with respect to υ* (the non-vanishing part). Given any sequence of polynomials $\vec{f} = (f_1, \ldots, f_s)$ and a variable assignment υ we define the *vanishing signature* of $\vec{f}$ as the sequence $\mathsf{v\text{-}sig}(\vec{f}, \upsilon) = (f_1, \ldots, f_k)$, where $k \leq s$ is the minimal number such that $\upsilon(f_k) \neq 0$, or s if they all vanish. For the polynomial $f(\vec{y}, x)$ as above, we define the *vanishing coefficients signature* as $\mathsf{v\text{-}coeff}(f, x, \upsilon) = \mathsf{v\text{-}sig}(a_m, \ldots, a_0, \upsilon)$.

Example 2.2. Consider the polynomial $f \in \mathbb{Z}[x, y]$

$$f = 2(x-1)y^3 + (x^2-1)y^2 + 2xy + y \ .$$

Under a variable assignment υ with $\upsilon(x) = 1$ the leading coefficients $2(x-1)$ and (x^2-1) evaluate to 0 (vanish), and we therefore have that the reductum and the vanishing coefficient signature of f with respect to υ are

$$\mathsf{R}(f, y, \upsilon) = 2xy + y \ , \qquad \mathsf{v\text{-}coeff}(f, y, \upsilon) = (2(x-1),\ x^2-1,\ 2x) \ .$$

A *basic polynomial constraint* F is a constraint of the form $f \triangledown 0$ where f is a polynomial and $\triangledown \in \{<, \leq, =, \neq, \geq, >\}$. We denote the polynomial constraint that represents the *negation* of a constraint F with $\neg F$.[2] In order to identify the polynomial f of the constraint F, and the variables of F, we write $\mathsf{poly}(F)$ and $\mathsf{vars}(F)$, respectively. We normalize all constraints over constant polynomials to the dedicated constants true and false with the usual semantics. We write $\upsilon(F)$ to denote the evaluation of F under υ, which is the constraint $\upsilon(f) \triangledown 0$. If f does not evaluate to a constant under υ, then $\upsilon(F)$ evaluates to a new polynomial constraint F', where $\mathsf{poly}(F')$ can contain algebraic coefficients.

To be able to denote roots of multivariate polynomials we define the notion of a *root object*. Given a variable x, a polynomial in $f \in \mathbb{Z}[\vec{y}, x]$, and a root index $k > 0$, we denote the root object as $\mathsf{root}(f, k, x)$. The root object denotes the k-th root of f in variable x, but this only makes sense when the variables $\vec{y}$ have assigned values. Given a variable assignment υ that interprets

[2] For example $\neg(x^2 + 1 > 0) \equiv x^2 + 1 \leq 0$.

the variables $\vec{y}$, the semantics of the root object are as follows. Let g be the univariate polynomial obtained from f by substituting the variables $\vec{y}$ by their values in υ. If g has at least k distinct real roots $\alpha_1 < \cdots < \alpha_n$, we define $\upsilon(\mathsf{root}(f, k, x)) = \alpha_k$. Otherwise, the value of the root object is undefined. To denote the number of roots k that the polynomial f has under the assignment υ (that excludes x as above), we write $\mathsf{rootcount}(f, x, \upsilon)$.

Borrowing from the extended Tarski language [14, Chapter 7], in addition to the basic constraints, we will also be working with *extended polynomial constraints* that include the above-defined root objects. An extended polynomial constraint F is of the form

$$x \bigtriangledown_r \mathsf{root}(f, k, x) \ , \tag{2.1}$$

where $\bigtriangledown_r \in \{<_r, \leq_r, =_r, \neq_r, \geq_r, >_r\}$. In order to allow extraction of the polynomial f from the constraint, we define $\mathsf{poly}(F) = f$. The semantics of the predicate (2.1) under a variable assignment υ is the following. If the value of the root object under υ is defined, then the value of the constraint is as expected. Otherwise, the constraint evaluates to false. Naturally, if F is an extended polynomial constraint, so is the negation $\neg F$.[3]

Example 2.3. Take the bivariate polynomial $f = 2yx^3 - 8x^2 + x + 3y - 8$ and the variable assignment $\upsilon_1 = [x \mapsto -1, y \mapsto 8]$. If we substitute the value of y into the polynomial f, we obtain the univariate polynomial $g = 16x^3 - 8x^2 + x + 16$ which, as we saw in Example 2.1, has only one root $\alpha = (-0.9, -0.8)_g$. Under υ_1, the value of $\mathsf{root}(f, 1, x)$ evaluates to α, but the value of $\mathsf{root}(f, 2, x)$ is undefined as g only has one root. Now consider the constraints

$$x <_r \mathsf{root}(f, 1, x), \quad x \geq_r \mathsf{root}(f, 1, x), \quad \neg(x <_r \mathsf{root}(f, 1, x)), \quad \neg(x <_r \mathsf{root}(f, 2, x)).$$

By definition the values of the constraints under υ_1 are true, false, false, true, correspondingly. The fourth constraint evaluates to true as the root object is undefined.

Now consider the assignment $\upsilon_2 = [x \mapsto -1, y \mapsto 0]$. Under this assignment, substitution of the value of y into f gives the polynomial $g = -8x^2 + x - 8$, which now does not have any real roots. The values of the above constraints under the assignment υ_2 are therefore false,

[3] Note that, for example, $\neg(x <_r \mathsf{root}(f, k, x))$ is not necessarily equivalent to $x \geq_r \mathsf{root}(f, k, x)$.

false, true, true. Note that the second and third constraint might be mistaken to be equal, but are not – as can be seen, they have different semantics.

A *polynomial constraint* is either a basic or an extended one. Given a set of polynomial constraints $\mathcal{F}$, we say that the variable assignment υ satisfies $\mathcal{F}$ if it satisfies each constraint in $\mathcal{F}$. If there is such a variable assignment, we say that $\mathcal{F}$ is *satisfiable*, otherwise it is *unsatisfiable*. A *clause* of polynomial constraints is a disjunction $C = F_1 \vee \ldots \vee F_n$ of polynomial constraints. We use $\mathsf{literals}(C)$ to denote the set $\{F_1, \neg F_1, \ldots, F_n, \neg F_n\}$. We say that the clause C is satisfied under the assignment υ if some polynomial constraint $F_j \in C$ evaluates to true under υ. Finally, a *polynomial constraint problem* is a set of clauses $\mathcal{C}$, and it is satisfiable if there is a variable assignment υ that satisfies all the clauses in $\mathcal{C}$. If the clauses of $\mathcal{C}$ contain the variables $x_1, \ldots, x_n$ then, for $k \leq n$, we denote with $\mathcal{C}_k$ the subset of the clauses that only contain variables $x_1, \ldots, x_k$.

2.2 An Abstract Decision Procedure

We describe our procedure as an abstract transition system in the spirit of Abstract DPLL [72, 62]. The crucial difference between the system we present is that we depart from viewing the Boolean search engine and the theory reasoning as two separate entities that communicate only through existing literals. Instead, we allow the model that the theory is trying to construct to be involved in the search and in explaining the conflicts, while allowing new literals to be introduced so as to support more complex conflict analyses. Additionally, our presentation makes the concept of relevancy inherent to the procedure (e.g. [33]). The transition system presented here applies to non-linear arithmetic, but it can in general be applied to other theories.

The states in the transition system are indexed pairs of the form $\langle M, \mathcal{C} \rangle_n$, where M is a sequence (usually called a *trail*) of *trail elements*, and $\mathcal{C}$ is a set of clauses. The index n denotes the current *stage* of the state. Trail elements can be decided literals, propagated literals, or a variable assignment. A *decided literal* is a polynomial constraint F that we assume to be true. On the other hand, a *propagated literal*, denoted as $E \rightarrow F$, marks a polynomial constraint $F \in E$ that is implied to be true in the current state by the clause E (the *explanation*). In both cases,

we say that the constraint F appears in M, and write this as $F \in M$. We denote the set of polynomial constraints appearing in M with $\mathsf{constraints}(M)$. We say M is *non-redundant* if no polynomial constraint appears in M more than once. A *trail variable assignment*, written as $x \mapsto \alpha$, is an assignment of a single variable to a value $\alpha \in \mathbb{R}_{\mathsf{alg}}$. Given a trail M, containing variable assignments $x_{i_1} \mapsto \alpha_1, \ldots, x_{i_k} \mapsto \alpha_k$, in order, we can construct an assignment

$$\upsilon[M] = \upsilon_0[x_{i_1} \mapsto \alpha_1] \ \ldots \ [x_{i_k} \mapsto \alpha_k] \ ,$$

where υ_0 is an empty assignment that does not assign any variables.

We say that the sequence M is *stage increasing* when the sequence is of the form

$$M = [\![N_1, x_1 \mapsto \alpha_1, \ldots, x_{k-1} \mapsto \alpha_{k-1}, N_k, x_k \mapsto \alpha_k, \ldots, x_{n-1} \mapsto \alpha_{n-1}, N_n]\!] \ ,$$

where, for each $k \leq n$, the sequence N_k does not contain any variable assignments, each constraint $F \in \mathsf{constraints}(N_k)$ contains the variable x_k, and (optionally) the variables $x_1, \ldots, x_{k-1}$. In such a sequence M, we denote with $\mathsf{stage}(M) = n$ the *stage* of the sequence. If $\mathcal{F} = \mathsf{constraints}(M)$, we say that M is *feasible*, when the set of univariate polynomial constraints $\upsilon[M](\mathcal{F})$ has a solution. We write $\mathsf{feasible}(M)$ to denote the feasible set of $\upsilon[M](\mathcal{F})$. Given an additional polynomial constraint $F \in \mathbb{Z}[x_1, \ldots, x_n]$, we say that F is *compatible* with the sequence M, when $\mathsf{feasible}([\![M, F]\!]) \neq \varnothing$ and denote this with a predicate $\mathsf{compatible}(F, M)$. In our actual implementation, we represent feasible sets using a set of intervals with real algebraic endpoints. The predicate $\mathsf{compatible}(F, M)$ is implemented using real root isolation and sign evaluation procedures. In the Appendix, we sketch the algorithms used to implement these procedures, and provide references to the relevant literature.

Our transition system will work over states that are *well-formed*. Intuitively, in such a state, we commit to the variable assignment, but make sure that the current stage is consistent on the Boolean level. With this in mind, given a polynomial constraint F with $\mathsf{vars}(F) \subseteq \{x_1, \ldots, x_n\}$,

and a state M with $\mathsf{stage}(M) = n$, we define the *state value* of F in M as

$$\mathsf{value}(F, M) = \begin{cases} \upsilon[M](F) & x_n \notin \mathsf{vars}(F) \ , \\ \mathsf{true} & F \in \mathsf{constraints}(M) \ , \\ \mathsf{false} & \neg F \in \mathsf{constraints}(M) \ , \\ \mathsf{undef} & \text{otherwise.} \end{cases}$$

Naturally, we overload value to also evaluate clauses of polynomial constraints, and sets of clauses, i.e. for a clause C we define $\mathsf{value}(C, M)$ to be true, if any of the literals evaluates to true, false if all literals evaluate to false, and undef otherwise.

Definition 2.1 (Well-Formed State). We say a state $\langle M, \mathcal{C} \rangle_n$ is well-formed when M is non-redundant, stage increasing with $\mathsf{stage}(M) = n$, and all of the following hold.

1. Clauses up to stage n are satisfied, i.e. we have that $\mathsf{value}(\mathcal{C}_{n-1}, M) = \mathsf{true}$.
2. The state is consistent, i.e. $\mathsf{feasible}(M) \neq \varnothing$ and for each $F \in \mathsf{constraints}(M)$ we have that that $\mathsf{value}(F, M) = \mathsf{true}$.
3. Propagated literals $E \rightarrow F$ are implied, i.e. $F \in E$ and for all other literals $F' \neq F$ in E, $\mathsf{value}(F', M) = \mathsf{false}$.

We are now ready to define the transition system. We separate the transition rules into three groups: the search rules, the clause processing rules, and the conflict analysis rules. The *search rules* are the main driver of the procedure, with the responsibility for selecting clauses to process, creating the variable assignment while lifting the stages, and detecting Boolean conflicts. The search rules operate on well-formed states $\langle M, \mathcal{C} \rangle_n$. If the search rules select a clause C to process, we switch to a state $\langle M, \mathcal{C} \rangle_n \vDash C$, where we can apply the set of *clause processing rules*. The notation $\vDash C$ designates that we are performing semantic reasoning in order to assign a value to a literal of C. If the search rules detect that in the current state some clause $C \in \mathcal{C}$ is falsified, we switch to a state $\langle M, \mathcal{C} \rangle_n \vdash C$, where we can apply the *conflict analysis rules*. The notation $\vdash C$ denotes that we are trying to produce a proof of why C is inconsistent in the current state.

Finally, given a polynomial constraint problem $\mathcal{C}$, with $\mathsf{vars}(\mathcal{C}) = \{x_1, \ldots, x_n\}$, the overall goal of the procedure is, starting from an initial state $\langle [\![]\!], \mathcal{C} \rangle_1$, and applying the rules, either to

end up either in a state $\langle v, \mathsf{sat} \rangle$, indicating that the initial set of clauses $\mathcal{C}$ is satisfiable where the assignment v is the witness, or to derive unsat, which indicates that the set $\mathcal{C}$ unsatisfiable.

Search Rules. Fig 2.1 presents the set of search rules. The SELECT-CLAUSE rule selects one of the clauses of the current stage, whose state value is still undetermined, and transitions into the clause processing mode that will hopefully satisfy the clause. The CONFLICT rule detects if there is a clause of the current stage that is inconsistent in the current state, and transitions into the conflict resolution mode that will explain the conflict and backtrack appropriately. On the other hand, if all the clauses of the current stage are satisfied, we can either transition to the next stage, using the LIFT-STAGE rule, or conclude that our problem is satisfiable, using the SAT rule. Since at this point the current stage is consistent, in addition to formally introducing the new stage, the LIFT-STAGE rule selects a particular value for the current variable from the feasible set of the current stage. Note that once we move to the next stage, all the clauses of previous stages have values in the state, and can never be selected by the SELECT-CLAUSE or the CONFLICT rules. We conclude this set of rules with the FORGET rule that can be used to eliminate any learnt clause (a clause added while analyzing conflicts) from the current set of clauses.

Clause Processing Rules. In this set of rules, presented in Fig 2.2, we are trying to assign a currently unassigned literal of the given clause C, hoping to satisfy the clause. When one of the clause processing rules is applied, we immediately switch back to the search rules. As usual in a CDCL-style procedure, the simplest way to satisfy the clause C is to perform the Boolean unit propagation, if applicable, by using the $\mathbb{B}$-PROPAGATE rule. We restrict the application of this rule so that adding the constraint to the state keeps it consistent, i.e., it is $\mathsf{compatible}$ with the current set of constraints. If this is the case, we add the constraint to the state together with the explanation (clause C itself). To allow more complex propagations, the ones that are valid in $\mathbb{R}$ modulo the current state, we provide the $\mathbb{R}$-PROPAGATE rule. This rule can propagate a constraint from the clause, if assuming the negation would be incompatible with the current state. The $\mathbb{R}$-PROPAGATE rule is equipped with an explanation function $\mathsf{explain}$. The $\mathsf{explain}$ function, given a polynomial constraint F, and the trail M, returns the explanation clause

SELECT-CLAUSE				
$\langle M, \mathcal{C}\rangle_k$	$\longrightarrow$	$\langle M, \mathcal{C}\rangle_k \vDash C$	**if**	$C \in \mathcal{C}_k$ $\mathsf{value}(C, M) = \mathsf{undef}$
CONFLICT				
$\langle M, \mathcal{C}\rangle_k$	$\longrightarrow$	$\langle M, \mathcal{C}\rangle_k \vdash C$	**if**	$C \in \mathcal{C}_k$ $\mathsf{value}(C, M) = \mathsf{false}$
SAT				
$\langle M, \mathcal{C}\rangle_k$	$\longrightarrow$	$\langle \upsilon[M], \mathsf{sat}\rangle$	**if**	$x_k \notin \mathsf{vars}(\mathcal{C})$
LIFT-STAGE				
$\langle M, \mathcal{C}\rangle_k$	$\longrightarrow$	$\langle [\![M, x_k \mapsto \alpha]\!], \mathcal{C}\rangle_{k+1}$	**if**	$x_k \in \mathsf{vars}(\mathcal{C})$ $\alpha \in \mathsf{feasible}(M)$ $\mathsf{value}(\mathcal{C}_k, M) = \mathsf{true}$
FORGET				
$\langle M, \mathcal{C}\rangle_k$	$\longrightarrow$	$\langle M, \mathcal{C} \setminus \{C\}\rangle_k$	**if**	$C \in \mathcal{C}$ C is a learnt clause

Figure 2.1: The search rules.

$E = \mathsf{explain}(F, M)$ that is valid in $\mathbb{R}$, and implies the constraint F under the current assignment i.e., $F \in E$, and all literals in E but F are false in the state. The clause E may contain new literals that do not occur in $\mathcal{C}$, as long as they evaluate to false in the state. To simplify the presentation, in the $\mathbb{R}$-PROPAGATE rule, the explanation clause E is eagerly generated, but in our actual implementation, we compute them only if they are needed during conflict resolution. Finally, if we cannot deduce the value of an unassigned literal, we can assume a value for such a literal using the DECIDE-LITERAL rule.

Conflict analysis rules. The conflict analysis rules start from an initial proper state $\langle M, \mathcal{C}\rangle_n \vdash C$, where $C \in \mathcal{C}$ is the conflicting clause. The conflict analysis is a standard Boolean conflict analysis [86] with a model-based twist. As the rules move the state backwards, the goal is to construct a new resolvent clause R, that will explain the conflict and ensure progress in the

DECIDE-LITERAL				
$\langle M, \mathcal{C} \rangle_k \vDash C$	$\longrightarrow$	$\langle [\![M, F_1]\!], \mathcal{C} \rangle_k$	**if**	$F_1, F_2 \in C$ $\forall i : \mathsf{value}(F_i, M) = \mathsf{undef}$ $\mathsf{compatible}(F_1, M)$
$\mathbb{B}$-PROPAGATE				
$\langle M, \mathcal{C} \rangle_k \vDash C$	$\longrightarrow$	$\langle [\![M, C \rightarrow F]\!], \mathcal{C} \rangle_k$	**if**	$C = F_1 \vee \ldots \vee F_m \vee F$ $\mathsf{value}(F, M) = \mathsf{undef}$ $\forall i : \mathsf{value}(F_i, M) = \mathsf{false}$ $\mathsf{compatible}(F, M)$
$\mathbb{R}$-PROPAGATE				
$\langle M, \mathcal{C} \rangle_k \vDash C$	$\longrightarrow$	$\langle [\![M, E \rightarrow F]\!], \mathcal{C} \rangle_k$	**if**	$F \in \mathsf{literals}(C)$ $\mathsf{value}(F, M) = \mathsf{undef}$ $\neg\, \mathsf{compatible}(\neg F, M)$ $E = \mathsf{explain}(F, M)$

Figure 2.2: The clause satisfaction rules.

search. This means that, when we backtrack the sequence M just enough, the addition of R will ensure progress in the search by eliminating the inconsistent part from the state, and thus forcing the search rules to change some of the choices made. On the other hand, if the conflict analysis backtracks the state all the way into an empty state, this will be a signal that the original problem is unsatisfiable. Once the conflict analysis backtracks enough and deduces the resolvent R, then we pass it to the clause processing immediately.[4]

Example 2.4. First, for the sake of this example, let us restrict ourselves to the case of linear constraints. When solving a set of linear constraints $\mathcal{C}$, one can use the Fourier-Motzkin elimination rule to define the explain function. As shown in [67, 60], this will give a finite basis $\mathcal{B}$ with respect to $\mathcal{C}$ that is obtained by closing $\mathcal{C}$ under the application of the Fourier-

[4]This is crucial in order to ensure termination.

RESOLVE-PROPAGATION

$$\langle [\![M, E{\rightarrow}F]\!], \mathcal{C} \rangle_k \vdash C \longrightarrow \langle M, \mathcal{C} \rangle_k \vdash R \quad \textbf{if} \quad \neg F \in C,\ R = \mathsf{resolve}(C, E, F)$$

▷ resolve returns the standard Boolean resolvent

RESOLVE-DECISION

$$\langle [\![M, F]\!], \mathcal{C} \rangle_k \vdash C \longrightarrow \langle M, \mathcal{C} \cup \{C\} \rangle_k \vDash C \quad \textbf{if} \quad \neg F \in C$$

CONSUME

$$\langle [\![M, F]\!], \mathcal{C} \rangle_k \vdash C \longrightarrow \langle M, \mathcal{C} \rangle_k \vdash C \quad \textbf{if} \quad \neg F \notin C$$

$$\langle [\![M, E{\rightarrow}F]\!], \mathcal{C} \rangle_k \vdash C \longrightarrow \langle M, \mathcal{C} \rangle_k \vdash C \quad \textbf{if} \quad \neg F \notin C$$

DROP-STAGE

$$\langle [\![M, x_{k+1}{\mapsto}\alpha]\!], \mathcal{C} \rangle_{k+1} \vdash C \longrightarrow \langle M, \mathcal{C} \rangle_k \vdash C \quad \textbf{if} \quad \mathsf{value}(C, M) = \mathsf{false}$$

$$\langle [\![M, x_{k+1}{\mapsto}\alpha]\!], \mathcal{C} \rangle_{k+1} \vdash C \longrightarrow \langle M, \mathcal{C} \cup \{C\} \rangle_k \vDash C \quad \textbf{if} \quad \mathsf{value}(C, M) = \mathsf{undef}$$

UNSAT

$$\langle [\![ ]\!], \mathcal{C} \rangle_1 \vdash C \longrightarrow \mathsf{unsat}$$

Figure 2.3: The conflict analysis rules.

Motzkin elimination step. It is fairly easy to show that the closure is a finite set, since we always produce constraints with one fewer variable.

We explain the search rules by applying them to the following set of linear polynomial constraints

$$\mathcal{C} = \{\ (\underbrace{x+1 \leq 0 \vee x-1 \geq 0}_{C_1}),\ \underbrace{x+y>0}_{C_2},\ \underbrace{x-y>0}_{C_3}\ \}\ .$$

During the search, we associate x with level 1, and y with level 2. Therefore the constraints of level 1 are $\mathcal{C}_1 = \{C_1\}$, and the constraints of level 2 are $\mathcal{C}_2 = \{C_2, C_3\}$. The following is a derivation of the transition system, starting from the initial state $\langle [\![]\!], \mathcal{C}\rangle$, applying the rules until we encounter a conflict.

$\langle [\![]\!], \mathcal{C}\rangle_1$

$\downarrow$ Select-Clause $(x+1 \leq 0) \vee (x-1 \geq 0)$, Decide-Literal $(x+1 \leq 0)$

$\langle [\![(x+1 \leq 0)]\!], \mathcal{C}\rangle_1$

$\downarrow$ Lift-Stage

$\langle [\![(x+1 \leq 0), x \mapsto -1]\!], \mathcal{C}\rangle_2$

$\downarrow$ Select-Clause $(x+y>0)$, $\mathbb{B}$-Propagate $(x+y>0)$

$\langle [\![(x+1 \leq 0), x \mapsto -1, C_2 \rightarrow (x+y>0)]\!], \mathcal{C}\rangle_2$

Select-Clause $(x-y>0)$

$\langle \underbrace{[\![(x+1 \leq 0), x \mapsto -1, C_2 \rightarrow (x+y>0)]\!]}_{M_1}, \mathcal{C}\rangle_2 \vDash (x-y>0)$

The derivation above starts by selecting C_1, as the only clause of the first stage, for processing and decides the literal $x+1 \leq 0$ of C_1 to true. Then, since there are no more clauses in the first stage, it proceeds to select the value of -1 for x and we go to the next stage. At the stage of y, we start with propagating $x+y>0$ to be true by unit propagation on C_2. We continue by selecting the clause $x-y>0$ for processing. But now, under the assignment $\upsilon[M_1]$ which maps x to -1, the constraints $x+y>0$ and $x-y>0$ evaluate to $y-1>0$ and $-y-1>0$, respectively, which taken together are inconsistent, i.e. we

can not take $(x - y > 0)$ as it is incompatible with the current state. We therefore use the $\mathbb{R}$-PROPAGATE rule to propagate $\neg(x - y > 0) \equiv x - y \leq 0$.

In order to explain the propagation of $x - y \leq 0$, we use a Fourier-Motzkin elimination step to obtain

$$\begin{aligned} E_1 &\equiv (x + y > 0) \wedge (x - y > 0) \implies (x > 0) \\ &\equiv (x + y \leq 0) \vee (x - y \leq 0) \vee (x > 0) \ . \end{aligned}$$

For this to be a proper explanation, all literals, except for the implied one, should evaluate to false, which is indeed the case

$$\mathsf{value}(x + y \leq 0, M_1) = \mathsf{false} \ , \qquad \mathsf{value}(x > 0, M_1) = \mathsf{false} \ .$$

Therefore E_1 is a proper explanation and we can use it to propagate $x - y \leq 0$. In contrast, an SMT solver in this situation would learn a weaker explanation

$$E_1' \equiv (x + y > 0) \wedge (x - y > 0) \implies (x + 1 > 0) \ .$$

This is due to the fact that they use only existing literals for producing explanations.

As soon as we propagate $x - y \leq 0$, we enter a conflict with the clause $C_3 \equiv (x - y > 0)$ and therefore enter the conflict analysis mode.

$$\begin{aligned} &\langle [\![(x + 1 \leq 0), x \mapsto -1, C_2 \rightarrow (x + y > 0)]\!], \mathcal{C} \rangle_2 \vDash (x - y > 0) \\ &\downarrow \mathbb{R}\text{-PROPAGATE } (x - 1 \leq 0) \text{ with } E_1 \equiv (x + y \leq 0) \vee (x - y \leq 0) \vee (x > 0) \\ &\langle [\![(x + 1 \leq 0), x \mapsto -1, C_2 \rightarrow (x + y > 0), E_1 \rightarrow (x - y \leq 0)]\!], \mathcal{C} \rangle_2 \\ &\downarrow \text{CONFLICT} \\ &\langle [\![(x + 1 \leq 0), x \mapsto -1, C_2 \rightarrow (x + y > 0), E_1 \rightarrow (x - y \leq 0)]\!], \mathcal{C} \rangle_2 \vdash (x - y > 0) \end{aligned}$$

We now apply the conflict analysis rules to backtrack to the point where we can try some other option, and learn a clause from the conflict that will ensure we don't enter this particular conflict again. Below is the continuation of the derivation that uses the conflict analysis rules.

$$\langle [\![(x+1\leq 0), x\mapsto -1, C_2\to(x+y>0), E_1\to(x-y\leq 0)]\!], \mathcal{C}\rangle_2 \vdash (x-y>0)$$

$$\Big\downarrow \textsc{Resolve-Propagation}$$

$$\Big\downarrow \mathsf{resolve}(x-y>0, E_1, (x-y\leq 0)) = (x+y\leq 0)\vee(x>0)$$

$$\langle [\![(x+1\leq 0), x\mapsto -1, C_2\to(x+y>0)]\!], \mathcal{C}\rangle_2 \vdash (x+y\leq 0)\vee(x>0)$$

$$\Big\downarrow \textsc{Resolve-Propagation}$$

$$\Big\downarrow \mathsf{resolve}((x+y\leq 0)\vee(x>0), (x+y>0), (x+y>0)) = (x>0)$$

$$\langle [\![(x+1\leq 0), x\mapsto -1]\!], \mathcal{C}\rangle_2 \vdash (x>0)$$

$$\downarrow \textsc{Drop-Stage}$$

$$\langle \underbrace{[\![(x+1\leq 0)]\!]}_{M_2}, \mathcal{C}\cup\{x>0\}\rangle_1 \vDash (x>0)$$

We exit the conflict analysis by learning $x > 0$ and immediately proceed to the processing rules. Again, we can not choose $x > 0$ to be true, since it is incompatible with the current state M_2 which contains $x+1 \leq 0$. We therefore apply the $\mathbb{R}$-Propagate rule to propagate $x \leq 0$ and use Fourier-Motzkin elimination again to obtain the explanation

$$\begin{aligned} E_2 \equiv &(x+1\leq 0)\wedge(x>0) \implies (0>1)\\ \equiv &(x+1>0)\vee(x\leq 0)\ . \end{aligned}$$

This is a proper explanation since the literal $(x + 1 > 0)$ evaluates to false in the current state M_2. And, again, as soon as we propagate $x \leq 0$ we enter the conflict analysis rules due to a conflict with the learned clause $(x > 0)$. The resolution of the conflict then proceeds as follows.

$\langle [\![(x+1 \leq 0)]\!], \mathcal{C} \cup \{x > 0\} \rangle_1 \vDash (x > 0)$

$\downarrow \mathbb{R}$-Propagate $(x \leq 0)$ with $E_2 \equiv (x+1 > 0) \vee (x \leq 0)$

$\langle [\![(x+1 \leq 0), E_2 \rightarrow (x \leq 0)]\!], \mathcal{C} \cup \{x > 0\} \rangle_1$

$\downarrow$ Conflict

$\langle [\![(x+1 \leq 0), E_2 \rightarrow (x \leq 0)]\!], \mathcal{C} \cup \{x > 0\} \rangle_1 \vdash (x > 0)$

$\Big\downarrow$ Resolve-Propagation
$\mathsf{resolve}((x > 0), E_2, (x \leq 0)) = (x+1 > 0)$

$\langle [\![(x+1 \leq 0)]\!], \mathcal{C} \cup \{x > 0\} \rangle_1 \vdash (x+1 > 0)$

$\downarrow$ Resolve-Decision $(x+1 \leq 0)$

$\langle [\![]\!], \mathcal{C} \cup \{x > 0, x+1 > 0\} \rangle_1 \vDash (x+1 > 0)$

$\downarrow \mathbb{B}$-Propagate $(x+1 > 0)$

$\langle [\![(x+1 > 0) \rightarrow (x+1 > 0)]\!], \mathcal{C} \cup \{x > 0, x+1 > 0\} \rangle_1$

After the conflict analysis is done, we have learned not only that our first choice of $(x+1 \leq 0)$ was bad, but we also have a stronger learned clause $x > 0$. With this new information the procedure proceeds directly, without further conflicts, to construct a satisfying model of the original formula.

2.2.1 Termination

Our decision procedure consists of all three sets of rules described above. Any derivation will proceed by switching amongst the three distinct modes. Proving termination in the basic CDCL(T) framework is usually a fairly straightforward task, as the new explanation and conflict clauses always contain only literals from the finite set of literals in the initial set of constraints. In our case, the main conundrum in proving termination is that we allow the explanations to contain fresh constraints, which, if we are not careful, could lead to non-termination. We therefore re-

quire the set of new constraints to be finite. We call an explanation function explain a *finite basis explanation function* with respect to a set of constraints $\mathcal{C}$, when there is a finite set of polynomial constraints $\mathcal{B}$ such that for any derivation of the proof rules, the clauses returned by applications of explain always contain only constraints from the basis $\mathcal{B}$. Having such an explanation function will therefore provide us with a termination argument, and we will provide one such explanation function for the theory of reals in the next section.

Theorem 2.2. *Given a set of polynomial constraints $\mathcal{C}$, and assuming a finite basis explanation function* explain*, any derivation starting from the initial state $\langle [\![]\!], \mathcal{C}\rangle_1$ will terminate either in a state $\langle \upsilon, \mathsf{sat}\rangle$, where the assignment υ satisfies the constraints $\mathcal{C}$, or in the* unsat *state. In the latter case, the set of constraints $\mathcal{C}$ is unsatisfiable in $\mathbb{R}$.*

Proof. Assume we have a set of polynomial constraints $\mathcal{C}_0$, over the variables $x_1, \ldots, x_n$, and a finite-basis explanation function explain. Starting from the initial state $\langle [\![]\!], \mathcal{C}_0\rangle_1$, we claim that any derivation of the transition system (finite or infinite), satisfies the following properties

1. the derivation consists of only well-formed states;
2. the only possible "sink states" are the sat and the unsat states;
3. all $\vdash C$ clauses are implied by the initial constraints $\mathcal{C}_0$;
4. during conflict analysis the $\vdash C$ clause evaluates to false;

Assuming termination, and the above properties, the statement can be proven easily. Since sat and unsat are the only sink states, the derivation will terminate in one of these states. Since the Lift-Stage rule considers the variables $x_1, \ldots, x_n$ in order, we can only enter the satisfiable state if it is of the form $\langle \upsilon, \mathsf{sat}\rangle_{n+1}$. Consequently, by the precondition of the Lift-Stage rule, and the fact that we never remove the original constraints from $\mathcal{C}_0$, all the constraints in $\mathcal{C}_0$ are satisfied by υ. Therefore if we terminate in a sat state, the original problem is indeed satisfiable. On the other hand, if we terminate in the unsat state, by the above properties, the conflicting clause is implied by $\mathcal{C}_0$ and evaluates to false in the state $\langle [\![]\!], \mathcal{C}\rangle_1$. But, since there are no assertions in the trail, and variable assignment $\upsilon([\![]\!])$ does not assign any variables, it must be that the constraint is trivially false. Given that false is implied by the original constraints, the initial constraints themselves must truly be unsatisfiable.

The first two properties in the list above are a fairly easy exercise in case analysis and induction, so we skip those and concentrate on the more interesting properties. Proving the properties of conflict analysis is also quite straightforward, via induction on the number of conflicts, and conflict analysis steps. Clearly, initially, we have that C evaluates to false (the precondition of the CONFLICT rule), and is implied by $\mathcal{C}$ by induction. Then, every new clause that we produce during conflict resolution is obtained by the Boolean $\mathsf{resolve}$ rule, which will produce a valid deduction. Additionally, since the clause we are resolving with is a proper explanation, it will have all literals except the one we are resolving evaluate to false. Therefore, the resolvent also evaluates to false. As we backtrack down the trail with the conflicting clause, by definition of value and the preconditions of the rules, the clause still remains false.

Now, let us prove that the system terminates. It is clear that both the clause processing rules (one step transitions) and the conflict analysis rules (always removing elements from the trail) always terminate in a finite number of steps, and return to the search rules (or the unsat state). For the sake of the argument, let us assume that there is a derivation that does not terminate, and therefore does not enter the unsat state. We can define a big-step transition relation $\longrightarrow_{\text{bs}}$ that covers a transition from a search state, applying one or more transitions in the processing or analysis rules, and returns to a search state.

By assumption, we have a finite-basis explanation function $\mathsf{explain}$, so we can assume a set of polynomial constraint literals $\mathcal{B}$ from which all the clauses that we can see during the search are constructed. In order to keep progress of the search, we first define a function $\mathsf{search\text{-}level}$ that, given the trail M, returns a pair (k, l), where k is the index of the next variable we are trying to assign, i.e. k is one more than the number of variable assignments in M, and l is the number of decided literals (applications of the DECIDE-LITERAL rule) in M. Note that the $\mathsf{search\text{-}level}$ of any state that we can encounter is always a pair (k, l) with $1 \leq k \leq n$ and $l \leq |\mathcal{B}|$. Given such a pair (k, l) we define the function $\mathsf{subseq}(M, k, l)$ to be the largest prefix of M that contains at most k variable assignments and at most l decided literals, i.e. the largest prefix of M with $\mathsf{search\text{-}level}(M) \leq (k, l)$ (using the lexicographic order).

To define the measure of a state, we first define a series of weight functions ω_k that, given a

sequence M, returns

$$\omega_{k,l}(M) = \begin{cases} |\{\ F \in \mathcal{B}_{k+1}\ \mid\ \mathsf{value}(F, \mathsf{subseq}(M,k,l)) = \mathsf{undef}\ \}| & (k,l) \leq \mathsf{search\text{-}level}(M), \\ \infty & \text{otherwise.} \end{cases}$$

In other words, if we are trying to assign the variable x_k, where we already performed a number of literal decisions, this state is as heavy as the number of literals left in the basis containing only variables $x_1, \ldots, x_k$ that could still possibly be assigned.

In order to prove termination, we will track the progress of all levels simultaneously. We define the function Ω to map the sequence M of a well-formed state $\langle M, \mathcal{C} \rangle_k$ into a $n(|\mathcal{B}|+1)+2$-tuple as

$$\begin{aligned} \Omega(M) = \langle\ & \omega_{1,0}(M),\ \omega_{1,1}(M),\ \ldots,\ \omega_{1,|\mathcal{B}|}(M), \\ & \omega_{2,0}(M),\ \omega_{2,1}(M),\ \ldots,\ \omega_{2,|\mathcal{B}|}(M), \\ & \vdots \\ & \omega_{n,0}(M),\ \omega_{n,1}(M) \ldots, \omega_{n,|\mathcal{B}|}(M), \\ & \omega_{n+1,0}(M), |\mathcal{C}|\ \rangle\ . \end{aligned}$$

Given two well-formed states with trails M_1 and M_2, we write $M_1 \lessdot M_2$ if $\Omega(M_1) <_{\mathsf{lex}} \Omega(M_2)$, where $<_{\mathsf{lex}}$ is the natural lexicographical extension of the order $<$ on $\mathbb{N} \cup \{\infty\}$. Now consider a transition of the search $\langle M_1, \mathcal{C}_1 \rangle_{k_1} \longrightarrow_{\mathrm{bs}} \langle M_2, \mathcal{C}_2 \rangle_{k_2}$, with $\mathsf{search\text{-}level}(M_1) = (k_1, l_1)$, and the following cases.

- If this transition was initiated by the Select-Clause rule, either a new literal was assigned at the current literal decision level by propagation, or a new decision was introduced. In both cases $M_2 \lessdot M_1$ as either the element (ω_{k_1,l_1}) of the sequence decreased by 2 (literal and its negation were assigned), or the next element of the sequence decreased from ∞ to a finite value (ω_{k_1,l_1+1}).

- If this transition was initiated by the Lift-Stage rule, the element $\omega_{k_1+1,0}$ in $\Omega(M)$ decreased from ∞ to a finite value, so $M_2 \lessdot M_1$ again.

- If we went into conflict analysis mode via the Conflict rule, we will backtrack to the search level (k_2, l_2), learn a new clause, and then assign at least one new literal of the

learned clause. We do so since from conflict analysis we always transition into the clause processing rules. We exit conflict resolution either using the DROP-STAGE rule, or the RESOLVE-DECISION rule. If we used the RESOLVE-DECISION rule, we will assign the single undefined literal of the learned clause using one of the propagation rules, thus decreasing ω_{k_2,l_2}. If we used the DROP-STAGE rule, we know that $\omega_{k_2,l_2+1}(M_1) = \infty$, and that the learnt clause has at least one undefined literal that we can assign a value. In this case we decrease the measure as in the case of SELECT-CLAUSE. In both cases we have that $M_2 \lessdot M_1$.

- If we applied the FORGET rule, it is clear that only last element of the measure decreases, and hence also $M_2 \lessdot M_1$.

Since, we covered all cases, the function Ω is always decreasing, and termination of the system follows. □

2.3 Producing Explanations

Given a polynomial constraint F with $\mathsf{poly}(F) \in \mathbb{Z}[\vec{y}, x]$, and a trail M such that $\neg F$ is not compatible with M, the procedure $\mathsf{explain}(F, M)$ returns an explanation clause E that implies F in the current state. This clause is of the form $\mathcal{E} \wedge \mathcal{F} \implies F$, where $\mathcal{E}$ and $\mathcal{F}$ are sets of literals with $\mathsf{poly}(\mathcal{E}) \subset \mathbb{Z}[\vec{y}]$ and $\mathsf{poly}(\mathcal{F}) \subset \mathbb{Z}[\vec{y}, x]$. All literals in $\mathcal{F}$ occur in M, and all literals in $\mathcal{E}$ evaluate to true in the current assignment. Note that $\mathcal{E}$ may contain new literals, so we must ensure that the new literals in $\mathsf{poly}(\mathcal{E})$ are a subset of some finite basis. In principle, for any theory that admits elimination of quantifiers, it is possible to construct an explanation function $\mathsf{explain}$. In this section, we describe how to produce an $\mathsf{explain}$ procedure for theory of the reals based on cylindrical algebraic decomposition (CAD). Before that, we first make a short interlude into the world of CAD.

2.3.1 Cylindrical Algebraic Decomposition

A crucial role in the theory of CADs and in the construction of our $\mathsf{explain}$ procedure is the property of delineability. Let us first give some intuition on what delineability means and how

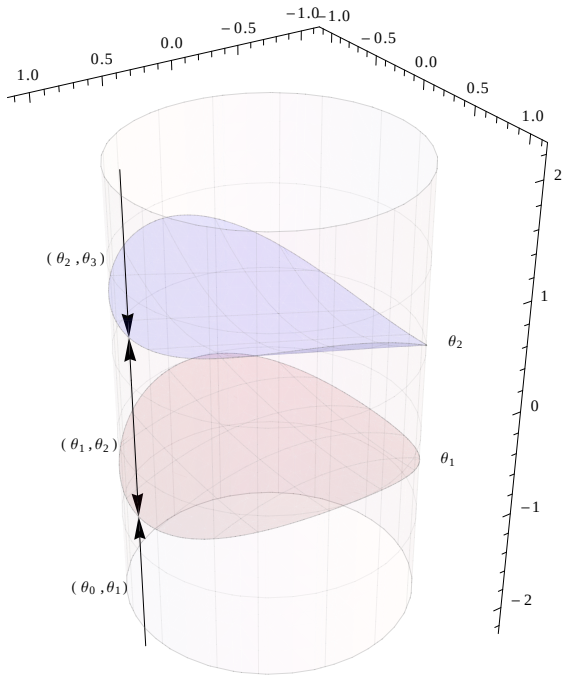


Figure 2.4: Diagram for Example 2.5 depicting a delineable region.

we might take advantage of it, by means of an example.

Example 2.5. Let us consider two polynomials f_1 and f_2 in $\mathbb{Z}[x, y, z]$

$$f_1 = 30z + 10x^2 - 3y^2 + 15 \ , \qquad f_2 = 6z - 2x - 3y^2 - 3 \ ,$$

and consider the region $S = \{ \ (x, y) \mid x^2 + y^2 < 1 \ \}$ and the cylinder $Z = S \times \mathbb{R} = \{ \ (x, y, z) \mid x^2 + y^2 < 1 \ \}$ over S.

To analyze f_1 and f_2 and how their signs change in the cylinder Z, we can solve the equations $f_1 = 0$ and $f_2 = 0$ for z, obtaining functions θ_1 and θ_2 that describe the zeroes of f_1 and f_2

$$z = \theta_1(x, y) = -\frac{1}{3}x^2 + \frac{1}{10}y^2 - \frac{1}{2} \ , \qquad z = \theta_2(x, y) = \frac{1}{3}x + \frac{1}{2}y^2 + \frac{1}{2} \ .$$

The cylinder Z and the functions θ_1 and θ_2 are depicted in Figure 2.4. As can be seen from the figure, the roots of f_1 and f_2 are separated, and so in each of the marked regions $(\theta_0 - \theta_1)$, θ_1, $(\theta_1 - \theta_2)$, θ_2, and $(\theta_2 - \theta_3)$ the signs of the polynomials f_1 and f_2 will be invariant. Moreover, for any point $\alpha \in S$, if we move z from $-\infty$ to $+\infty$, the polynomials f_1 and f_2 will always produce the same sequence of signs.

	(θ_0, θ_1)	θ_1	(θ_1, θ_2)	θ_2	$(\theta_2 - \theta_3)$
$\mathsf{sgn}(f_1(\alpha, z))$	-1	0	1	1	1
$\mathsf{sgn}(f_2(\alpha, z))$	-1	-1	-1	0	1

Now, imagine that we are solving constraints $f_1 < 0$ and $f_2 > 0$, therefore requiring the sign combination to be $(-1, +1)$. If we try to find the required combination of signs at some particular point in $\alpha \in S$, we will fail. But now, due to the polynomials behaving as above, i.e. with roots cleanly separated across the cylinder Z, we can safely conclude that we can not find the solution anywhere in S. In particular, we can explain the conflict that the choice of α introduced by the constraint $x^2 + y^2 \geq 1$ that describes the points not in S.

Following the terminology used in CAD, we say that a connected subset of $\mathbb{R}^k$ is a *region*. Given a region S, the *cylinder* Z over S is $S \times \mathbb{R}$. A θ-*section* of Z is a set of points $\langle \vec{\alpha}, \theta(\vec{\alpha}) \rangle$, where $\vec{\alpha}$ is in S and θ is a continuous function from S to $\mathbb{R}$. A (θ_1, θ_2)-*sector* of Z is the set of points $\langle \vec{\alpha}, \beta \rangle$, where $\vec{\alpha}$ is in S and $\theta_1(\vec{\alpha}) < \beta < \theta_2(\vec{\alpha})$ for continuous functions $\theta_1 < \theta_2$ from S to $\mathbb{R}$. Sections and sectors are also regions. Given a subset of S of $\mathbb{R}^k$, a *decomposition* of S is a finite collection of disjoint regions $S_1, \ldots, S_n$ such that $S_1 \cup \ldots \cup S_n = S$. Given a region S, and a set of continuous functions $\theta_1 < \ldots < \theta_n$ from S to $\mathbb{R}$, we can decompose the cylinder $S \times \mathbb{R}$ into the following regions:

- the θ_i-sections, for $1 \leq i \leq n$, and
- the (θ_i, θ_{i+1})-sectors, for $0 \leq i \leq n$,

where, with slight abuse of notation, we define θ_0 as the constant function that returns $-\infty$ and θ_{n+1} the constant function that returns ∞.

A set of polynomials $\{f_1, \ldots f_s\} \subset \mathbb{Z}[\vec{y}, x]$, with $\vec{y} = (y_1, \ldots, y_n)$, is said to be *delineable* in a region $S \subseteq \mathbb{R}^n$ if for every f_i (and f_j) from the set, the following properties are invariant for any $\alpha \in S$:

1. the *total number of complex roots* of $f_i(\alpha, x)$;
2. the *number of distinct complex roots* of $f_i(\alpha, x)$;
3. the *number of common complex roots* of $f_i(\alpha, x)$ and $f_j(\alpha, x)$.

Delineability, as described above, is particularly useful because all three properties of the definition have an algebraic characterization in terms of polynomial operations. For our purpose it is the consequence of delineability on the arrangement of real roots that matters the most. As explained by the following theorem, if a set of polynomials A is delineable on a region S, then the number of real roots of the polynomials does not change on S. Moreover, these roots maintain their relative order on the whole of S. This will imply that any set of polynomial constraints over polynomials in A will have invariant truth values over the region S.

Theorem 2.3 (Corollary 8.6.5 of [68]). *Let A be a set of polynomials in $\mathbb{Z}[\vec{y}, x]$, delineable in a region $S \subset \mathbb{R}^n$. Then, the real roots of A vary continuously over S, while maintaining their order.*

Example 2.6. Consider the polynomial $f = x^2 + y^2 + z^2 - 1$, with zeros of f depicted in Fig 2.5 together with two squiggly regions of $\mathbb{R}^2$. In the region S_1 that does not intersect the sphere, polynomial f is delineable, as the number of real roots of $f(\alpha, x)$ is 2 for any α in S_1. In the region S_2 that intersects the sphere, f is not delineable, as the number of real roots of f may be from 0 (α's outside the unit circle), 1 (on the circle), or 2 (inside the unit circle).

In order to reason about a set of polynomials A, we will decompose the real space into regions where these polynomials are delineable. We can identify these regions by referring to the roots of a

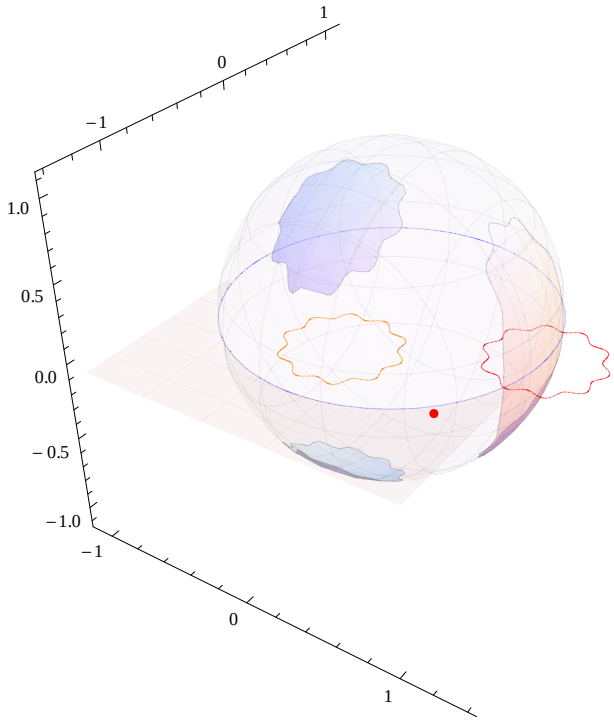


Figure 2.5: The sphere corresponding to the roots of $x^2 + y^2 + z^2 - 1$, and regions of Ex 2.6 and Ex 2.8.

.

related set of polynomials that are derived from the set A by using a projection operator. We will call a *projection operator* any map P that, given a variable x and set of polynomials $A \subset \mathbb{Z}[\vec{y}, x]$, transforms A into a set of polynomials $\mathsf{P}(A, x) \subset \mathbb{Z}[\vec{y}]$. We call $\mathsf{P}(A, x)$ the *projection* of A under P with respect to variable x. In his seminal paper [26], Collins introduced a projection operator which we denote with P_c. In order to define the operator P_c, we first need to define some "advanced" operations on polynomials, and we refer the reader to [64, 9, 17] for a more detailed exposition.

Let $f, g \in \mathbb{Z}[\vec{y}, x]$ be two polynomials with $n = \min(\mathsf{deg}(f, x), \mathsf{deg}(g, x))$. For $k = 0, \ldots, n-1$, we denote with $\mathsf{S}_k(f, g, x)$ the k-th *subresultant* of f and g. The k-th subresultant is defined as the determinant of the k-th Sylvester-Habicht matrix of f and g, and is a polynomial of degree $\leq k$ in x with coefficients in $\mathbb{Z}[\vec{y}]$. The matrix in question is a particular matrix containing as

elements the coefficients of f and g. Additionally, we denote with $\mathsf{psc}_k(f,g,x)$ the k-th *principal subresultant coefficient* of f and g, which is the coefficient of x^k in the polynomial $\mathsf{S}_k(f,g,x)$, and define $\mathsf{psc}_n(f,g,x) = 1$. We denote the sequence of principal subresultant coefficients as $\mathsf{psc}(f,g,x) = (\mathsf{psc}_0(f,g,x), \ldots, \mathsf{psc}_n(f,g,x))$.

We can ensure delineability of a set of polynomials A, as defined above, if we can define a region where we can control the behavior of complex roots of the polynomials in A. Since the number of common complex roots of two polynomials corresponds to the degree of their gcd, the following theorem provides us with an algebraic means to describe this number using the operations we introduced above.

Theorem 2.4 (Theorem 2 in [26]). *Let $f, g \in \mathbb{Z}[\vec{y}, x]$ be non-zero polynomials. Then the degree $\mathsf{deg}(\mathsf{gcd}(f,g),x) = k$ if and only if k is the least j such that $\mathsf{psc}_j(f,g) \neq 0$.*

We now present the projection operator that Collins introduced [26], that can be used to capture the regions of delineability.

Definition 2.5 (Collins Projection). Given a set of polynomials $A = \{f_1, \ldots, f_m\} \subset \mathbb{Z}[\vec{y}, x]$ the Collins projector operator $\mathsf{P}_c(A,x)$ is defined as

$$\bigcup_{f \in A} \mathsf{coeff}(f,x) \cup \bigcup_{\substack{f \in A \\ g \in \mathsf{R}^*(f,x)}} \mathsf{psc}(g, g'_x, x) \cup \bigcup_{\substack{i<j \\ g_i \in \mathsf{R}^*(f_i,x) \\ g_j \in \mathsf{R}^*(f_j,x)}} \mathsf{psc}(g_i, g_j, x) \ ,$$

In order to denote the individual parts of the projection, in order, we designate them as $\mathsf{P}_c^1(A,x)$, $\mathsf{P}_c^2(A,x)$ and $\mathsf{P}_c^3(A,x)$.

It can be shown that, given a set of polynomials A, the Collins projection operator P_c produces a new set (the projection) of polynomials that can be used to describe the regions where A is delineable. This description of delineability relies only on the sign-invariance of the projection polynomials, where by sign-invariance we mean the following. Let $A = \{f_1, \ldots, f_m\} \subset \mathbb{Z}[\vec{y}]$ be a set of polynomials, where $\vec{y} = (y_1, \ldots, y_n)$, and S be a region of $\mathbb{R}^n$. If for any assignment υ such that $\upsilon(\vec{y}) = \vec{\alpha} \in S$, the polynomials in A have the same sign under υ, we say that A is *sign-invariant on S*.

Theorem 2.6 (Theorem 4 in [26]). *Let $A \subset \mathbb{Z}[\vec{y}, x]$ be a finite set of polynomials, where $\vec{y} = (y_1, \ldots, y_n)$, and let S be a region of $\mathbb{R}^n$. If $\mathsf{P}_c(A)$ is sign invariant on S, then A is delineable over S.*

The projection operator P_c guarantees delineability on any region S where the projection set $\mathsf{P}_c(A, x)$ is sign-invariant, due to the following:

1. The degree of $f_i(\alpha, x)$ (and the total number of complex roots) remains invariant for any α in S, by $\mathsf{P}_c^1(A, x)$ being sign-invariant.

2. The multiplicities of complex roots of $f_i(\alpha, x)$ remains invariant for any α in S, by $\mathsf{P}_c^2(A, x)$ being sign-invariant and Theorem 2.4.

3. The number of common complex roots of $f_i(\alpha, x)$ and $f_j(\alpha, x)$ remain invariant for any α in S, by $\mathsf{P}_c^3(A, x)$ being sign-invariant and Theorem 2.4.

A *sign assignment* for a set of polynomials A is a mapping σ, from polynomials in A to $\{-1, 0, 1\}$. Given a set of polynomials $A \subset \mathbb{Z}[\vec{y}, x]$, we say a sign assignment σ is *realizable* with respect to some $\vec{\alpha}$ in $\mathbb{R}^n$, if there exists a $\beta \in \mathbb{R}$ such that every $f \in A$ takes the sign corresponding to its sign assignment, i.e., $\mathsf{sgn}(f(\vec{\alpha}, \beta)) = \sigma(f)$. The function sgn maps a real number to its sign $\{-1, 0, 1\}$. We use $\mathsf{signs}(A, \alpha)$ to denote the set of realizable sign assignments of A with respect to α.

Lemma 2.1. *If a set of polynomials $A \subset \mathbb{Z}[\vec{y}, x]$ is delineable over a region S, then $\mathsf{signs}(A, \alpha)$ is invariant over S.*

Proof. Since A is delineable over S, by Theorem 2.3, there are real functions θ_i, continuous on S and ordered, corresponding to roots of polynomials in A. We can therefore decompose the cylinder $S \times \mathbb{R}$ into θ_i-sections and (θ_i, θ_{i+1})-sectors, where each of these regions is connected and the signs of polynomials from A do not change. Let $\sigma_1 \in \mathsf{signs}(A, \vec{\alpha}_1)$ be a realizable sign assignment, with $\beta_1 \in \mathbb{R}$, such that at $(\vec{\alpha}_1, \beta_1)$ every polynomial $f \in A$ takes a sign corresponding to $\mathsf{signs}(A, \vec{\alpha}_1)$. Lets pick an arbitrary other $\alpha_2 \in S$, and show that we realize σ_1 at $\vec{\alpha}_2$. We can pick an arbitrary point β_2 in the same sector (or section) R where β_1 came from. We claim that at $\langle \vec{\alpha}_2, \beta_2 \rangle$ the polynomials in A have the signs required by σ_1.

Assume the opposite, i.e. that there is a polynomial $f \in A$ with $\sigma_1(f) = \mathsf{sgn}(f(\vec{\alpha}_1, \beta_1)) \neq \mathsf{sgn}(f(\vec{\alpha}_2, \beta_2))$. Since R is connected we can connect $\langle \vec{\alpha}_1, \beta_1 \rangle$ and $\langle \vec{\alpha}_2, \beta_2 \rangle$ with a path π that does not leave R. Having that the sign of f is different at the endpoints of π, it must be that there is a point $\langle \vec{\alpha}_3, \beta_3 \rangle$ on the path, where the sign of f is 0, and at least another point where the sign of f is not 0. Now we distinguish the following cases

- If R is a (θ_i, θ_{i+1})-sector, then we have isolated a root of a polynomial in A that is between $\theta_i(\vec{\alpha}_3)$ and $\theta_{i+1}(\vec{\alpha}_3)$, which is impossible by the construction of the decomposition.
- If R is a θ_i-section, then the polynomial $f(\vec{\alpha}_3)$ has a root, and this root diverges from θ_i on R, which is impossible due to delineability. □

2.3.2 Projection-Based Explanations

Suppose that we need to produce an explanation for propagating a polynomial constraint F, i.e. we are in a state such that $\neg\,\mathsf{compatible}(\neg F, M)$, with $\mathsf{poly}(F) \in \mathbb{Z}[\vec{y}, x]$, where $\vec{y} = (y_1, \ldots, y_n)$. To simplify the presentation, in the following, we write υ for $\upsilon[M]$. The explanation procedure $\mathsf{explain}(F, M)$ consists of the following steps.

IsolateCore: Find a minimal set $\mathcal{F}$ of literals in M such that $\upsilon(\mathcal{F} \cup \{\neg F\})$ does not allow a solution for x. We call the set $\mathcal{F} \cup \{\neg F\}$ (not necessarily unique) a *conflicting core*.

Project: Construct a region S of $\mathbb{R}^n$ where $A = \mathsf{poly}(\mathcal{F} \cup \{F\})$ is delineable, and $\upsilon(\vec{y}) = \vec{\alpha}$ is in S. Note that, from Lemma 2.1, $\neg F$ is incompatible with $\mathcal{F}$ for any other assignment to $\vec{y}$ to $\vec{\alpha}'$ in S.

Explain: Define the region S using extended polynomial constraints, obtaining a set of constraints $\mathcal{E}$. Then, we define $\mathsf{explain}(F, M) \equiv (\mathcal{E} \wedge \mathcal{F}) \implies F$.

We focus here on the second step of the procedure. To obtain the region S we will use a projection operator which, with insights of Theorem 2.6, will ensure delineability. Since our procedure requires a region S that contains the current assignment $\upsilon(\vec{y}) = \vec{\alpha}$, we add the assignment υ as an additional argument to the projection operator, and call such a projection operator *model-based*. Given a variable assignment υ, we denote the vanishing signature of a principal

subresultant sequence as $\mathsf{v\text{-}psc}(f, g, x, \upsilon) = \mathsf{v\text{-}sig}(\mathsf{psc}_0(f, g, x), \ldots, \mathsf{psc}_n(f, g, x), \upsilon)$, and define our model-based projection operator $\mathsf{P}_m(A, x, \upsilon)$ as follows.

Definition 2.7 (Model-Based Projection). Given a set of polynomials $A = \{f_1, \ldots, f_m\} \subset \mathbb{Z}[\vec{y}, x]$ and a variable assignment υ, the model-based Collins projector operator $\mathsf{P}_m(A, x, \upsilon)$ is defined as

$$\bigcup_{f \in A} \mathsf{v\text{-}coeff}(f, x, \upsilon) \cup \bigcup_{\substack{f \in A \\ g = \mathsf{R}(f, x, \upsilon)}} \mathsf{v\text{-}psc}(g, g'_x, x, \upsilon) \cup \bigcup_{\substack{i<j \\ g_i = \mathsf{R}(f_i, x, \upsilon) \\ g_j = \mathsf{R}(f_j, x, \upsilon)}} \mathsf{v\text{-}psc}(g_i, g_j, x, \upsilon) \ .$$

In order to denote the individual parts of the projection, in order, we designate them as $\mathsf{P}^1_m(A, x, \upsilon)$, $\mathsf{P}^2_m(A, x, \upsilon)$ and $\mathsf{P}^3_m(A, x, \upsilon)$.

Example 2.7. Consider the variable assignment υ, with $\upsilon(x) = 0$, and the set A containing two polynomials $f_2 = x^2 + y^2 - 1$ and $f_3 = -4xy - 4x + y - 1$. The projection operator P_m maps the set A into $\mathsf{P}_m(A, y, \upsilon)$

$$\{\ (\underbrace{16x^3 - 8x^2 + x + 16}_{f_1})x,\ -4x + 1,\ 4(x+1)(x-1),\ 2,\ 1\ \}\ , \tag{2.2}$$

where f_1 is the polynomial from Ex. 2.1. The zeros of f_2 and f_3 are depicted in Fig. 2.7, together with a set of important points $\{-1, \alpha_1, 0, \frac{1}{4}, 1\}$, where α_1 is the algebraic number from Ex. 2.1. These points are exactly the roots of the projection polynomials (2.2). It is easy to see that f_2 and f_3 are delineable in the intervals defined by these points. But, considering a polynomial $f_4 = x^3 + 2x^2 + 3y^2 - 5$, we can see that it is not delineable on the interval $(1, +\infty)$.

We will use the projection operator P_m to compute the required region S and the constraints $\mathcal{E}$ that define it, and show that A is delineable in S. First, we close the set of polynomials $A \subset \mathbb{Z}[y_1, \ldots, y_n, x]$ under the application of a projection operator P_m. We compute this closure by computing sets of polynomials $\mathcal{P}^n, \ldots, \mathcal{P}^1$ iteratively, starting from $\mathcal{P}^n = \mathsf{P}_m(A, \upsilon, x)$, and

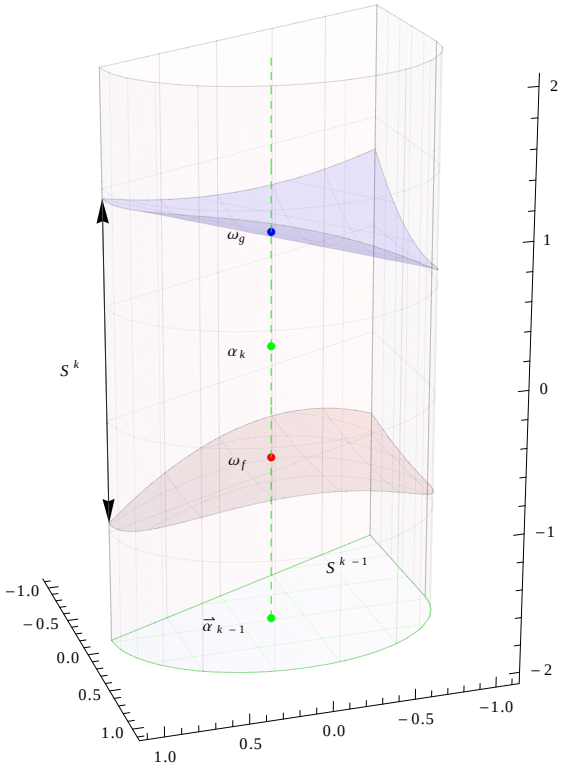


Figure 2.6: Diagram representing a step in construction of the explanation.

then for $k = n, \ldots, 2$, compute the subsequent ones as

$$\mathcal{P}^{k-1} = \mathsf{P}_m(\mathcal{P}^k, y_k, \upsilon) \cup (\mathcal{P}^k \cap \mathbb{Z}[y_1, \ldots, y_{k-1}]) \ .$$

Each set of polynomials $\mathcal{P}^k \subseteq \mathbb{Z}[y_1, \ldots, y_k]$ is obtained by projecting the previous set $\mathcal{P}^{k+1}$ and adding all the polynomials from $\mathcal{P}^{k+1}$ that do not involve the variable y_{k+1}.

Now, we can build the region S inductively, in a bottom-up fashion, by constructing a sequence of regions $S^k \subset \mathbb{R}^k$, and the corresponding set of constraints $\mathcal{E}^k$ that define them, such that

- $\mathcal{P}^1 \cup \cdots \cup \mathcal{P}^k$ is sign invariant in S^k, and
- $\mathcal{P}^{k+1}$ is delineable in S^k.

For convenience, we include a diagram representing one step in this construction in Figure 2.6.

Assume, by induction that the region S^{k-1}, and its defining constraints $\mathcal{E}^{k-1}$, have already been constructed. Now, consider the set of root objects

$$R^k = \left\{ \mathsf{root}(f, i, y_k) \;\mid\; f \in \mathcal{P}^k,\ 1 \leq i \leq \mathsf{rootcount}(f, y_k, \upsilon) \right\} \quad .$$

Under the given assignment υ each of the root objects $\mathsf{root}(f, i)$ is defined and evaluates to some value $\omega_f^i \in \mathbb{R}_{\mathsf{alg}}$. Moreover, since the polynomials in $\mathcal{P}^k$ are delineable over S^{k-1} by inductive assumption, then by Theorem 2.3, for any other assignment υ' that maps $y_1, \ldots, y_{k-1}$ into S^{k-1}, the polynomials $f \in \mathcal{P}^k$ will have the same number of roots, and the same number of common roots. Therefore, the root objects in R^k will also be defined under any such υ', and will evaluate to values that are in the same exact order.

The values ω_f^i partition the real line into maximal intervals where the polynomials $f \in \mathcal{P}^k$ are sign invariant. We pick the one interval that contains $\upsilon(y_k) = \alpha_k$ and construct the defining constraints $\mathcal{E}^k$ of the region S^k by selecting one of the appropriate cases

$$\begin{aligned}
&\alpha_k \in (\omega_f^i, \omega_g^j) && \implies \mathcal{E}^k = \mathcal{E}^{k-1} \cup \{\ y_k >_r \mathsf{root}(f, i, y_k), y_k <_r \mathsf{root}(g, j, y_k)\ \} \quad , \\
&\alpha_k \in (-\infty, \omega_f^i) && \implies \mathcal{E}^k = \mathcal{E}^{k-1} \cup \{\ y_k <_r \mathsf{root}(f, i, y_k)\ \} \quad , \\
&\alpha_k \in (\omega_f^i, +\infty) && \implies \mathcal{E}^k = \mathcal{E}^{k-1} \cup \{\ y_k >_r \mathsf{root}(f, i, y_k)\ \} \quad , \\
&\alpha_k = \omega_f^i && \implies \mathcal{E}^k = \mathcal{E}^{k-1} \cup \{\ y_k =_r \mathsf{root}(f, i, y_k)\ \} \quad .
\end{aligned}$$

In other words, we pick S^k to be the region corresponding to a section (or a sector) of the cylinder over S^{k-1} that contains the assignment value $(\alpha_1, \ldots, \alpha_k)$. We know that this is well defined as, due to delineability of $\mathcal{P}^k$ over S^{k-1}, the decomposition into the sections and sectors is possible.

Finally, we guarantee that $\mathcal{P}^{k+1}$ is delineable in S^k. First, because $\mathcal{P}^* = \mathcal{P}^1 \cup \ldots \cup \mathcal{P}^k$ is by construction sign invariant over the region S^k, then so is $\mathsf{P}_m(\mathcal{P}^{k+1}, \upsilon, y_{k+1})$ as a subset of $\mathcal{P}^*$. Since the projection is sign-invariant on S^k we use the resoning similar to the proof of Theorem 2.6 to show delineability. In constrast to P_c, the model-based projection operator P_m does not include all coefficients, reductums, and the whole psc chain, because the current assignment indicates which coefficients will (and will not) vanish in any element of S^k. But, it does include precisely the ones are matter, i.e. for all polynomials $f, g \in \mathcal{P}^{k+1}$ and all $(\beta_1, \ldots, \beta_k) \in S^k$ the following holds.

1. The degree and the total number of complex roots of $f(\beta_1, \ldots, \beta_k, y_{k+1})$ is invariant by $\mathsf{P}_m^1(\mathcal{P}^{k+1}, x, \upsilon)$ being sign-invariant on S^k.

2. The multiplicity of the complex roots of $f(\beta_1, \ldots, \beta_k, y_{k+1})$ is invariant by $\mathsf{P}_m^2(\mathcal{P}^{k+1}, x, \upsilon)$ being sign-invariant on S^k and Theorem 2.4.

3. The number of common complex roots of $f(\beta_1, \ldots, \beta_k, y_{k+1})$ and $g(\beta_1, \ldots, \beta_k, y_{k+1})$ is invariant by $\mathsf{P}_m^3(\mathcal{P}^{k+1}, x, \upsilon)$ being sign-invariant and Theorem 2.4.

Since the above are the requirements of delineability, $\mathcal{P}^{k+1}$ is indeed delineable in S^k.

Once we have computed the regions $S^1, \ldots, S^n$, we can use the region $S = S^n$ and the corresponding constraints $\mathcal{E} = \mathcal{E}^n$ to explain why $\neg F$ is incompatible with $\mathcal{F}$. Thus, we set $\mathsf{explain}(F, M) \equiv (\mathcal{E} \wedge \mathcal{F}) \implies F$.

Theorem 2.8. *The explanation function* $\mathsf{explain}(F, M)$ *is a finite-basis explanation function for the existential theory of real closed fields.*

Proof. The key observation is that $\mathsf{P}_m(A, x, \upsilon) \subseteq \mathsf{P}_c(A, x)$, for any A, x and υ. Let $A_0 \subset \mathbb{Z}[x_1, \ldots, x_n]$ be the set of polynomials in the initial set of constraints $\mathcal{C}_0$. Using Collins projection operator $\mathsf{P}_c(A, x)$ we define the sets of polynomials $\mathcal{A}^n, \ldots, \mathcal{A}^1$ iteratively, starting from $\mathcal{A}^n = A_0$, and then for $k = n, \ldots, 2$,

$$\mathcal{A}^{k-1} = \mathsf{P}_c(\mathcal{A}^k, x_k) \cup (\mathcal{A}^k \cap \mathbb{Z}[x_1, \ldots, x_{k-1}])$$

Now, let $\mathcal{A}_c$ be the set $\mathcal{A}^n \cup \ldots \cup \mathcal{A}^1$. The set $\mathcal{A}_c$ is finite and for any $A \subseteq \mathcal{A}_c$ and variable x, we have $\mathsf{P}_c(A, x) \subseteq \mathcal{A}_c$. Consequently, for any $A \subseteq \mathcal{A}_c$, variable x, and assignment υ, $\mathsf{P}_m(A, x, \upsilon) \subseteq \mathcal{A}_c$.

Given a finite set of polynomials A, we have finitely many different polynomial constraints F s.t. $\mathsf{poly}(F) \in A$. This is clear for basic constraints, there are $6 \times |A|$ different basic constraints. For extended constraints $x \bigtriangledown_r \mathsf{root}(f, k)$, we recall that $k \leq \mathsf{deg}(f, x)$. Let $\mathcal{B}_c$ be the set $\{F \mid \mathsf{poly}(F) \in \mathcal{A}_c\}$. Thus, $\mathcal{B}_c$ is finite.

Now, it is clear that $\mathsf{explain}(F, M)$ is a finite basis explanation function. Given an initial set of constraints $\mathcal{C}_0$, for any application of $\mathsf{P}_m(A, x, \upsilon)$ in any application of $\mathsf{explain}(F, M)$ in any derivation of our procedure, we have that $\mathsf{P}_m(A, x, \upsilon) \subseteq \mathcal{A}_c$, and consequently $\mathcal{B}_c$ is a finite basis for $\mathsf{explain}(F, M)$. □

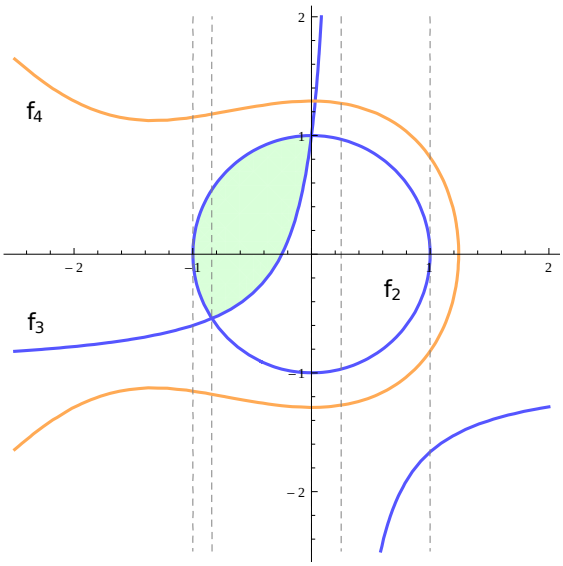


Figure 2.7: Solutions of $f_2 = x^2 + y^2 - 1 = 0$, $f_3 = -4xy - 4x + y - 1 = 0$, and $f_4 = x^3 + 2x^2 + 3y^2 - 5 = 0$, with the solution set of $\{f_2 < 0, f_3 > 0, f_4 < 0\}$ emphasized. The dashed lines represent the zeros of the projection set (2.2).

Example 2.8. Consider the polynomial $f = x^2 + y^2 + z^2 - 1$, from Example 2.6, and the constraint $f < 0$ corresponding to the interior of the sphere in Figure 2.5. Under an assignment υ with $\upsilon(x) = \frac{3}{4}$ and $\upsilon(y) = -\frac{3}{4}$ (the red point in Fig 2.5) this constraint does not allow a solution for z (it evaluates to $z^2 < -\frac{1}{8}$). In order to explain it, we can compute the projection closure of $A = \{f\}$, using P_m, obtaining

$$\mathcal{P}_3 = A = \{\ x^2 + y^2 + z^2 - 1\ \}\ ,$$
$$\mathcal{P}_2 = \mathsf{P}_m(\mathcal{P}_3, \upsilon, z) = \{\ 4x^2 + 4y^2 - 4,\ 2,\ 1\ \}\ ,$$
$$\mathcal{P}_1 = \mathsf{P}_m(\mathcal{P}_2, \upsilon, y) = \{\ 256x^2 - 256,\ 8,\ 4,\ 2,\ 1\ \}\ .$$

The sets of root objects under υ are then

$$R^2 = \{\ \mathsf{root}(y^2 + x^2 - 1, 1, y),\ \mathsf{root}(y^2 + x^2 - 1, 2, y)\ \}\ ,$$
$$R^1 = \{\ \mathsf{root}(x^2 - 1, 1, x),\ \mathsf{root}(x^2 - 1, 2, x)\ \}\ .$$

The root objects of R_1 evaluate to -1 and 1, respectively, and since $v(x) = \frac{3}{4} = 0.75$, the constraints corresponding to the region S^1 are $(x > -1)$ and $(x < 1)$. The root objects of R_2 evaluate to

$$\omega_1 = -\frac{\sqrt{7}}{4} \approx -0.6614 \ , \qquad \omega_2 = \frac{\sqrt{7}}{4} \approx 0.6614 \ .$$

Since $v(y) = -\frac{3}{4} = -0.75$ and thus $v(y) \in (-\infty, \omega_1)$, we describe the region S^2 with the additional constraint $(y < \mathsf{root}(y^2 - x^2 - 1, 1, y))$. Using the constraints defining the region S^2 we construct the explanation $\mathsf{explain}(f < 0, v)$ as

$$(x \leq -1) \vee (x \geq 1) \vee \neg(y < \mathsf{root}(y^2 - x^2 - 1, 1, y)) \vee (f \geq 0) \ .$$

The explanation clause states that, in order to fix the conflict under v, we must change v so as to exit the region $-1 < x < 1$ below (in y) the unit circle. This is the region in Fig 2.5 containing $(x, y) = (\frac{3}{4}, -\frac{3}{4})$, colored red.

Isolating the conflicting core. Given a constraint F incompatible with a trail M, we can compute a minimal set of constraints $\mathcal{F}$ from M that is not compatible with F by taking the constraints that that caused the inconsistency and then refine it by trying to eliminate the constraints one by one.

Example 2.9. Consider the set of polynomial constraints $\mathcal{C} = \{f_2 < 0, f_3 > 0, f_4 < 0\}$, where the polynomials f_2 and f_3 are from Ex. 2.7. The roots of these polynomials and the feasible region of $\mathcal{C}$ are depicted in Fig. 2.7. Assume the transition is in the state $\langle [\![x \mapsto 0, (f_2 < 0), (f_4 < 0), E \to (f_3 \leq 0)]\!], \mathcal{C} \rangle_2$, and we need to compute the explanation E of the last propagation. Although the propagation was based on the inconsistency of $\mathcal{C}$ under M, we can pick the subset $\{f_2 < 0, f_3 > 0\}$ to produce the explanation. It is a smaller set, but sufficient, as it is also inconsistent with M. Doing so we reduce the number of polynomials we need to project, which, in CAD settings, is always an improvement.

2.4 Related Work and Experimental Results

In addition to CAD, a number of other procedures have been developed and implemented in working tools since the 1980s, including Weispfenning's method of virtual term substitution (VTS) [99] (as implemented in Reduce/Redlog), and the Harrison-McLaughlin proof producing version of the Cohen-Hörmander method [66]. Abstract Partial Cylindrical Algebraic Decomposition [75] combines fast, sound but incomplete procedures with CAD. Tiwari [96] presents an approach using Gröbner bases and sign conditions to produce unsatisfiability witnesses for nonlinear constraints. Platzer, Quesel and Rümmer combine Gröbner bases with semidefinite programming [76] for the real Nullstellensatz.

In order to evaluate the new decision procedure we have implemented a new solver nlsat, the implementation being a clean translation of the decision procedure described in this paper. We compare the new solver to the following solvers that have been reported to perform reasonably well on fragments of non-linear arithmetic: the z3 3.2 [35], cvc3 2.4.1 [7], and MiniSmt 0.3 [102] SMT solvers; the quantifier elimination based solvers Mathematica 8.0 [88, 87], QEPCAD 1.65 [16], Redlog-CAD and Redlog-VTS [40]; and the interval based iSAT [44] solver.[5]

We ran all the solvers on several sets of benchmarks, where each benchmark set has particular characteristics that can be problematic for a non-linear solver. The meti-tarski benchmarks are proof obligations extracted from the MetiTarski project [3], where the constraints are of high degree and the polynomials represent approximations of the elementary real functions being analyzed. The keymaera benchmark set contains verification conditions from the Keymaera verification platform [76]. The zankl set of problems are the benchmarks from the QF_NRA category of the SMT-LIB library, with most problems originating from attempts to prove termination of term-rewrite systems [45]. We also have two crafted sets of benchmarks, the hong benchmarks, which are a parametrized generalization of the problem from [53], and the kissing problems that describe some classic kissing number problems, both sets containing instances of increasing dimensions.

All tests were conducted on an Intel Pentium E2220 2.4 GHz processor, with individual

[5] We ran the solvers with default settings, using the `Resolve` command of Mathematica, the `rlcad` command for Redlog-CAD, and the `rlqe` for Redlog-VTS.

Table 2.1: Experimental results.

	meti-tarski (1006)		keymaera (421)		zankl (166)		hong (20)		kissing (45)		all (1658)	
solver	solved	time (s)	solved	time (s)	solved	time (s)	solved	time (s)	solved	time (s)	solved	time (s)
nlsat	1002	343	**420**	**5**	**89**	**234**	10	170	13	95	**1534**	**849**
Mathematica	**1006**	**796**	420	171	50	366	9	208	6	29	1491	1572
QEPCAD	991	2616	368	1331	21	38	6	43	4	5	1390	4036
Redlog-VTS	847	28640	419	78	42	490	6	3	10	275	1324	29488
Redlog-CAD	848	21706	363	730	21	173	6	2	4	0	1242	22613
z3	266	83	379	1216	21	0	1	0	0	0	667	1299
iSAT	203	122	291	16	21	24	**20**	**822**	0	0	535	986
cvc3	150	13	361	5	12	3	0	0	0	0	523	22
MiniSmt	40	697	35	0	46	1370	0	0	**18**	**44**	139	2112

runs limited to 2GB of memory and 900 seconds. The results of our experimental evaluation are presented in Table 2.1. The rows are associated with the individual solvers, and columns separate the problem sets. For each problem set we write the number of problems that the solver managed to solve within the time limit, and the cumulative time (rounded) for the solved problems. A plot of solver behavior with respect to solved problems is presented in Fig 2.8. All the benchmarks, with versions corresponding to the input languages of the solvers, as well as the accompanying experimental data, are available from the authors website.[6]

The results are both revealing and encouraging. On this set of benchmarks, except for nlsat and the quantifier elimination based solvers, all other solvers that we've tried have a niche problem set where they perform well (or reasonably well), whereas on others they perform poorly. The new nlsat solver, on the other hand, is consistently one of the best solvers for each problem set, with impressive running times, and, overall manages to solve the most problems, in much less time.

[6] http://cs.nyu.edu/~dejan/nonlinear/

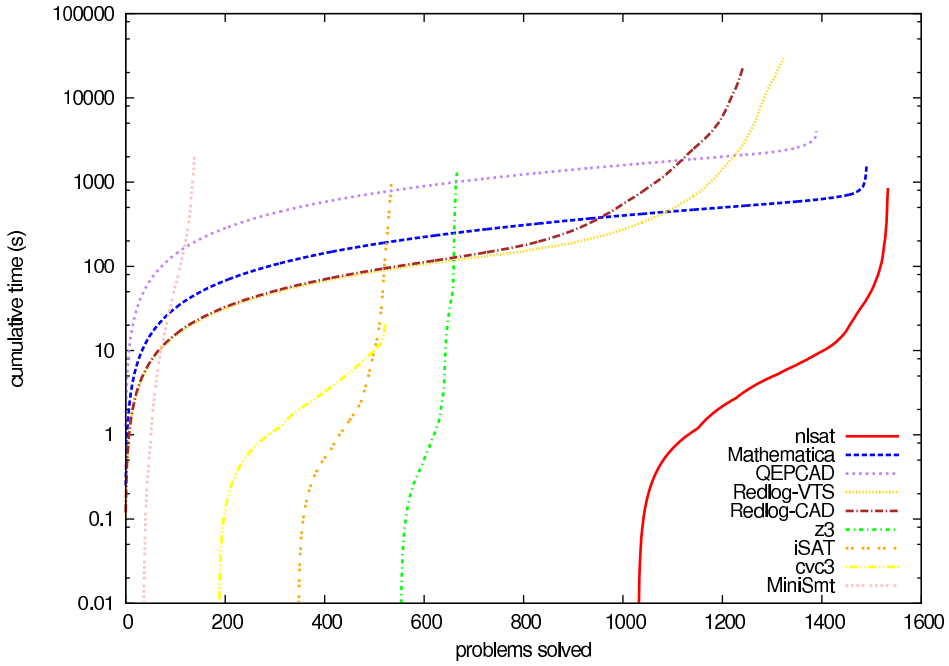


Figure 2.8: Number of problems solved by each solver against the cumulative time of the solver (logarithmic time scale).

3
Combination of Theories

Combination of theories is one of the most basic and practically important questions in SMT. When we say "combination of theories" what we are talking about is the following problem. Given two theories, where we know how decide these theories individually, is there a way to devise a decision procedure for the combined theory? For example, if we have a decision procedure for the theory of arrays and a procedure to decide the theory of linear integer arithmetic, can we then devise a procedure for the theory of arrays with integer elements. Such a combination method should be general enough to be able to combine at least the theories we encounter in practice, and at the same time aim for efficiency.

The seminal paper of Nelson and Oppen [70] introduced (in a way starting the field of SMT) such a framework for combining quantifier-free first-order theories in a modular fashion. Using the Nelson-Oppen framework, decision procedures for two individual theories can be used as black boxes to create a decision procedure for the combined theory. Although very general and widely-used in practice, the Nelson-Oppen approach is not applicable to all theories encountered in practical applications. A significant restriction of Nelson-Oppen is the requirement that theories be *stably-infinite*. While many important theories are stably-infinite, some are not, including those with inherently finite domains such as the theory of bit-vectors. As bit-precise reasoning about both programs and hardware is becoming more important and more feasible, it is desirable to find ways of overcoming this restriction.

The core idea driving the Nelson-Oppen method (and ensuring its correctness) is the exchange of equalities and disequalities over the interface variables between the theories involved in the combination. Interface variables are the problem variables that are shared by both theories, and both theories must agree on an arrangement over these variables. Most modern SMT solvers perform the search for such an arrangement by first using aggressive theory propagation to determine as much of the arrangement as possible and then relying on an efficient SAT solver to guess the rest of the arrangement, backtracking and learning lemmas as necessary [5, 12, 20]. In some cases, if the theories that are being combined have additional properties, such as convexity and/or complete and efficient equality propagation, there are more efficient ways of obtaining a

suitable arrangement. But, in general, since the number of shared variables can be substantial, guessing an arrangement over the shared variables can have an exponential impact[1] on the running time [73]. Trying to minimize the burden of non-deterministic guessing is thus of the utmost importance for a practical and efficient combination mechanism. For example, a recent model-based theory combination approach [34], in which the solver keeps a model for each theory, takes the optimistic stance of eagerly propagating all equalities that hold in the model (whether or not they are truly implied), obtaining impressive performance improvements.

In this chapter we will present theoretical results on combination of theories that resolve the mentioned limitations of the Nelson-Oppen approach, and give a new combination method that can correctly combine a wider range of theories and does so more efficiently, by allowing a reduction in the amount of non-deterministic guessing. To ensure correctness in the cases where Nelson-Oppen does not apply, we rely on the concept of *polite theories* [79]. Polite theories can be combined with an arbitrary other theory and do not require much additional work on the side of the individual theories. There are other related approaches (e.g. [95, 63]), but they require reasoning about cardinalities explicitly which is an additional computational burden we can avoid. And, while proving that a theory is polite can be difficult and needs to be done on a per-theory basis, once this is done, the combination method can be easily implemented. After resolving the theoretical limitations, we then go on to tackle the complexity overhead imposed by Nelson-Oppen. We equip the theories with an *equality propagator* and a *care function*. The role of the theory-specific equality propagator is, given a context, to propagate entailed equalities and disequalities over the interface variables. The care function, on the other hand, provides information about which variable pairs among the interface variables are important for maintaining the satisfiability of a given formula. With the information provided by these two functions we can, in many cases, drastically reduce the search space for finding a suitable arrangement.

The main result of this chapter is a reformulation of the Nelson-Oppen method that uses these two functions to decide a combination of two theories. The method can easily be adapted to the combination method for polite theories, where reducing the number of shared variables

[1] If the two theories can decided in time $O(\mathcal{T}_1(n))$ and $O(\mathcal{T}_2(n))$, the combination can be decided in $O(2^{n^2} \times (\mathcal{T}_1(n) + \mathcal{T}_2(n)))$.

is even more important (as the polite theory combination method requires extending the set of interface variables significantly).

3.1 Preliminaries

We start with a brief overview of the syntax and semantics of many-sorted first-order logic. For a more detailed exposition, we refer the reader to [42, 93].

A *signature* Σ is a triple (S, F, P) where S is a set of *sorts*, F is a set of *function symbols*, and P is a set of *predicate symbols*. For a signature $\Sigma = (S, F, P)$, we write $\Sigma^{\mathbb{S}}$ for the set S of sorts, $\Sigma^{\mathbb{F}}$ for the set F of function symbols, and $\Sigma^{\mathbb{P}}$ for the set P of predicates. Each predicate and function symbol is associated with an *arity*, a tuple constructed from the sorts in S. Functions whose arity is a single sort are called *constants*. We assume that each set of predicates P includes the equality predicates $=_\sigma$, for each sort $\sigma \in S$, where we omit the subscript σ when obvious from the context. We write $\Sigma_1 \cup \Sigma_2 = (S_1 \cup S_2, F_1 \cup F_2, P_1 \cup P_2)$ for the union of signatures $\Sigma_1 = (S_1, F_1, P_1)$ and $\Sigma_2 = (S_2, F_2, P_2)$ (with arities as in Σ_1 and Σ_2). In this thesis we always assume that, except for equality, function and predicate symbols from different theories do not overlap, so that the arities in the union are well-defined. On the other hand, two different theories are allowed to have non-disjoint sets of sorts. Additionally, we write $\Sigma_1 \subseteq \Sigma_2$ if $S_1 \subseteq S_2$, $F_1 \subseteq F_2$, $P_1 \subseteq P_2$, and the symbols of Σ_1 have the same arity as those in Σ_2.

We assume the standard notions of a Σ-*term*, Σ-*literal*, and Σ-*formula*. In the following, we assume that all formulas are quantifier-free, if not explicitly stated otherwise. A literal is called *flat* if it is of the form $x = y$, $x \neq y$, $x = f(y_1, \ldots, y_n)$, $p(y_1, \ldots, y_n)$, or $\neg p(y_1, \ldots, y_n)$, where $x, y, y_1, \ldots, y_n$ are variables, f is a function symbol, and p is a predicate symbol. If ϕ is a term or a formula, we will denote by $\mathsf{vars}_\sigma(\phi)$ the set of variables of sort σ that occur (free) in ϕ. We overload this function in the usual way, $\mathsf{vars}_S(\phi)$ denoting variables in ϕ of the sorts in S, and $\mathsf{vars}(\phi)$ denoting all variables in ϕ. If the set of free variables $\mathsf{vars}(\phi)$ in a (possibly quantified) formula ϕ is empty, we call ϕ a sentence. We also sometimes refer to a set Φ of formulas as if it were a single formula, in which case the intended meaning is the conjunction $\bigwedge \Phi$ of the formulas in the set.

Let Σ be a signature, and let X be a set of variables whose sorts are in $\Sigma^{\mathbb{S}}$. A Σ-*interpretation*

$\mathcal{A}$ over X is a map that interprets each sort $\sigma \in \Sigma^{\mathbb{S}}$ as a non-empty domain A_σ,[2] each variable $x \in X$ of sort σ as an element $x^{\mathcal{A}} \in A_\sigma$, each function symbol $f \in \Sigma^{\mathbb{F}}$ of arity $\sigma_1 \times \cdots \times \sigma_n \times \tau$ as a function $f^{\mathcal{A}} : A_{\sigma_1} \times \cdots \times A_{\sigma_n} \to A_\tau$, and each predicate symbol $p \in \Sigma^{\mathbb{P}}$ of arity $\sigma_1 \times \cdots \times \sigma_n$ as a subset $p^{\mathcal{A}}$ of $A_{\sigma_1} \times \cdots \times A_{\sigma_n}$. The equality predicate $=_\sigma$ is always interpreted as equality in the domain A_σ. A Σ-*structure* is a Σ-interpretation over an empty set of variables.

As usual, the interpretations of terms and formulas in an interpretation $\mathcal{A}$ are defined inductively over their structure. For a term t, we denote with $t^{\mathcal{A}}$ the evaluation of t under the interpretation $\mathcal{A}$. Likewise, for a formula ϕ, we denote with $\phi^{\mathcal{A}}$ the truth-value (true or false) of ϕ under interpretation $\mathcal{A}$. A Σ-formula ϕ is *satisfiable* iff it evaluates to true in some Σ-interpretation over (at least) $\mathsf{vars}(\phi)$. Let $\mathcal{A}$ be an Ω-interpretation over some set V of variables. For a signature $\Sigma \subseteq \Omega$, and a set of variables $U \subseteq V$, we denote with $\mathcal{A}^{\Sigma,U}$ the interpretation obtained from $\mathcal{A}$ by restricting it to interpret only the symbols in Σ and the variables in U.

We will use the definition of theories as classes of structures, rather than sets of sentences. We define a theory formally as follows (see e.g. [92] and Definition 2 in [79]).

Definition 3.1 (Theory). Given a set of Σ-sentences $\mathbf{Ax}$ a Σ-*theory* $T_{\mathbf{Ax}}$ is a pair $(\Sigma, \mathbf{A})$ where Σ is a signature and $\mathbf{A}$ is the class of Σ-structures that satisfy $\mathbf{Ax}$.

Given a theory $T = (\Sigma, \mathbf{A})$, a T-*interpretation* is a Σ-interpretation $\mathcal{A}$ such that $\mathcal{A}^{\Sigma,\varnothing} \in \mathbf{A}$, i.e. it if we take out the interpretation of the variables it is one of the structures belonging to the theory. A Σ-formula ϕ is T-satisfiable iff it is satisfiable in some T-interpretation $\mathcal{A}$. This is denoted as $\mathcal{A} \vDash_T \phi$, or just $\mathcal{A} \vDash \phi$ if the theory is clear from the context.

As theories in our formalism are represented by classes of structures, a combination of two theories is represented by those structures that can interpret both theories (Definition 3 in [79]).

Definition 3.2 (Combination). Let $T_1 = (\Sigma_1, \mathbf{A}_1)$ and $T_2 = (\Sigma_2, \mathbf{A}_2)$ be two theories. The *combination* of T_1 and T_2 is the theory $T_1 \oplus T_2 = (\Sigma, \mathbf{A})$ where $\Sigma = \Sigma_1 \cup \Sigma_2$ and $\mathbf{A} = \{\Sigma\text{-structures } \mathcal{A} \mid \mathcal{A}^{\Sigma_1,\varnothing} \in \mathbf{A_1} \text{ and } \mathcal{A}^{\Sigma_2,\varnothing} \in \mathbf{A_2}\}$.

The set of Σ-structures resulting from the combination of two theories is indeed a theory in the sense of Definition 3.1. If $\mathbf{Ax}_1$ is the set of sentences defining theory T_1, and $\mathbf{Ax}_2$ is

[2] In the rest of the chapter we will use the calligraphic letters $\mathcal{A}$, $\mathcal{B}$, ... to denote interpretations, and the corresponding subscripted Roman letters A_σ, B_σ, ... to denote the domains of the interpretations.

the set of sentences defining theory T_2, then $\mathbf{A}$ is the set of Σ-structures that satisfy the set $\mathbf{Ax} = \mathbf{Ax}_1 \cup \mathbf{Ax}_2$ (see Proposition 4 in [79]).

3.2 Nelson-Oppen

Given decision procedures for the satisfiability of formulas in theories T_1 and T_2, we are interested in constructing a decision procedure for satisfiability in $T_1 \oplus T_2$ using these procedures as black boxes. The Nelson-Oppen combination method [70, 92, 93] gives a general mechanism for doing this. Given a formula ϕ over the combined signature $\Sigma_1 \cup \Sigma_2$, the first step is to *purify* ϕ by constructing an equisatisfiable set of formulas $\phi_1 \cup \phi_2$ such that each ϕ_i consists of only Σ_i-formulas. This can easily be done by finding a pure (i.e. Σ_i- for some i) subterm t, replacing it with a new variable v, adding the equation $v = t$, and then repeating this process until all formulas are pure. The next step is to force the decision procedures for the individual theories to agree on whether variables appearing in both ϕ_1 and ϕ_2 (called *shared* or *interface* variables) are equal. This is done by introducing an *arrangement* over the shared variables [79, 92].

Definition 3.3 (Arrangement). Given a set of variables V over a set of sorts S, with $V_\sigma = \mathsf{vars}_\sigma(V)$ so that $V = \bigcup_{\sigma \in S} V_\sigma$, we call a formula δ_V an *arrangement* of V if there exists a family of equivalence relations $E = \{\ E_\sigma \subseteq V_\sigma \times V_\sigma \ \mid \ \sigma \in S\ \}$, such that the equivalence relations induce δ_V, i.e. $\delta_V = \bigwedge_{\sigma \in S} \delta_\sigma$, where each δ_σ is determined by E_σ as follows:

$$\delta_\sigma = \ \{\ x = y \mid (x,y) \in E_\sigma\ \} \ \cup \ \{\ x \neq y \mid (x,y) \in (V_\sigma \times V_\sigma) \setminus E_\sigma\ \}\ .$$

When the family of equivalence relations is not clear from the context, we will denote the arrangement as $\delta_V(E)$.

The Nelson-Oppen combination theorem states that ϕ is satisfiable in $T_1 \oplus T_2$ iff there exists an arrangement δ_V of the shared variables $V = \mathsf{vars}(\phi_1) \cap \mathsf{vars}(\phi_2)$ such that $\phi_i \cup \delta_V$ is satisfiable in T_i, for $i = 1, 2$. However, as mentioned earlier, some restrictions on the theories are necessary in order for the Nelson-Oppen method to be complete. Sufficient conditions for completeness are:

- the two signatures have no function or predicate symbols in common, and

- the two theories are *stably-infinite* over (at least) the set of common sorts $\Sigma_1^{\mathbb{S}} \cap \Sigma_2^{\mathbb{S}}$.

Stable-infiniteness was originally introduced in a single-sorted setting [73]. In the many-sorted setting, stable-infiniteness is defined with respect to a subset of the signature sorts (Definition 6 from [93]).

Definition 3.4 (Stable-Infiniteness). Let Σ be a signature, let $S \subseteq \Sigma^{\mathbb{S}}$ be a set of sorts, and let T be a Σ-theory. We say that T is *stably-infinite* with respect to S if for every T-satisfiable quantifier-free Σ-formula ϕ, there exists a T-interpretation $\mathcal{A}$ satisfying ϕ, such that A_σ is infinite for each sort $\sigma \in S$.

The following theorem, together with the Läwenheim-Skolem theorem and the property of stable-infiniteness, is the basis for showing that the Nelson-Oppen is correct, and we will be relying on it when proving correctness results in the rest of this chapter. It is an adaptation of Theorems 10 and 11 of [93].

Theorem 3.5. *Let T_i be a Σ_i-theory for $i = 1, 2$ such that the two theories have no function or predicate symbols in common. Let $\Sigma = \Sigma_1 \cup \Sigma_2$, $T = T_1 \oplus T_2$, and let $S = \Sigma_1^{\mathbb{S}} \cap \Sigma_2^{\mathbb{S}}$ be the set of shared sorts. Let Γ_i be a set of Σ_i-literals for $i = 1, 2$, and let $V = \mathsf{vars}(\Gamma_1) \cap \mathsf{vars}(\Gamma_2)$ be the set of variables shared between Γ_1 and Γ_2. If there exists a T_1-interpretation $\mathcal{A}$ and a T_2-interpretation $\mathcal{B}$ and an arrangement δ_V of V such that:*

- $\mathcal{A} \models_{T_1} \Gamma_1 \cup \delta_V$,
- $\mathcal{B} \models_{T_2} \Gamma_2 \cup \delta_V$, *and*
- $|A_\sigma| = |B_\sigma|$, *for all* $\sigma \in S$,

then there exists a T-interpretation $\mathcal{C}$ such that:

- $\mathcal{C} \models_T \Gamma_1 \cup \Gamma_2 \cup \delta_V$,
- $C_\sigma = A_\sigma$ *for all* $\sigma \in \Sigma_1^{\mathbb{S}}$, *and*
- $C_\sigma = B_\sigma$ *for all* $\sigma \in \Sigma_2^{\mathbb{S}} \setminus S$.

Proof. Let $V_\sigma = \mathsf{vars}_\sigma(\Gamma_1) \cap \mathsf{vars}_\sigma(\Gamma_2)$, for $\sigma \in S$. Define a family of functions $h = \{h_\sigma : V_\sigma^{\mathcal{B}} \mapsto V_\sigma^{\mathcal{A}} \mid \sigma \in S\}$ such that $h_\sigma(v^{\mathcal{B}}) = v^{\mathcal{A}}$ for each $v \in V_\sigma$. Since the interpretations $\mathcal{B}$ and $\mathcal{A}$ agree on equalities over V (by satisfying the same arrangement δ_V), the functions h_σ are well-defined and bijective. This implies that $|V_\sigma^{\mathcal{B}}| = |V_\sigma^{\mathcal{A}}|$ and, since $|B_\sigma| = |A_\sigma|$ for $\sigma \in S$, we can extend each function to a bijection $h'_\sigma : B_\sigma \mapsto A_\sigma$. Let h'_σ be the identity function for each $\sigma \in \Sigma_2^{\mathbb{S}} \setminus S$. Now, we can define a new Σ_2-interpretation $\mathcal{B}'$ (over the same set of variables as in $\mathcal{B}$) in such a way that $h' = \bigcup_{\sigma \in S} h'_\sigma$ is an isomorphism from $\mathcal{B}$ to $\mathcal{B}'$:

$$
\begin{aligned}
B'_\sigma &= \begin{cases} A_\sigma & \text{if } \sigma \in S \\ B_\sigma & \text{if } \sigma \in \Sigma_2^{\mathbb{S}} \setminus S \end{cases} \\
v^{\mathcal{B}'} &= h'(v^{\mathcal{B}}) \\
f^{\mathcal{B}'}(b_1, \ldots, b_n) &= h'(f^{\mathcal{B}}(h'^{-1}(b_1), \ldots, h'^{-1}(b_n))), \text{ and} \\
\langle b_1, \ldots, b_n \rangle \in p^{\mathcal{B}'} &\text{ iff } \left\langle h'^{-1}(b_1), \ldots, h'^{-1}(b_n) \right\rangle \in p^{\mathcal{B}}.
\end{aligned}
$$

Because h' is an isomorphism, we have $\mathcal{B}' \models \Gamma_2 \cup \delta_V$. We can now define the Σ-interpretation $\mathcal{C}$ as follows:

$$
C_\sigma = \begin{cases} A_\sigma & \text{if } \sigma \in \Sigma_1^{\mathbb{S}} \setminus S \\ A_\sigma = B'_\sigma & \text{if } \sigma \in S \\ B'_\sigma = B_\sigma & \text{if } \sigma \in \Sigma_2^{\mathbb{S}} \setminus S \end{cases}
\qquad
v^{\mathcal{C}} = \begin{cases} v^{\mathcal{A}} & \text{if } v \text{ is of sort } \sigma \in \Sigma_1^{\mathbb{S}} \setminus S \\ v^{\mathcal{A}} = v^{\mathcal{B}'} & \text{if } v \text{ is of sort } \sigma \in S \\ v^{\mathcal{B}'} & \text{if } v \text{ is of sort } \sigma \in \Sigma_2^{\mathbb{S}} \setminus S \end{cases}
$$

$$
f^{\mathcal{C}} = \begin{cases} f^{\mathcal{A}} & \text{if } f \in \Sigma_1^F \\ f^{\mathcal{B}'} & \text{if } f \in \Sigma_2^F \end{cases}
\qquad
p^{\mathcal{C}} = \begin{cases} p^{\mathcal{A}} & \text{if } p \in \Sigma_1^P \\ p^{\mathcal{B}'} & \text{if } p \in \Sigma_2^P \end{cases}
$$

Clearly, $C_\sigma = A_\sigma$ for all $\sigma \in \Sigma_1^{\mathbb{S}}$ and $C_\sigma = B_\sigma$ for all $\sigma \in \Sigma_2^{\mathbb{S}} \setminus S$. It is also easy to see from the definition above that $\mathcal{C}^{\Sigma_1, \mathsf{vars}(\Gamma_1)} = \mathcal{A}$ and $\mathcal{C}^{\Sigma_2, \mathsf{vars}(\Gamma_2)} = \mathcal{B}'$, and thus $\mathcal{C} \models \Gamma_1 \cup \Gamma_2 \cup \delta_V$. □

3.3 Polite Theories

Although many interesting theories are stably-infinite, some important theories are not. For example, the theory of fixed-size bit-vectors contains sorts whose domains are all finite. Hence, this theory cannot be stably-infinite. The Nelson-Oppen method may be incomplete for combinations involving this theory as shown by the following example.

Example 3.1. Consider the theory of arrays T_{array} where both indices and elements are of the same sort bv, so that the sorts of T_{array} are $\{\mathsf{array}, \mathsf{bv}\}$, and a theory T_{bv} that requires the sort bv to be interpreted as bit-vectors of size 1. Both theories are decidable and we would like to decide the combination theory in a Nelson-Oppen-like framework. Let $a_1, \ldots, a_5$ be array variables and consider the following constraints:

$$a_i \neq a_j, \text{ for } 1 \leq i < j \leq 5 \ .$$

These constraints are entirely within the language of T_{array} (i.e. no purification is necessary), there are no shared variables, and there are no constraints over bit-vectors. Thus, the array theory decision procedure is given all of the constraints and the bit-vector decision procedure is given an empty set of constraints. Any decision procedure for the theory of arrays will tell us that these constraints are satisfiable. But, there are only four possible different arrays with elements and indices over bit-vectors of size 1, so this set of constraints is unsatisfiable.

Polite theories were introduced in [79] to extend the Nelson-Oppen method to allow combinations with non-stably-infinite theories. A theory can be combined with any other theory (with no common function or predicate symbols) if it is *polite* with respect to the set of shared sorts. The notion of politeness depends on two other important properties: *smoothness* and *finite witnessability*. In this section, we define these terms, noting that our definition of finite witnessability differs from that given in [79] in order to fix a correctness problem in that paper (as we explain below).

First we define the smoothness property of a theory (Definition 7 from [79]).

Definition 3.6 (Smoothness). Let Σ be a signature, let $S \subseteq \Sigma^{\mathbb{S}}$ be a set of sorts, and let T be a Σ-theory. We say that T is *smooth* with respect to S if:

- for every T-satisfiable quantifier-free Σ-formula ϕ,
- for every T-interpretation $\mathcal{A}$ satisfying ϕ,
- for all choices of cardinal numbers κ_σ, such that $\kappa_\sigma \geq |A_\sigma|$ for all $\sigma \in S$,

there exists a T-interpretation $\mathcal{B}$ satisfying ϕ such that $|B_\sigma| = \kappa_\sigma$, for all $\sigma \in S$.

Recall that when a theory T is stably-infinite with respect to a sort σ and a T-interpretation exists, we can always find another T-interpretation in which the domain of σ is infinite. On the other hand, if T is smooth with respect to σ and we have a T-interpretation, then there exist interpretations in which the domain of σ can be chosen to be any larger size. Hence every theory that is smooth with respect to a set of sorts S is also stably-infinite with respect to S.

As can be seen from the proof of Theorem 3.5, being able to combine two interpretations from different theories mainly depends on the ability to bring the domains of the shared sorts to the same size. This is where stable-infiniteness helps in the Nelson-Oppen framework: it ensures that the domains of the shared sorts can have the same infinite cardinalities. Since we are interested in combining theories that may require finite domains, we need more flexibility than that afforded by stable-infiniteness. Smoothness gives us more flexibility in resizing structures upwards. This is not quite enough as we also need to ensure that the structures are small enough. Rather than attempting to resize structures downwards, we rely on the notion of *finite witnessability* which allows us to find a kind of "minimal" structure for a theory.

Definition 3.7 (Finite Witnessability). Let Σ be a signature, let $S \subseteq \Sigma^{\mathbb{S}}$ be a set of sorts, and let T be a Σ-theory. We say that T is *finitely witnessable* with respect to S if there exists a computable function, *witness*, which, for every quantifier-free Σ-formula ϕ, returns a quantifier-free Σ-formula $\psi = \mathit{witness}(\phi)$ such that

- ϕ and $(\exists \overrightarrow{w})\psi$ are T-equivalent, where $\overrightarrow{w} = \mathsf{vars}(\psi) \setminus \mathsf{vars}(\phi)$ are fresh variables;
- if $\psi \wedge \delta_V$ is T-satisfiable, for an arrangement δ_V, where V is a set of variables of sorts in S, then there exists a T-interpretation $\mathcal{A}$ satisfying $\psi \wedge \delta_V$ such that $A_\sigma = [\mathsf{vars}_\sigma(\psi \wedge \delta_V)]^{\mathcal{A}}$, for all $\sigma \in S$,

where the notation $[U]^{\mathcal{A}}$ indicates the set $\{ v^{\mathcal{A}} \mid v \in U \}$.

Both of the definitions above use an arbitrary quantifier-free formula ϕ in the definition. As shown by Proposition 11 and Proposition 12 in [80] (see Lemmas B.1, B.2 in Appendix B), it is enough to restrict ourselves to conjunctions of flat literals in the definitions. This follows in a straightforward fashion from the fact that we can always construct an equisatisfiable formula in disjunctive normal form over flat literals.

It is important to note that our definition of finite witnessability differs from the definition given in [79]. Their definition is equivalent to ours except that there is no mention of an arrangement (i.e. the formula ψ appears alone everywhere $\psi \wedge \delta_V$ appears in the definition above).[3] The reason for this is explained and illustrated in Section 3.3.1 below.

Finally, a theory that is both smooth and finitely witnessable is *polite* (Definition 9 in [79]).

Definition 3.8 (Politeness). Let Σ be a signature, let $S \subseteq \Sigma^{\mathbb{S}}$ be a set of sorts, and let T be a Σ-theory. We say that T is *polite* with respect to S if it is both smooth and finitely witnessable with respect to S.

Note that any theory is polite (stably-infinite, smooth, finitely witnessable) with respect to an empty set of sorts.

Example 3.2. The extensional theory of arrays $T_{\mathtt{arr}}$ operates over the signature $\Sigma_{\mathtt{arr}}$ that contains the sorts $\{\mathsf{array}, \mathsf{index}, \mathsf{elem}\}$ and function symbols

$$\mathsf{read} : \mathsf{array} \times \mathsf{index} \mapsto \mathsf{elem} \ , \qquad \mathsf{write} : \mathsf{array} \times \mathsf{index} \times \mathsf{elem} \mapsto \mathsf{array} \ ,$$

where read represents reading from an array at a given index, and write represents writing a given value to an array at an index. The semantics of the theory are given by the three

[3] It is worth noting that in order to prove Proposition 3.1 and Theorem B.6, below, it is sufficient to require V to be equal to $\mathsf{vars}_S(\psi_2)$ rather than letting it be an arbitrary set of variables with sorts in S. However, this more general flexibility is needed for the proofs of Lemma B.2 and Theorem B.1.

axioms:

$$\forall\, a{:}\mathsf{array}.\, \forall\, i{:}\mathsf{index}.\, \forall\, v{:}\mathsf{elem}.\ \mathsf{read}(\mathsf{write}(a,i,v),i) = v\ \ , \tag{RW1}$$

$$\forall\, a{:}\mathsf{array}.\, \forall\, i,j{:}\mathsf{index}.\, \forall\, v{:}\mathsf{elem}.\ i \neq j \rightarrow \mathsf{read}(\mathsf{write}(a,i,v),j) = \mathsf{read}(a,j)\ \ , \tag{RW2}$$

$$\forall\, a,b{:}\mathsf{array}.\ (\forall\, i{:}\mathsf{index}.\, \mathsf{read}(a,i) = \mathsf{read}(b,i)) \rightarrow a = b\ \ . \tag{EX}$$

The first two axioms above specify the behavior of read when operating over a write, and the third axiom corresponds to the extensionality, making any two arrays with the same content equal.

The flat literals of the theory are of the form $x = \mathsf{read}(a,i)$, $a = \mathsf{write}(b,i,x)$, $i = j$, $i \neq j$, $x = y$, $x \neq y$, $a = b$, $a \neq b$, where here and below we use the convention that x, y, v are variables of sort elem; i, j are variables of sort index; a, b, c are variables of sort array; and w, z are variables of any sort.

It is not hard to see that T_{array} is smooth with respect to the sorts $\{\mathsf{index}, \mathsf{elem}\}$ – any interpretation satisfying a quantifier-free formula ϕ can be extended to arbitrary cardinalities over indices and elements by adding as many additional indices and elements as we need while keeping the satisfiability of ϕ.

As for finite witnessability, it is enough to use a witness transformation that works over conjunctions of flat literals and replaces each array disequality $a \neq b$ with the conjunction of literals

$$e_1 = \mathsf{read}(a,i) \wedge e_2 = \mathsf{read}(b,i) \wedge e_1 \neq e_2\ \ ,$$

where i is a fresh variable of sort index and e_1, e_2 are fresh variables of sort elem. The witness function creates a fresh witness index i, to witness the position where a and b are different, and names those different elements e_1 and e_2.

For the detailed proof of politeness for the theory T_{array} we refer the reader to [79].

3.3.1 Finite Witnessability Revisited

A main result of [79] is a combination method for two theories, one of which is polite over the shared sorts.

Proposition 3.1 (Proposition 12 of [79]). *Let T_i be a Σ_i-theory for $i = 1, 2$ such that the two theories have no function or predicate symbols in common. Assume that T_2 is polite with respect to $S = \Sigma_1^{\mathbb{S}} \cap \Sigma_2^{\mathbb{S}}$. Also, let Γ_i be a set of Σ_i literals for $i = 1, 2$, and let $\psi_2 = witness_{T_2}(\Gamma_2)$. Finally, let $V_\sigma = \mathsf{vars}_\sigma(\psi_2)$, for each $\sigma \in S$, and let $V = \bigcup_{\sigma \in S} V_\sigma$. Then the following are equivalent:*

1. *$\Gamma_1 \cup \Gamma_2$ is $(T_1 \oplus T_2)$-satisfiable;*
2. *There exists an arrangement δ_V such that $\Gamma_1 \cup \delta_V$ is T_1-satisfiable and $\{\psi_2\} \cup \delta_V$ is T_2-satisfiable.*

Proposition 3.1 differs from the standard Nelson-Oppen theorem in its application of the witness function to Γ_2 and in that the arrangement is over *all* the variables with shared sorts in ψ_2 rather than just over the shared variables.

As mentioned above, our definition of finite witnessability (Definition 3.7 above) differs from the definition given in [79]. Without the change, Proposition 3.1 does not hold, as demonstrated by the following example.

Example 3.3. Let Σ be a signature containing no function or predicate symbols and a single sort σ. Let T_1 be a Σ-theory containing all structures such that the domain of σ has exactly one element (i.e. the structures of T_1 are those satisfying $\forall\, x\, y.\ x = y$). Similarly, let T_2 be a Σ-theory over the same sort σ containing all structures such that the domain of σ has at least two elements (i.e. axiomatized by $\exists\, x\, y.\ x \neq y$). Note that the combination of these two theories contains no structures, and hence no formula is satisfiable in $T_1 \oplus T_2$.

Theory T_2 is clearly smooth with respect to σ. To be polite, T_2 must also be finitely witnessable with respect to σ. Consider the following candidate witness function:

$$witness(\phi) \triangleq \phi \wedge w_1 = w_1 \wedge w_2 = w_2 \ ,$$

where w_1 and w_2 are fresh variables of sort σ not appearing in ϕ.

Let ϕ be a conjunction of flat Σ-literals, let $\psi = \mathit{witness}(\phi)$, and let $V = \mathsf{vars}(\psi)$. It is easy to see that the first condition for finite witnessability holds: ϕ is satisfied in a T_2 model iff $\exists\, w_1 w_2.\, \psi$ is. Now, consider the second condition according to [79] (i.e. without the arrangement). We must show that if ψ is T_2-satisfiable (in interpretation $\mathcal{B}$, say), then there exists a T_2-interpretation $\mathcal{A}$ satisfying ψ such that $A_\sigma = [V]^{\mathcal{A}}$. The obvious candidate for $\mathcal{A}$ is obtained by setting $A_\sigma = [V]^{\mathcal{B}}$ and by letting $\mathcal{A}$ interpret only those variables in V (interpreting them as in $\mathcal{B}$). Clearly $\mathcal{A}$ satisfies ψ. However, if $[V]^{\mathcal{B}}$ contains only one element, then $\mathcal{A}$ is not a T_2-interpretation. But in this case, we can always first modify the way variables are interpreted in $\mathcal{B}$ to ensure that $w_2^{\mathcal{B}}$ is different from $w_1^{\mathcal{B}}$ ($\mathcal{B}$ is a T_2-interpretation, so B_σ must contain at least two different elements). Since w_2 does not appear in ϕ, this change cannot affect the satisfiability of ψ in $\mathcal{B}$. After making this change, $[V]^{\mathcal{B}}$ is guaranteed to contain at least two elements, so we can always construct $\mathcal{A}$ as described above. Thus, the second condition for finite witnessability is satisfied and the candidate witness function is indeed a witness function according to [79].

As we will see below, however, this witness function leads to problems. Notice that according to our definition of finite witnessability, the candidate witness function is not acceptable. To see why, consider again the second condition. Let δ_V be an arrangement of V. According to our definition, we must show that if $\psi \wedge \delta_V$ is satisfied by T_2-interpretation $\mathcal{B}$, then there exists a T_2-interpretation $\mathcal{A}$ satisfying $\psi \wedge \delta_V$ such that $A_\sigma = [V]^{\mathcal{A}}$. We can consider the same construction as above, but this time, the case when $[V]^{\mathcal{B}}$ contains only one element cannot be handled as before. This is because δ_V requires $\mathcal{A}$ to preserve equalities and disequalities in V. In particular, δ_V may include $w_1 = w_2$. In this case, there is no way to construct an appropriate interpretation $\mathcal{A}$.

Now, we show what happens if the candidate witness function given above is allowed. Consider using Proposition 3.1 to check the satisfiability of $x = x$ (where x is a variable of sort σ). Although this is trivially satisfiable in any theory that has at least one structure, it is not satisfiable in $T_1 \oplus T_2$ since there are no structures to satisfy it. To apply the proposition we let

- $\Gamma_1 = \varnothing, \Gamma_2 = \{x = x\}$,
- $\psi_2 = \mathit{witness}(\Gamma_2) = (x = x \wedge w_1 = w_1 \wedge w_2 = w_2)$, and
- $V = \mathsf{vars}(\psi_2) = \{x, w_1, w_2\}$.

Proposition 3.1 allows us to choose an arrangement over the variables of V. Let

$$\delta_V = \{x = w_1, x = w_2, w_1 = w_2\}$$

be an arrangement over the variables in V. It is easy to see that $\Gamma_1 \cup \delta_V$ is satisfiable in a T_1-interpretation $\mathcal{A}$ and $\psi_2 \cup \delta_V$ is satisfiable in a T_2-interpretation $\mathcal{B}$, where $\mathcal{A}$ and $\mathcal{B}$ interpret the domains and variables as follows:

$$\sigma^{\mathcal{A}} = \{a_1\}, \sigma^{\mathcal{B}} = \{b_1, b_2\}, x^{\mathcal{A}} = w_1^{\mathcal{A}} = w_2^{\mathcal{A}} = a_1, x^{\mathcal{B}} = w_1^{\mathcal{B}} = w_2^{\mathcal{B}} = b_1 \ .$$

Thus, according to Proposition 3.1, $\Gamma_1 \cup \Gamma_2$ should be $T_1 \oplus T_2$-satisfiable, but we know that this is impossible.

Finally, consider what happens if we use a witness function for T_2 that is acceptable according to our new definition:

$$\mathit{witness}(\phi) \triangleq \phi \wedge w_1 \neq w_2 \ .$$

If we look at the same example using this witness function, we can verify that for every arrangement δ_V, either $w_1 \neq w_2 \in \delta_V$, in which case $\Gamma_1 \cup \delta_V$ is not T_1-satisfiable, or else $w_1 = w_2 \in \delta_V$, in which case $\mathit{witness}(\Gamma_2) \cup \delta_V$ is not T_2-satisfiable.

As shown by the example above, the definition of finite witnessability in [79] is not strong enough. It allows witness functions that can falsify Proposition 3.1. The changes in Definition 3.7 remedy the problem. For completeness, we include the proof of Proposition 3.1 below, adapted from [79], indicating where the proof fails if the weaker definition of finite witnessability is used.

Proof. (1 $\Rightarrow$ 2): Assume that $\Gamma_1 \cup \Gamma_2$ is $(T_1 \oplus T_2)$-satisfiable and let $\overrightarrow{v} = \mathsf{vars}(\psi_2) \setminus \mathsf{vars}(\Gamma_2)$. Since Γ_2 and $(\exists \overrightarrow{v})\psi_2$ are T_2-equivalent, it follows that $\Gamma_1 \cup \{\psi_2\}$ is also $(T_1 \oplus T_2)$-satisfiable.

We can therefore fix a $(T_1 \oplus T_2)$-interpretation $\mathcal{A}$ satisfying $\Gamma_1 \cup \{\psi_2\}$. Next, let δ_V be the arrangement of V induced by $\mathcal{A}$, that is the arrangement determined by the equivalence classes $E_\sigma = \{(x, y) \mid x, y \in V_\sigma \text{ and } x^{\mathcal{A}} = y^{\mathcal{A}}\}$, for $\sigma \in S$. By construction we have an interpretation $\mathcal{A}$ such that both $\Gamma_1 \cup \delta_V$ is T_1 satisfied and $\{\psi_2\} \cup \delta_V$ is T_2-satisfied.

$(2 \Rightarrow 1)$: Let $\mathcal{A}$ be a T_1-interpretation satisfying $\Gamma_1 \cup \delta_V$, and let $\mathcal{B}$ be a T_2-interpretation satisfying $\{\psi_2\} \cup \delta_V$. Since T_2 is finitely witnessable, we can assume without loss of generality that $B_\sigma = V_\sigma^{\mathcal{B}}$.[4] For each $\sigma \in S$ we now have that

$$\begin{aligned} |B_\sigma| &= |V_\sigma^{\mathcal{B}}| && \text{since } B_\sigma = V_\sigma^{\mathcal{B}} \ , \\ &= |V_\sigma^{\mathcal{A}}| && \text{since both } \mathcal{A} \text{ and } \mathcal{B} \text{ satisfy } \delta_V \ , \\ &\leq |A_\sigma| && \text{since } V_\sigma^{\mathcal{A}} \subseteq A_\sigma \ . \end{aligned}$$

Now we can use the smoothness of T_2 to obtain a T_2-interpretation $\mathcal{C}$ that satisfies $\{\psi_2\} \cup \delta_V$ such that $|C_\sigma| = |A_\sigma|$, for each $\sigma \in S$. Now we have all the conditions necessary to combine T_1-interpretation $\mathcal{A}$ and T_2-interpretation $\mathcal{C}$ into a $(T_1 \oplus T_2)$-interpretation $\mathcal{D}$, via Theorem 3.5,[5] $\mathcal{D}$ satisfies $\Gamma_1 \cup \{\psi_2\} \cup \delta_V$. Since Γ_2 and $(\exists \overrightarrow{v})\psi_2$ are T_2-equivalent, it follows that $\mathcal{D}$ also satisfies $\Gamma_1 \cup \Gamma_2$. □

In the same paper, the authors also prove that a number of theories are polite. We are confident that the proofs of politeness for the theories of equality, arrays, sets, and multi-sets are still correct, given the new definition. Other results in the paper (in particular the proof of politeness for the theory of lists and the proof that shiny theories are polite) have some problems in their current form. We hope to address these in future work.

3.4 New Combination Method

In this section we present a new method for combining two signature-disjoint theories. The method is based on Nelson-Oppen, but it makes equality propagation explicit and also includes a

[4] This is where the proof breaks with the original definition of finite witnessability–it is clear that in order to make this assumption and keep the satisfiability of $\{\psi_2\} \cup \delta_V$ we need to include δ_V in the definition of finite witnessability.

[5] Note that δ_V may contain more variables than those shared between Γ_1 and ψ_2, but we can still apply the theorem simply by assuming that δ_V is included among the literals of both theories.

care function for each theory, enabling a more efficient mechanism for determining equalities and dis-equalities among the shared variables. Another notable difference from the original method is that we depart from viewing the combination problem as symmetric. Instead, as in the method for combining polite theories [56, 57, 79], one of the theories is designated to take the lead in selecting which variable pairs are going to be part of the final arrangement.

Example 3.4. We will illustrate the Nelson-Oppen method and motivate the possibility of improving it's efficiency by means of an example. Suppose we would like to check the following formula ϕ for satisfiability:

$$\underbrace{\bigwedge_{1\leq i<j\leq 3} f(x_i, x_4) \neq f(x_j, x_4)}_{\phi_{\mathtt{euf}}} \wedge \underbrace{\bigwedge_{1\leq i\leq 4} (0 \leq x_i \wedge x_i \leq 1)}_{\phi_{\mathtt{lia}}} \ .$$

Assume that the variables $x_1, \ldots, x_4$ are of integer sort and f is an uninterpreted function symbol. The formula ϕ belongs to the language that combines the theory of linear integer arithmetic ($T_{\mathtt{lia}}$) and the theory of uninterpreted functions ($T_{\mathtt{euf}}$).

In order to apply the Nelson-Oppen method, we first partition the formula ϕ into $\phi_{\mathtt{euf}}$ (belonging to $T_{\mathtt{euf}}$) and $\phi_{\mathtt{lia}}$ (belonging to $T_{\mathtt{lia}}$). Note that both of these formulas are satisfiable in their corresponding theories. But, to check satisfiability in the combined theory, in accordance with Nelson-Oppen, we must find an arrangement δ of the set of shared variables $V = \{x_1, x_2, x_3, x_4\}$ such that both $\phi_{\mathtt{euf}} \cup \delta$ is satisfiable in $T_{\mathtt{euf}}$ and $\phi_{\mathtt{lia}} \cup \delta$ is satisfiable in $T_{\mathtt{lia}}$.

To check this, we must search through the 15 different ways (this is the Bell number B_4) of arranging the variables, and check each one.

$$\begin{aligned}
\delta_V^1 &= \{x_1 = x_2, x_1 = x_3, x_1 = x_4, x_2 = x_3, x_2 = x_4, x_3 = x_4\} \ , \\
\delta_V^2 &= \{x_1 \neq x_2, x_1 \neq x_3, x_1 \neq x_4, x_2 = x_3, x_2 = x_4, x_3 = x_4\} \ , \\
&\vdots \\
\delta_V^{15} &= \{x_1 \neq x_2, x_1 \neq x_3, x_1 \neq x_4, x_2 \neq x_3, x_2 \neq x_4, x_3 \neq x_4\} \ .
\end{aligned}$$

It turns out that none of these arrangements work for both theories, and we may thus conclude that the original formula ϕ is unsatisfiable. To see that ϕ is unsatisfiable, it is

enough to notice that $\phi_{\texttt{euf}}$ requires the three variables x_1, x_2, x_3 (but not x_4) to be different, but this is impossible since $\phi_{\texttt{lia}}$ requires these variables to take on values from a set of only two integers.

The important insight is that the variable x_4 is not part of the reasoning that leads to inconsistency and thus, intuitively, it should be sufficient to check only arrangements over $V' = \{x_1, x_2, x_3\}$. The number of different arrangements over V' is much smaller (the Bell number $B_3 = 5$), and thus the complexity of the search would be significantly reduced.

Note that even if the arrangements are explored incrementally, with each theory solver propagating entailed equalities (as is typically done in efficient implementations), effort will still be wasted if any case-split involving x_4 is done. Without any additional information to guide the search, there is no guarantee that this wasted effort can be avoided.

In this section we will use a more general definition of an arrangement that allows us to restrict the pairs of variables that we are interested in. We do so by introducing the notion of a care graph.

Definition 3.9 (Care Graph). Given a set V of variables, a graph $\mathbf{G} = \langle V, E \rangle$ is a *care graph* over V if $E \subseteq V \times V$ and edges exist only between variables of the same sort. A care graph is *trivial* if it contains all possible edges.

Intuitively, if an edge $(x, y) \in E$ is present in a care graph, it means that we are interested in the relationship between the variables x and y.

Definition 3.10 (Arrangement over G). Given a care graph $\mathbf{G} = \langle V, E \rangle$, we call $\delta_{\mathbf{G}}$ an *arrangement over* $\mathbf{G}$ if $\delta_{\mathbf{G}}$ is the restriction of some arrangement of V (in the sense of Definition 3.3) to the edges in $\mathbf{G}$. More precisely, $\delta_{\mathbf{G}}$ is an arrangement over $\mathbf{G}$ if there exists an arrangement δ of V such that $\delta_{\mathbf{G}} = \{\ x = y \in \delta \mid (x, y) \in E\ \} \cup \{\ x \neq y \in \delta \mid (x, y) \in E\ \}$.

To simplify the presentation (as well as the formal proofs), we present the method in its non-deterministic flavor, following the approach of [92]. We will first define the equality propagator and the care function, and then proceed to presenting and proving correctness of the combination method.

Definition 3.11 (Equality Propagator). For a Σ-theory T we call a function $\mathfrak{P}_T[\![\cdot]\!]$ an *equality propagator* for T if, for every set V of variables, it maps every set ϕ of flat Σ-literals into a set of equalities and dis-equalities between variables:

$$\mathfrak{P}_T[\![V]\!](\phi) = \{x_1 = y_1, \ldots, x_m = y_m\} \cup \{z_1 \neq w_1, \ldots, z_n \neq w_n\} \ ,$$

where $\mathsf{vars}(\mathfrak{P}_T[\![V]\!](\phi)) \subseteq V$ and

1. for each equality $x_i = y_i \in \mathfrak{P}_T[\![V]\!](\phi)$ it holds that $\phi \vDash_T x_i = y_i$;
2. for each dis-equality $z_i \neq w_i \in \mathfrak{P}_T[\![V]\!](\phi)$ it holds that $\phi \vDash_T z_i \neq w_i$;
3. $\mathfrak{P}_T[\![V]\!]$ is monotone, i.e., $\phi \subseteq \psi \implies \mathfrak{P}_T[\![V]\!](\phi) \subseteq \mathfrak{P}_T[\![V]\!](\psi)$; and
4. $\mathfrak{P}_T[\![V]\!]$ contains at least those equalities and dis-equalities, over variables in V, that appear in ϕ.

An equality propagator, given a set of theory literals, returns a set of entailed equalities and dis-equalities between the variables in V. It does not need to be complete (i.e. it does not need to return *all* entailed equalities and dis-equalities), but the more complete it is, the more helpful it is in reducing the arrangement search space (note also that for a complete propagator, properties 3 and 4 are consequences of properties 1 and 2).

When combining two theories, the combined theory can provide more equality propagation than just the union of the individual propagators. The following construction defines an equality propagator that reuses the individual propagators in order to obtain a propagator for the combined theory. This is achieved by allowing the propagators to incrementally exchange literals until a fix-point is reached.

Definition 3.12 (Combined Propagator). Let T_1 and T_2 be two theories over the signatures Σ_1 and Σ_2, equipped with equality propagators $\mathfrak{P}_{T_1}[\![\cdot]\!]$ and $\mathfrak{P}_{T_2}[\![\cdot]\!]$, respectively. Let $T = T_1 \oplus T_2$ and $\Sigma = \Sigma_1 \cup \Sigma_2$. Let V be a set of variables and ϕ a set of flat Σ-literals partitioned into a set ϕ_1 of Σ_1-literals and a set ϕ_2 of Σ_2-literals. We define the combined propagator $\mathfrak{P}_T[\![\cdot]\!]$ for the theory T as

$$\mathfrak{P}_T[\![V]\!](\phi) = (\mathfrak{P}_{T_1} \oplus \mathfrak{P}_{T_2})[\![V]\!](\phi) = \psi_1^* \cup \psi_2^* \ ,$$

where $\langle \psi_1^*, \psi_2^* \rangle$ is the least fix-point of the following operator $\mathcal{F}$

$$\mathcal{F}\langle \psi_1, \psi_2 \rangle = \langle \mathfrak{P}_{T_1}[\![V]\!](\phi_1 \cup \psi_2), \mathfrak{P}_{T_2}[\![V]\!](\phi_2 \cup \psi_1) \rangle \quad .$$

The fix-point exists as the propagators are monotone and the set V is finite. Moreover, the value of the fix-point is easily computable by iteration from $\langle \varnothing, \varnothing \rangle$. Also, it is clear from the definition that the combined propagator is at least as strong as the individual propagators, i.e., $\mathfrak{P}_{T_1}[\![V]\!](\phi_1) \subseteq \mathfrak{P}_T[\![V]\!](\phi_1) \subseteq \mathfrak{P}_T[\![V]\!](\phi)$, $\mathfrak{P}_{T_2}[\![V]\!](\phi_2) \subseteq \mathfrak{P}_T[\![V]\!](\phi_2) \subseteq \mathfrak{P}_T[\![V]\!](\phi)$.

Definition 3.13 (Care Function). For a Σ-theory T we call a function $\mathfrak{C}[\![\cdot]\!]$ a *care function* for T with respect to a T-equality propagator $\mathfrak{P}_T[\![\cdot]\!]$ when for every set V of variables and every set ϕ of flat Σ-literals

1. $\mathfrak{C}[\![V]\!]$ maps ϕ to a care graph $\mathbf{G} = \langle V, E \rangle$;
2. if $x = y$ or $x \neq y$ are in $\mathfrak{P}_T[\![V]\!](\phi)$ then $(x, y) \notin E$;
3. if $\mathbf{G} = \langle V, \varnothing \rangle$ and ϕ is T-satisfiable then, for any arrangement δ_V such that $\mathfrak{P}_T[\![V]\!](\phi) \subseteq \delta_V$, it holds that $\phi \cup \delta_V$ is also T-satisfiable.

The main feature of a care function is that when it returns an empty graph, this guarantees that ϕ can be satisfied regardless of the relationships between the remaining variables. In other cases, notice that the definition does not specify (beyond requirement 2) anything about what the care graph should contain. This is because no additional conditions are required for correctness. However, to be effective, a care function should return a care graph which (to the extent that is efficiently possible) contains only edges corresponding to pairs of variables which, if set equal or dis-equal, could affect the satisfiability of the formula ϕ.

Example 3.5. For any Σ-theory T and a set of variables V, the *trivial care function* $\mathfrak{C}_0[\![\cdot]\!]$ is the one that maps a set of variables to a maximal care graph, i.e.,

$$\mathfrak{C}_0[\![V]\!](\phi) = \langle V, E_0 \rangle\,, \text{ where } (x, y) \in E_0 \text{ iff } \begin{cases} x \in V, y \in V, \\ x \text{ and } y \text{ have the same sort}, \\ x = y \notin \mathfrak{P}_T[\![V]\!](\phi), \text{ and} \\ x \neq y \notin \mathfrak{P}_T[\![V]\!](\phi) \quad . \end{cases}$$

Notice that $\mathfrak{C}_0[\![\cdot]\!]$ trivially satisfies the conditions of Definition 3.13 with respect to any equality propagator. To see this, the only case to consider is when the care graph returned has no edges and ϕ is satisfiable. If the care graph is empty that means that the arrangement is fully implied by the propagator, i.e. for any arrangement δ_V such that $\mathfrak{P}_T[\![V]\!](\phi) \subseteq \delta_V$, we must have that $\mathfrak{P}_T[\![V]\!](\phi) = \delta_V$. Since the propagator only produces implied facts, and ϕ is satisfiable, it is clear that $\phi \cup \delta_V$ is also satisfiable.

3.4.1 Combination Method

Let T_1 be a Σ_1-theory and T_2 be a Σ_2-theory, and let $S = \Sigma_1^{\mathbb{S}} \cap \Sigma_2^{\mathbb{S}}$ be the set of common sorts. Further, assume that each T_i is stably-infinite with respect to S_i, decidable, and equipped with an equality propagator $\mathfrak{P}_{T_i}[\![\cdot]\!]$. Additionally, let T_2 be equipped with a care function $\mathfrak{C}_{T_2}[\![\cdot]\!]$ operating with respect to the propagator $\mathfrak{P}_{T_2}[\![\cdot]\!]$. We are interested in deciding the combination theory $T = T_1 \oplus T_2$ over the signature $\Sigma = \Sigma_1 \cup \Sigma_2$. We denote the combined theory propagator with $\mathfrak{P}_T[\![\cdot]\!]$.The combination method takes as input a set ϕ of Σ-literals and consists of the following steps:

Purify: The output of the purification phase is two new sets of literals, ϕ_1 and ϕ_2 such that $\phi_1 \cup \phi_2$ is equisatisfiable (in T) with ϕ and each literal in ϕ_i is a flat Σ_i-literal, for $i = 1, 2$. This step is identical to the first step in the standard Nelson-Oppen combination method.

Arrange: Let $V = \mathsf{vars}(\phi_1) \cap \mathsf{vars}(\phi_2)$ be the set of all variables shared by ϕ_1 and ϕ_2, and let δ_V be an arrangement (chosen non-deterministically) of V. Let the care graph $\mathbf{G}_2$ be a fix-point of the following operator:

$$\mathcal{G}\langle \mathbf{G} \rangle = \mathbf{G} \cup \mathfrak{C}_{T_2}[\![V]\!](\phi_2 \cup \mathfrak{P}_T[\![V]\!](\phi_1 \cup \phi_2 \cup \delta_{\mathbf{G}})) \ , \tag{3.1}$$

where $\delta_{\mathbf{G}}$ is the arrangement over $\mathbf{G}$ obtained by restricting δ_V to the edges in $\mathbf{G}$.

Check: Check the following formulas for satisfiability in T_1 and T_2 respectively

$$\phi_1 \cup \mathfrak{P}_T[\![V]\!](\phi_1 \cup \phi_2 \cup \delta_{\mathbf{G}_2}) \ , \qquad \phi_2 \cup \mathfrak{P}_T[\![V]\!](\phi_1 \cup \phi_2 \cup \delta_{\mathbf{G}_2}) \ .$$

If both are satisfiable, output satisfiable, otherwise output unsatisfiable.

Notice that above, since the graph is finite, and the operator $\mathcal{G}$ is increasing, a fix-point always exists. Moreover, it is in our interest to choose a minimal such fix-point, which we can obtain by doing a fix-point iteration starting from $\mathbf{G}_0 = \langle V, \varnothing \rangle$ ($\mathcal{G}$ preserves the property of being a care graph). Another important observation is that for any fix-point $\mathbf{G}_2$ (with respect to the chosen value of δ_V) of the operator $\mathcal{G}$ above, we must have that the care function application in (3.1) returns an empty graph. This follows from the fact that $\mathbf{G}_2$ is a fix-point and the observation that the propagator must return all the equalities and dis-equalities from $\delta_{\mathbf{G}}$, by definition, and the care function then must ignore them, also by definition.

Example 3.6. Consider the case of combining two theories T_1 and T_2 equipped with trivial care functions and propagators $\mathfrak{P}_{T_i}[\![V]\!]$ that simply return those input literals that are either equalities or dis-equalities over variables in V. Assume that ϕ_1 and ϕ_2 are the outputs of the purification phase, and let V be the set of variables shared by ϕ_1 and ϕ_2. Let δ_V be the arrangement of V chosen in the **Arrange** phase, and let $\mathbf{G}_2$ be the fix-point graph. Since $\mathfrak{C}_{T_2}[\![\cdot]\!]$ is a trivial care function, the arrangement $\delta_{\mathbf{G}_2}$ over $\mathbf{G}_2$ must be equivalent to δ_V. Then, since the equality propagators simply keep the input equalities and dis-equalities over V, and all relationships between variables in V are determined by $\delta_{\mathbf{G}_2}$, the combined propagator will simply return δ_V and we will thus check $\phi_1 \cup \delta_V$ and $\phi_2 \cup \delta_V$ for satisfiability in the **Check** phase. This shows that our method can effectively simulate the standard Nelson-Oppen combination method.

We now show the correctness of the method.

Theorem 3.14. *Let T_i be a Σ_i-theory, stably-infinite with respect to the set of sorts S_i, and equipped with equality propagator $\mathfrak{P}_{T_i}[\![\cdot]\!]$, for $i = 1, 2$. Additionally, let T_2 be equipped with a care function $\mathfrak{C}_{T_2}[\![\cdot]\!]$ operating with respect to $\mathfrak{P}_{T_2}[\![\cdot]\!]$. Let $\Sigma = \Sigma_1 \cup \Sigma_2$, $T = T_1 \oplus T_2$ and let ϕ be a set of flat Σ-literals, which can be partitioned into a set ϕ_1 of Σ_1-literals and a set ϕ_2 of Σ_2-literals, with $V = \mathsf{vars}(\phi_1) \cap \mathsf{vars}(\phi_2)$. If $\Sigma_1^{\mathbb{S}} \cap \Sigma_2^{\mathbb{S}} = S_1 \cap S_2$, then the following are equivalent:*

1. *ϕ is T-satisfiable;*

2. *there exists a care graph $\mathbf{G}_2$ and an arrangement $\delta_{\mathbf{G}_2}$ over $\mathbf{G}_2$ such that $\mathbf{G}_2$ is a fix-point solution of* (3.1)*, and such that the following sets are T_1- and T_2-satisfiable respectively:*

$$\phi_1 \cup \mathfrak{P}_T[\![V]\!](\phi_1 \cup \phi_2 \cup \delta_{\mathbf{G}_2}) \ , \qquad \phi_2 \cup \mathfrak{P}_T[\![V]\!](\phi_1 \cup \phi_2 \cup \delta_{\mathbf{G}_2}) \ .$$

Moreover, T is stably-infinite with respect to $S_1 \cup S_2$.

Proof. (1) $\Rightarrow$ (2) : Suppose $\phi = \phi_1 \cup \phi_2$ is T-satisfiable in a T-interpretation $\mathcal{A}$. Let δ_V be the arrangement of V satisfied by $\mathcal{A}$, and let $\mathbf{G}_2$ be the trivial care graph over V. It is easy to see that $\mathbf{G}_2$ is a fix-point solution of (3.1) (with respect to arrangement δ_V), and that $\delta_{\mathbf{G}_2} \subseteq \delta_V$. Then, because $\mathcal{A}$ satisfies ϕ_1, ϕ_2, and δ_V, and the propagator only adds formulas that are entailed, it is clear that $\mathcal{A}$ satisfies both sets of formulas, which proves one direction.

(2) $\Leftarrow$ (1) : Assume that there is a T_1-interpretation $\mathcal{A}_1$ and a T_2-interpretation $\mathcal{A}_2$ (and assume wlog that both interpret all the variables in V) such that

$$\mathcal{A}_1 \vDash_{T_1} \phi_1 \cup \mathfrak{P}_T[\![V]\!](\phi_1 \cup \phi_2 \cup \delta_{\mathbf{G}_2}) \ ,$$
$$\mathcal{A}_2 \vDash_{T_2} \phi_2 \cup \mathfrak{P}_T[\![V]\!](\phi_1 \cup \phi_2 \cup \delta_{\mathbf{G}_2}) \ .$$

Let δ_V be the arrangement of V satisfied by $\mathcal{A}_1$, so

$$\delta_{\mathbf{G}_2} \subseteq \mathfrak{P}_{T_2}[\![V]\!](\phi_2 \cup \delta_{\mathbf{G}_2}) \subseteq \mathfrak{P}_T[\![V]\!](\phi_1 \cup \phi_2 \cup \delta_{\mathbf{G}_2}) \subseteq \delta_V \ .$$

Because $\mathbf{G}_2$ is a fix-point, we know that $\mathfrak{C}_{T_2}[\![V]\!](\phi_2 \cup \mathfrak{P}_T[\![V]\!](\phi_1 \cup \phi_2 \cup \delta_{\mathbf{G_2}})) = \langle V, \varnothing \rangle$. We then know, by property 3 of the care function, that there is a T_2-interpretation $\mathcal{B}_2$ such that $\mathcal{B}_2 \vDash_{T_2} \phi_2 \cup \delta_V$. Since δ_V is an arrangement of the shared variables and we also have that $\mathcal{A}_1 \vDash_{T_1} \phi_1 \cup \delta_V$, we can now appeal to the correctness of the standard Nelson-Oppen combination method (Theorem 3.5) to obtain a T-interpretation $\mathcal{C}$ that satisfies $\phi_1 \cup \phi_2 = \phi$. The proof that the combined theory is stably-infinite can be found in Appendix B. □

3.4.2 Extension to Polite Combination

The method described in Section 3.4 relies on the correctness argument for the standard Nelson-Oppen method, meaning that the theories involved should be stably-infinite. We now show that

the method can easily be adapted to combination of polite theories. Assuming that the theory T_2 is polite with respect to a set of sorts S_2 with $\Sigma_1^{\mathbb{S}} \cap \Sigma_2^{\mathbb{S}} \subseteq S_2$, and is equipped with a witness function $witness_2$. We modify the combination method of Section 3.4.1 as follows:

1. In the **Arrange** and **Check** phases, instead of using ϕ_2, we use the formula produced by the witness function, i.e. $\phi_2' = witness_2(\phi_2)$.

2. We define $V = \mathsf{vars}_S(\phi_2')$ instead of $V = \mathsf{vars}(\phi_1) \cap \mathsf{vars}(\phi_2)$.

Theorem 3.15. *Let T_i be a Σ_i-theory polite with respect to the set of sorts S_i, and equipped with equality propagator $\mathfrak{P}_{T_i}[\![\cdot]\!]$, for $i = 1, 2$. Additionally, let T_2 be equipped with a care function $\mathfrak{C}_{T_2}[\![\cdot]\!]$ operating with respect to $\mathfrak{P}_{T_2}[\![\cdot]\!]$. Let $\Sigma = \Sigma_1 \cup \Sigma_2$, $T = T_1 \oplus T_2$ and let ϕ be a set of flat Σ-literals, which can be partitioned into a set ϕ_1 of Σ_1-literals and a set ϕ_2 of Σ_2-literals. Let $\phi_2' = witness_{T_2}(\phi_2)$ and $V = \mathsf{vars}_S(\phi_2')$. If $\Sigma_1^{\mathbb{S}} \cap \Sigma_2^{\mathbb{S}} \subseteq S_2$, then following are equivalent*

1. *ϕ is T-satisfiable;*

2. *there exists a care-graph $\mathbf{G}_2$ and arrangement $\delta_{\mathbf{G}_2}$, fix-point solutions of (3.1), such that the following sets are T_1- and T_2-satisfiable respectively*

$$\phi_1 \cup \mathfrak{P}_T[\![V]\!](\phi_1 \cup \phi_2' \cup \delta_{\mathbf{G}_2}) \ , \qquad \phi_2' \cup \mathfrak{P}_T[\![V]\!](\phi_1 \cup \phi_2' \cup \delta_{\mathbf{G}_2}) \ .$$

Moreover, T is polite with respect to $S_1 \cup (S_2 \setminus \Sigma_1^{\mathbb{S}})$.

Proof. The proof is identical to the one given in Theorem 3.14 for the case of stably-infinite theories, except that in the last step, instead of relying on the correctness of the standard Nelson-Oppen method, we rely on the correctness of the method for combination of polite theories (Theorem 3.1). The proof that the combined theory is polite can be found in Appendix B. □

3.5 Theory of Uninterpreted Functions

The theory of uninterpreted functions, over a signature $\Sigma_{\mathtt{euf}}$, is the theory $T_{\mathtt{euf}} = (\Sigma_{\mathtt{euf}}, \mathbf{A})$, where $\mathbf{A}$ is simply the class of all $\Sigma_{\mathtt{euf}}$-structures. Conjunctions of literals in this theory can be decided for satisfiability in polynomial time by congruence closure algorithms (e.g. [83, 38]). We

make use of insights from these algorithms in defining both the equality propagator and the care function. For simplicity, we assume $\Sigma_{\texttt{euf}}$ contains no predicate symbols, but the extension to the case with predicate symbols is straightforward.

3.5.1 Equality Propagator

Let ϕ be a set of flat literals and V a set of variables. We write $\mathcal{E}(\phi)$ to denote the smallest equivalence relation over the terms occurring in ϕ containing $\{(x,t) \mid x = t \in \phi\}$. We also write $\mathcal{E}_c(\phi)$ for the smallest congruence relation[6] over terms in ϕ containing the same base set $\{(x,t) \mid x = t \in \phi\}$. We define a dis-equality (modulo congruence) relation $\mathcal{N}_c(\phi)$ as the smallest relation satisfying

$$(x,x') \in \mathcal{E}_c(\phi) \text{ and } (y,y') \in \mathcal{E}_c(\phi) \text{ and } x' \neq y' \in \phi \Longrightarrow (x,y) \in \mathcal{N}_c(\phi) \ .$$

Now, we define the equality propagator as

$$\begin{aligned} \mathfrak{P}_{\texttt{euf}}[\![V]\!](\phi) = \{\ x = y \mid x,y \in V,\ (x,y) \in \mathcal{E}_c(\phi)\ \} \\ \cup \{\ x \neq y \mid x,y \in V,\ (x,y) \in \mathcal{N}_c(\phi)\ \} \ . \end{aligned}$$

It is easy to see that $\mathfrak{P}_{\texttt{euf}}[\![\cdot]\!]$ is indeed an equality propagator. Moreover, it can easily be implemented as part of a decision procedure based on congruence closure.

Example 3.7. Given the set of flat literals $\phi = \{\ x = z,\ y = f(a),\ z \neq f(a)\ \}$, the equality propagator would return

$$\mathfrak{P}_{\texttt{euf}}[\![x,y]\!](\phi) = \{\ x = x,\ y = y,\ x \neq y,\ y \neq x\ \} \ .$$

In practice, of course, an implementation of such a propagator does not need to bother with the propagation of trivial equalities, such as $x = x$, or symmetric versions of the same (dis-)equality.

[6] In this context, a congruence relation is an equivalence relation that also satisfies the congruence property: if $f(x_1,\ldots,x_n)$ and $f(y_1,\ldots,y_n)$ are terms in ϕ, and if for each $1 \leq i \leq n$, $(x_i,y_i) \in \mathcal{E}_c(\phi)$, then $(f(x_1,\ldots,x_n), f(y_1,\ldots,y_n)) \in \mathcal{E}_c(\phi)$.

3.5.2 Care Function

We will define the care function based on the observation that, during construction of the congruence closure, we only care about equalities between pairs of variables that occur as (or are implied to be equal to) an argument of the same function symbol, in the same position. Again, let V be a set of variables and let ϕ be a set of flat literals. Assume also that ϕ only contains function symbols from $F = \{f_1, f_2, \ldots, f_n\} \subseteq \Sigma^{\mathbb{F}}_{\texttt{euf}}$.

Each function symbol $f \in F$ of arity $\sigma_1 \times \sigma_2 \times \cdots \times \sigma_k \mapsto \sigma$ will contribute to the care function with it's own set of pairs that are important. Let therefore E_f be a set containing pairs of variables from V that could trigger an application of congruence over the function f. More precisely, a pair of shared variables (x, y) from V is in the set E_f iff:

1. relationship between x and y is not yet known, i.e. $(x, y) \notin \mathcal{E}_c(\phi) \cup \mathcal{N}_c(\phi)$;

2. there exist terms $t_1 = f(x_1, \ldots, x_i, \ldots, x_k)$ and $t_2 = f(y_1, \ldots, y_i, \ldots, y_k)$ in ϕ that are not know to be equal, i.e. $(t_1, t_2) \notin \mathcal{E}_c(\phi)$, and

   (a) for some $1 \leq i \leq k$, $(x, x_i) \in \mathcal{E}_c(\phi)$ and $(y, y_i) \in \mathcal{E}_c(\phi)$;

   (b) for $1 \leq j \leq k$, variables x_j and y_j are not known to be disequal, i.e. $(x_j, y_j) \notin \mathcal{N}_c(\phi)$.

Having a set E_f for each function symbol f from the formula ϕ, we can now define $E = \bigcup_{f \in F} E_f$, and define the care function as $\mathfrak{C}_{\texttt{euf}}[\![V]\!](\phi) = \langle V, E \rangle$.

Example 3.8. Consider the following sets of literals

$$\phi_1 = \{\ f(x_1) \neq f(y_1),\ y_1 = x_2\ \}\ ,$$
$$\phi_2 = \{\ z_1 = f(x_1),\ z_2 = f(y_1),\ g(z_1, x_2) \neq g(z_2, y_2)\ \}\ ,$$
$$\phi_3 = \{\ y_1 = f(x_1),\ y_2 = f(x_2),\ z_1 = g(x_1),\ z_2 = g(x_2),\ h(y_1) \neq h(z_1)\ \}\ .$$

and corresponding sets of shared variables

$$V_1 = \{x_1, x_2\}\ , \qquad V_2 = \{x_1, x_2, y_1, y_2\}\ , \qquad V_3 = \{x_1, x_2, y_2, z_2\}\ .$$

Each of the individual formulas ϕ_1, ϕ_2, and ϕ_3 are satisfiable in $T_{\texttt{euf}}$. Let us examine each of them in turn, where for the sake of brevity, during the computation, we only report the non-reflexive, non-symmetric versions of the propagated equalities and relations, and skip the clearly irrelevant ones.

The congruence relations corresponding to the formula ϕ_1 are the following

$$\mathcal{E}_c(\phi_1) = \{\ (x_2, y_1)\ \}\ \ , \qquad\qquad \mathcal{N}_c(\phi_1) = \{\ (f(x_1), f(y_1))\ \}\ \ .$$

Note that, since y_1 is not a shared variable ($y_1 \notin V_1$), the $T_{\texttt{euf}}$ propagator will return $\mathfrak{P}_{\texttt{euf}}[\![V_1]\!](\phi_1) = \varnothing$. Nevertheless, the variable y_1 does occur as an argument of f and since x_2 is a shared variable with $(y_1, x_2) \in \mathcal{E}_c(\phi_1)$, we do consider x_2 important as it is an alias for y_1. We now have two terms $f(x_1)$ and $f(y_1)$ that are not congruent in $\mathcal{E}_c(\phi_1)$ and, to make sure that we can ensure satisfiability, we need to know the relation between x_1 and x_2. By definition, the care function therefore returns the care graph $\mathbf{G}_1 = \langle V_1, \{(x_1, x_2)\}\rangle$.

Moving on to ϕ_2, let's assume that, in addition to ϕ_2, we are also considering the integer arithmetic formula $\phi_2^{\texttt{lia}} = \{x_1 \leq y_1, x_1 \geq y_1, x_2 \leq y_2\}$, thus checking $\phi_2 \wedge \phi_2^{\texttt{lia}}$ for satisfiability in the combined theory $T = T_{\texttt{euf}} \oplus T_{\texttt{lia}}$. The congruence relations for ϕ_2 alone are the following

$$\mathcal{E}_c(\phi_2) = \{\ (z_1, f(x_1)), (z_2, f(y_1))\ \}\ \ , \qquad \mathcal{N}_c(\phi_2) = \{\ (g(z_1, x_2), g(z_2, y_2))\ \}\ \ .$$

Again, from ϕ_2 alone we can not conclude any relationship between the shared variables so, at this point, the propagator will return $\mathfrak{P}_{\texttt{euf}}[\![V_2]\!](\phi_2) = \varnothing$. But, since we are combining two theories, we will compute the combined propagator by exchanging the propagated equalities and dis-equalities. The propagator for the theory of integer arithmetic might be sophisticated enough to propagate $\mathfrak{P}_{\texttt{lia}}[\![V_2]\!](\phi_2^{\texttt{lia}}) = \{x_1 = y_1\}$. If so, this new equality is appended to ϕ_2, obtaining $\phi_2^1 = \phi_2 \cup \{x_1 = y_1\}$ and processed further. With the new information, we can obtain the stronger equivalence relations

$$\begin{aligned}\mathcal{E}_c(\phi_2^1) &= \{\ (z_1, f(x_1)), (z_2, f(y_1)), (x_1, y_1), (f(x_1), f(y_1)), (z_1, z_2)\ \}\ \ ,\\ \mathcal{N}_c(\phi_2^1) &= \{\ (g(z_1, x_2), g(z_2, y_2))\ \}\ \ .\end{aligned}$$

The stronger equivalence relations then influence the $T_{\texttt{euf}}$ propagator to return $\mathfrak{P}_{\texttt{euf}}[\![V_2]\!](\phi_2^1) = \{x_1 = y_1\}$ and, in fact, this is the final result of the combined theory propagator $\mathfrak{P}_T[\![V_2]\!](\phi_2^{\texttt{lia}}) = \{x_1 = y_1\}$.

We can now proceed with computation of the care graph. In the set of shared variables, the pairs (x_1, y_1) and (x_2, y_2) appear as arguments of function terms that could become equal, at the same position. But, we already know the relationship between x_1 and y_1, so the only pair that we are interested is the pair (x_2, y_2), i.e. the care graph we compute is $\mathbf{G}_2 = \langle V_2, \{(x_2, y_2)\}\rangle$. If we now choose the arrangement $\delta_{\mathbf{G}_2} = \{x_2 \neq y_2\}$, the pair $\langle \mathbf{G}_2, \delta_{\mathbf{G}_2}\rangle$ will in fact be a fix-point solution to (3.1), and we check the following formulas for satisfiability

$$T_{\texttt{euf}} : \phi_2 \wedge x_1 = y_1 \wedge x_2 \neq y_2$$
$$T_{\texttt{lia}} : \phi_2^{\texttt{lia}} \wedge x_1 = y_1 \wedge x_2 \neq y_2$$

Both of these formulas are satisfiable, and we can conclude that the original formula $\phi_2 \wedge \phi_2^{\texttt{lia}}$ is satisfiable in the combined theory.

Finally, let's consider the formula ϕ_3 and its $T_{\texttt{lia}}$ counterpart $\phi_3^{\texttt{lia}} = \{x_1 + x_2 + y_2 + z_2 \leq 1\}$. The equivalence relations corresponding to ϕ_3 are

$$\mathcal{E}_c(\phi_3) = \{\ (y_1, f(x_1)), (y_2, f(x_2)), (z_1, g(x_1)), (z_2, g(x_2))\ \}\ ,$$
$$\mathcal{N}_c(\phi_3) = \{\ (h(y_1), h(z_1))\ \}\ .$$

In this case, none of the individual theory propagators provide us with any additional information about the shared variables, so the combined propagator returns $\mathfrak{P}_T[\![V_3]\!](\phi_3 \wedge \phi_3^{\texttt{lia}}) = \varnothing$. We proceed to compute the care graph by noting that the only pair of shared variables under the same function symbol are x_1 and x_2, and so the care graph we compute is $\mathbf{G}_3^1 = \langle V_3, \{(x_1, x_2)\}\rangle$. We can now choose the arrangement $\delta_{\mathbf{G}_3^1} = \{x_1 = x_2\}$. As this pair is still not a fix-point of (3.1), we continue with the computation, by considering $\phi_3^1 = \phi_3 \cup \{x_1 = x_2\}$.

With the new information, the stronger equivalence relations are

$$\mathcal{E}_c(\phi_3^1) = \mathcal{E}_c(\phi_3) \cup \{\ (y_1, y_2), (z_1, z_2), (f(x_1), f(x_2)), (g(x_1), g(x_2))\ \}\ ,$$
$$\mathcal{N}_c(\phi_3^1) = \{\ (h(y_1), h(z_1))\ \}\ .$$

We are now in a similar situation as when considering ϕ_1. The variables y_1 and z_1 are not shared, but under the equivalence relation $\mathcal{E}_c(\phi_3^1)$ are in fact aliases for the shared variables y_2 and z_2, respectively. Since x_1 and z_1 appear as arguments of h in the formula, we now care about the pair (y_2, z_2), so we add it to the care graph to obtain $\mathbf{G}_3^2 = \langle V_3, \{(x_1, x_2), (y_2, z_2)\}\rangle$. We can now choose that y_2 and z_2 are different obtaining the arrangement $\delta_{\mathbf{G}_3^2} = \langle V_3, \{x_1 = x_2, y_2 \neq z_2\}\rangle$. This pair is a fix-point of (3.1), so we check the following formulas for satisfiability in the individual theories

$$T_{\mathtt{euf}} : \phi_3 \wedge x_1 = x_2 \wedge y_2 \neq z_2$$
$$T_{\mathtt{lia}} : \phi_3^{\mathtt{lia}} \wedge x_1 = x_2 \wedge y_2 \neq z_2$$

With both of the formulas satisfiable, we conclude that $\phi_3 \wedge \phi_3^{\mathtt{lia}}$ is satisfiable in the combined theory T.

We prove correctness of the care function for $\mathfrak{C}_{\mathtt{euf}}[\![\cdot]\!]$ by relying on the following well-known proposition.

Proposition 3.2. *A set ϕ of flat literals is $T_{\mathtt{euf}}$-satisfiable if and only if, for each dis-equality $x \neq y \in \phi$, we have that $(x, y) \notin \mathcal{E}_c(\phi)$.*

Theorem 3.16. *Let $T_{\mathtt{euf}}$ be the theory of uninterpreted functions with equality over the signature $\Sigma_{\mathtt{euf}}$. $\mathfrak{C}_{\mathtt{euf}}[\![\cdot]\!]$ is a care function for $T_{\mathtt{euf}}$ with respect to the equality propagator $\mathfrak{P}_{\mathtt{euf}}[\![\cdot]\!]$.*

Proof. Proving the first two properties of the care function is an easy exercise as they hold directly by construction. To show the third property let ϕ be a satisfiable set of flat literals, V a set of variables, and $\mathbf{G} = \mathfrak{C}_{\mathtt{euf}}[\![V]\!](\phi) = \langle V, \varnothing\rangle$. Suppose that, as required, we are given an arrangement δ_V, corresponding to an equivalence relation E_V, with $\mathfrak{P}_{\mathtt{euf}}[\![V]\!](\phi) \subseteq \delta_V$. We must

show that any such arrangement does not affect the satisfiability of ϕ, i.e. that $\phi \wedge \delta_V$ is also $T_{\texttt{euf}}$-satisfiable. We know, by Proposition 3.2 above, that if $x \neq y \in \phi$, then $(x, y) \notin \mathcal{E}_c(\phi)$. In order to prove the theorem it suffices to show that

$$\text{if } x \neq y \in \phi \cup \delta_V \text{ then } (x, y) \notin \mathcal{E}_c(\phi \cup \delta_V) \ .$$

First, it is clear that, since δ_V is compatible with the propagated equalities from ϕ, we must have that

$$\text{if } v, w \in V \text{ and } (v, w) \in \mathcal{E}_c(\phi) \text{ then } (v, w) \in E_V \ . \tag{3.2}$$

We now show that that even a stronger property holds, namely that

$$\text{if } v, w \in V \text{ and } (v, w) \in \mathcal{E}(\mathcal{E}_c(\phi) \cup E_V) \text{ then } (v, w) \in E_V \ . \tag{3.3}$$

Let's assume the opposite, that $v, w \in V$ and $(v, w) \in \mathcal{E}(\mathcal{E}_c(\phi) \cup E_V)$ but $(v, w) \notin E_V$. We know by (3.2) that it then must be that $(v, w) \notin \mathcal{E}_c(\phi)$. Therefore, this new equivalence must have resulted from some transitive chain from v to w that uses pairs from both $\mathcal{E}_c(\phi)$ and E_V. Let $(t_0, t_1), (t_1, t_2), \ldots, (t_{n-1}, t_n)$ be the smallest such chain (with $v = t_0$ and $w = t_n$). Let (t_i, t_{i+1}) be the first pair such that $t_i \in V$ but $t_{i+1} \notin V$. Such a pair must exist since, by (3.2), every pair in $\mathcal{E}_c(\phi) \cap (V \times V)$ is also in E_V, so that if $t_k \in V$ for all k, we would have $(t_0, t_n) \in E_V$. Then, let (t_j, t_{j+1}) be first pair such that $j > i$ and $t_j \notin V$ and $t_{j+1} \in V$ (there must be such a pair since $t_n \in V$). Notice that every pair from (t_i, t_{i+1}) to (t_j, t_{j+1}) must be in $\mathcal{E}_c(\phi)$ since each contains a term not in V. But then $(t_i, t_{j+1}) \in \mathcal{E}_c(\phi)$ which contradicts the assumption that $(t_0, t_1), (t_1, t_2), \ldots, (t_{n-1}, t_n)$ is the smallest chain from t_0 to t_n. This establishes (3.3).

Additionally, the following property follows

$$\text{if } (s, t) \in \mathcal{E}(\mathcal{E}_c(\phi) \cup E_V) \text{ then } (s, t) \notin \mathcal{N}_c(\phi) \ . \tag{3.4}$$

Suppose $(s, t) \in \mathcal{E}(\mathcal{E}_c(\phi) \cup E_V)$ and $(s, t) \in \mathcal{N}_c(\phi)$. Because ϕ is satisfiable, we know that $(s, t) \notin \mathcal{E}_c(\phi)$. So, as above, there must be some transitive chain from s to t using pairs from both $\mathcal{E}_c(\phi)$ and E_V. Let v be the first term in this chain such that $v \in V$ and w the last term in the chain such that $w \in V$. By definition, we must have $(v, w) \in \mathcal{N}_c(\phi)$. It follows that $v \neq w \in \delta_V$ and thus $(v, w) \notin E_V$. But by (3.3), it follows that $(v, w) \in E_V$. This establishes (3.4).

Consider now to following property

$$\mathcal{E}_c(\phi \cup \delta_V) = \mathcal{E}(\mathcal{E}_c(\phi) \cup E_V) \ . \tag{3.5}$$

To see that it holds, note first that by basic properties of equivalence and congruence closures we have that

$$\mathcal{E}_c(\phi \cup \delta_V) = \mathcal{E}_c(\mathcal{E}_c(\phi) \cup \mathcal{E}(\delta_V)) = \mathcal{E}_c(\mathcal{E}_c(\phi) \cup E_V) \ .$$

To see that $\mathcal{E}_c(\mathcal{E}_c(\phi) \cup E_V) = \mathcal{E}(\mathcal{E}_c(\phi) \cup E_V)$, suppose that this is not the case. Then there must exist a pair of function applications $t_1 = f(x_1, x_2, \ldots x_n)$ and $t_2 = f(y_1, y_2, \ldots y_n)$, appearing in ϕ, such that $(t_1, t_2) \notin \mathcal{E}_c(\phi)$ but for each $1 \leq j \leq n$, $(x_j, y_j) \in \mathcal{E}(\mathcal{E}_c(\phi) \cup E_V)$. From this it follows by (3.4) that $(x_j, y_j) \notin \mathcal{N}_c(\phi)$. Therefore, terms t_1 and t_2 are candidate terms of our care function.

Since $\mathcal{E}_c(\phi)$ is closed under congruence, there must be some i such that $(x_i, y_i) \notin \mathcal{E}_c(\phi)$ and $(x_i, y_i) \in \mathcal{E}(\mathcal{E}_c(\phi) \cup E_V)$. But then we must have a chain of equalities connecting x_i to y_i, such that at least one equality comes from E_V. This chain then has to contain at least two variables from V. Let x be the first variable from V, and y the last variable from V, in this equality chain. Since the chains from x_i to x, and y_i to y do not contain any variables from V, all these equalities must come from $\mathcal{E}_c(\phi)$, so it must be that $(x_i, x) \in \mathcal{E}_c(\phi)$ and $(y_i, y) \in \mathcal{E}_c(\phi)$. We can conclude that $(x, y) \notin \mathcal{E}_c(\phi)$ since otherwise we could deduce that $(x_i, y_i) \in \mathcal{E}_c(\phi)$. Additionally, it must be that $(x, y) \notin \mathcal{N}_c(\phi)$, as otherwise we would have $x \neq y \in \delta_V$ and thus $(x, y) \notin E_V$, but we know from (3.3) that $(x, y) \in E_V$.

But now we can see that x and y satisfy all the requirements necessary to ensure that the edge (x, y) must be in the care graph $\mathbf{G}$, contradicting the fact that it should be empty, which establishes (3.5).

Finally, we return to the main proof and show that if $x \neq y \in \phi \cup \delta_V$, then $(x, y) \notin \mathcal{E}_c(\phi \cup \delta_V)$. We consider two cases.

- First, suppose $x \neq y \in \delta_V$ (and thus $x, y \in V$). Clearly, we cannot also have $x = y \in \delta_V$, so $(x, y) \notin E_V$. It follows by (3.3) that $(x, y) \notin \mathcal{E}(\mathcal{E}_c(\phi) \cup E_V)$, and thus, by (3.5), $(x, y) \notin \mathcal{E}_c(\phi \cup \delta_V)$.

- On the other hand, suppose that $x \neq y \in \phi$. This means that $(x, y) \in \mathcal{N}_c(\phi)$. By (3.4), it follows that $(x, y) \notin \mathcal{E}(\mathcal{E}_c(\phi) \cup E_V)$, and thus, by (3.5), $(x, y) \notin \mathcal{E}_c(\phi \cup \delta_V)$. □

3.6 Theory of Arrays

In this section we show how to construct, and prove correct, a care function for the theory of arrays that we defined in Example 3.2 of Section 3.3. Before presenting the equality propagator and care function, it will be helpful to present a simple rule-based decision procedure for $T_{\mathtt{arr}}$. The procedure we present is based on [36], with the main difference that in our procedure, we exclude literals containing write terms from the $T_{\mathtt{euf}}$-satisfiability check as they are not needed and this allows us to have a simpler care function.

3.6.1 A Decision Procedure

For a set ϕ of flat $T_{\mathtt{arr}}$-literals, we define $\alpha(\phi)$ to be the subset of ϕ that does not contain literals of the form $a = \mathsf{write}(b, i, v)$. We also define $\mathcal{E}_a(\Gamma)$ as $\mathcal{E}_c(\alpha(\Gamma))$ and, as usual, the corresponding dis-equality relation $\mathcal{N}_a(\Gamma)$ as the smallest relation satisfying:

$$\text{if } (w, w') \in \mathcal{E}_a(\Gamma) \text{ and } (z, z') \in \mathcal{E}_a(\Gamma) \text{ and } w' \neq z' \in \Gamma \text{ then } (w, z) \in \mathcal{N}_a(\Gamma) \ .$$

As a matter of notational convenience in this section we write $x \approx_a^\Gamma$ for $(x, y) \in \mathcal{E}_a(\Gamma)$ and $x \neq_a^\Gamma y$ for $(x, y) \in \mathcal{N}_a(\Gamma)$.

The following simple lemma is a straightforward consequence of the fact that adding additional information can only increase the set of consequences of a set of formulas.

Lemma 3.1. *Suppose ϕ and Γ are sets of flat $T_{\mathtt{arr}}$-literals with $\phi \subseteq \Gamma$. Then:*

- $s \approx_a^\phi t \implies s \approx_a^\Gamma t$ *, and*
- $s \neq_a^\phi t \implies s \neq_a^\Gamma t$.

We now present the inference rules of the decision procedure for $T_{\mathtt{arr}}$. For a set of literals Γ, we write $\Gamma[l_1, \ldots, l_n]$ to denote that literals $l_1, \ldots, l_n$ appear in Γ. For every pair (a, b) of

variables from $\mathsf{vars}_{\mathsf{array}}(\Gamma)$, we let $k_{a,b}$ be a distinguished fresh variable of sort index. Let $\mathcal{D}_{\mathtt{arr}}$ be the following set of inference rules.

$$\mathsf{RIntro1}\quad \frac{\Gamma[a = \mathsf{write}(b,i,v)]}{\Gamma, v = \mathsf{read}(a,i)} \quad \text{if} \quad v \not\approx_a^{\Gamma} \mathsf{read}(a,i)$$

$$\mathsf{RIntro2}\quad \frac{\Gamma[a = \mathsf{write}(b,i,v), x = \mathsf{read}(c,j)]}{\Gamma, i = j \qquad \Gamma, \mathsf{read}(a,j) = \mathsf{read}(b,j)} \quad \text{if} \quad \begin{cases} a \approx_a^{\Gamma} c \text{ or } b \approx_a^{\Gamma} c, \\ i \not\approx_a^{\Gamma} j, \text{ and} \\ \mathsf{read}(a,j) \not\approx_a^{\Gamma} \mathsf{read}(b,j) \end{cases}$$

$$\mathsf{ArrDiseq}\quad \frac{\Gamma[a \neq b]}{\Gamma, \mathsf{read}(a,k_{a,b}) \neq \mathsf{read}(b,k_{a,b})} \quad \text{if} \quad \text{not } \mathsf{read}(a,k_{a,b}) \neq_a^{\Gamma} \mathsf{read}(b,k_{a,b})$$

Note that although non-flat literals appear in the conclusions of rules RIntro2 and ArrDiseq, we only use this as a shorthand for the flattened version of these literals. For example, $\mathsf{read}(a,j) = \mathsf{read}(b,j)$ is shorthand for $x = \mathsf{read}(a,j) \wedge y = \mathsf{read}(b,j) \wedge x = y$, where x and y are fresh variables (there are other possible flattenings, especially if one or more of the terms appears already in Γ, but any of them will do). We say that a set Γ of literals is $\mathcal{D}_{\mathtt{arr}}$-*saturated* if no rules from $\mathcal{D}_{\mathtt{arr}}$ can be applied.

Theorem 3.17. *The inference rules of $\mathcal{D}_{\mathtt{arr}}$ are sound and terminating.*

Proof. By sound, we mean that for each rule, the set of literals in the premise is $T_{\mathtt{arr}}$-satisfiable iff one of the conclusion sets is $T_{\mathtt{arr}}$-satisfiable. It is not hard to see that the soundness of the RIntro1 rule follows from axiom (RW1) of $T_{\mathtt{arr}}$, the soundness of RIntro2 follows from axiom (RW2), and the soundness of ArrDiseq from axiom (EX).

To see that the rules are terminating, first notice that applying a rule results in a new set Γ which no longer satisfies the side conditions of the rule just applied, so every application of a rule along a derivation branch must involve different "trigger" formulas (the ones in square brackets). Now, no rule introduces array dis-equalities, so it is clear that ArrDiseq can only be applied a finite number of times. Similarly, no rule introduces a new literal containing write, so RIntro1 can only be applied a finite number of times. Now, suppose we have a set Γ in which both the RIntro1 rule and the ArrDiseq rule no longer apply, and consider rule RIntro2 which may introduce new read terms. The rule cannot, however, introduce new array or index variables, so

there are only a finite number of read terms that can be generated. Each application of RIntro2 merges the equivalence classes of either two index terms or two read terms. Since there are a finite number of both, eventually, no more merges will be possible. □

If we apply the decision procedure rules to exhaustion, as shown by the theorem below, we can resort to equality reasoning provided by the congruence reasoning to check the $T_{\mathtt{arr}}$-satisfiability of the original formula. This hints to the possibility of reusing the care function of $T_{\mathtt{euf}}$ in our construction of the care function for $T_{\mathtt{arr}}$.

Additionally, we will show that if a formula is $T_{\mathtt{arr}}$ satisfiable, it is satisfiable in a particular model that allows for unrelated arrays to be interpreted as distinct. To formalize this, we define the relation $\mathcal{P}_a(\phi)$ as the smallest equivalence relation that contains all pairs (a, b) of array variables such that either $a = b \in \phi$ or $a = \mathsf{write}(b, i, v) \in \phi$.

Theorem 3.18. *Let Γ be a $\mathcal{D}_{\mathtt{arr}}$-saturated set of flat $T_{\mathtt{arr}}$-literals. Then Γ is $T_{\mathtt{arr}}$-satisfiable if and only if $\alpha(\Gamma)$ is $T_{\mathtt{euf}}$-satisfiable.*

Proof. Since read and write are function symbols, with function interpretations, all the structures of $T_{\mathtt{arr}}$ are included in the structures of $T_{\mathtt{euf}}$. With this, and the fact that $\alpha(\Gamma) \subseteq \Gamma$, the only-if direction is trivial.

For the other direction, suppose Γ is a $\mathcal{D}_{\mathtt{arr}}$-saturated set of flat $T_{\mathtt{arr}}$-literals. Let $\mathcal{A}$ be a maximally diverse $T_{\mathtt{euf}}$ model of $\alpha(\Gamma)$ (for instance, the $\approx_a^\Gamma$-quotient of the term model) and note that it has the property that for any two terms s and t from Γ of the same sort, $s \approx_a^\Gamma t$ iff $s^{\mathcal{A}} = t^{\mathcal{A}}$. Also note that since the theory $T_{\mathtt{euf}}$ is stably infinite with respect to the sorts of elements and indices, we can assume that in $\mathcal{A}$ the domains of elem and index are infinite.

We will show that Γ is $T_{\mathtt{arr}}$-satisfiable by constructing a $T_{\mathtt{arr}}$ interpretation $\mathcal{B}$ that satisfies Γ. We start by defining the domains of $\mathcal{B}$ as

$$
\begin{aligned}
B_{\mathsf{index}} &= A_{\mathsf{index}} \ , \\
B_{\mathsf{elem}} &= A_{\mathsf{elem}} \ , \\
B_{\mathsf{array}} &= \{ \ f \ \mid \ f : B_{\mathsf{index}} \mapsto B_{\mathsf{elem}} \ \} \ .
\end{aligned}
$$

We further define the interpretations of function symbols as

$$\mathsf{read}^{\mathcal{B}} = \lambda\, a : B_{\mathsf{array}}.\ \lambda\, i : B_{\mathsf{index}}.\ a(i)\ ,$$

$$\mathsf{write}^{\mathcal{B}} = \lambda\, a : B_{\mathsf{array}}.\ \lambda\, i : B_{\mathsf{index}}.\ \lambda\, x : B_{\mathsf{elem}}.\ (\lambda\, j : B_{\mathsf{index}}.\text{if } i = j \text{ then } x \text{ else } a(j))\ .$$

For all index variables i and elem variables x we keep their original interpretations in $\mathcal{A}$, i.e. we take $i^{\mathcal{B}} = i^{\mathcal{A}}$ and $x^{\mathcal{B}} = x^{\mathcal{A}}$.

We interpret each array $a \in \mathsf{vars}_{\mathsf{array}}(\Gamma)$ as the corresponding function from $\mathcal{A}$, but in a restricted manner. Let $a_1, \ldots, a_n$ be some representatives of the equivalence classes in $\mathcal{P}_a(\Gamma)$ and let $e_1, \ldots e_n$ be distinguished distinct elements of $B_{\mathsf{elem}} = A_{\mathsf{elem}}$ (they exist as the domains are infinite). Then

$$a^{\mathcal{B}} = \lambda\, e : B_{\mathsf{index}}.\ \begin{cases} x^{\mathcal{A}} & \text{if } x = \mathsf{read}(b, i) \in \Gamma \text{ and } a \approx_a^{\Gamma} b \text{ and } i^{\mathcal{A}} = e \\ e_k & \text{otherwise, where } (a, a_k) \in \mathcal{P}_a(\Gamma) \end{cases}$$

Intuitively, we interpret the elements according to the corresponding interpretations of the reads, but we make sure that the default elements are picked different for the different classes in $\mathcal{P}_a(\Gamma)$. To see that this interpretation is well-defined, suppose that for some variable a, we have both $x = \mathsf{read}(b, i) \in \Gamma$ and $y = \mathsf{read}(c, j) \in \Gamma$, with $a \approx_a^{\Gamma} b \approx_a^{\Gamma} c$ and $i^{\mathcal{A}} = j^{\mathcal{A}} = e$. Clearly, $b^{\mathcal{A}} = c^{\mathcal{A}}$, but then it must be the case that $\mathsf{read}(b, i)^{\mathcal{A}} = \mathsf{read}(c, j)^{\mathcal{A}}$ and so $x^{\mathcal{A}} = y^{\mathcal{A}}$.

It is easy to see that the definitions of read and write satisfy the axioms of $T_{\mathtt{arr}}$. Now, we proceed to show that $\mathcal{B} \models \Gamma$. First, note that by definition, equalities and dis-equalities between variables of sort index or elem are trivially satisfied. Next, consider an equality of the form $x = \mathsf{read}(a, i)$. Since this equality is in Γ, we know by the definition of $a^{\mathcal{B}}$ that $a^{\mathcal{B}}(i^{\mathcal{B}}) = x^{\mathcal{B}}$, so by the definition of read, such equalities must be satisfied. This shows that for terms t of sort index or elem, $t^{\mathcal{B}} = t^{\mathcal{A}}$ and thus if s is a term of the same sort as t, $s \approx_a^{\Gamma} t$ iff $s^{\mathcal{B}} = t^{\mathcal{B}}$. Similarly, it is not hard to see that since $\alpha(\Gamma)$ is satisfiable, we must have that if $s \not\approx_a^{\Gamma} t$ then $s^{\mathcal{B}} \neq t^{\mathcal{B}}$.

Next, consider equalities and dis-equalities between array-variables. For every dis-equality $a \neq b \in \Gamma$, we know that (because Γ is saturated), $\mathsf{read}(a, k_{a,b}) \not\approx_a^{\Gamma} \mathsf{read}(b, k_{a,b})$, and thus $\mathsf{read}(a, k_{a,b})^{\mathcal{B}} \neq \mathsf{read}(b, k_{a,b})^{\mathcal{B}}$, from which it is clear that $a^{\mathcal{B}} \neq b^{\mathcal{B}}$. To see that equalities $a = b$ are satisfied, note that we have $(a, b) \in \mathcal{P}_a(\Gamma)$ and $a \approx_a^{\Gamma} b$, and thus the definitions of $a^{\mathcal{B}}$ and $b^{\mathcal{B}}$ will yield the same function.

Finally, consider an equality of the form $a = \mathsf{write}(b, i, v)$. Let $f_a = a^{\mathcal{B}}$ and $f_{\mathsf{write}} = \mathsf{write}(b, i, v)^{\mathcal{B}}$. We will show this directly by showing that for all index-elements $\iota \in B_{\mathsf{index}}$ we also have $f_a(\iota) = f_{\mathsf{write}}(\iota)$. First, suppose that $\iota = i^{\mathcal{B}}$. In this case, it is clear that $f_{\mathsf{write}}(\iota) = v^{\mathcal{B}}$ by the definition of $\mathsf{write}^{\mathcal{B}}$. Also, by the RIntro1 rule (and saturation of Γ), we know that $v \approx_a^{\Gamma} \mathsf{read}(a, i)$ and so $\mathsf{read}(a, i)^{\mathcal{B}} = v^{\mathcal{B}}$. Then, by the definition of $a^{\mathcal{B}}$, we must have $f_a(\iota) = v^{\mathcal{B}}$.

Suppose, on the other hand that $\iota \neq i^{\mathcal{B}}$. Note that by the definition of $\mathsf{write}^{\mathcal{B}}$, this implies that $f_{\mathsf{write}}(\iota) = b^{\mathcal{B}}(\iota)$. Based on how we defined the interpretations of arrays a and b, we now consider the following cases:

- Suppose that we have $x = \mathsf{read}(c, j) \in \Gamma$ with $a \approx_a^{\Gamma} c$ and $j^{\mathcal{B}} = \iota$. In this case, the definition of $a^{\mathcal{B}}$ ensures that $f_a(\iota) = \mathsf{read}(c, j)^{\mathcal{B}}$. Looking at rule RIntro2, we can see that because Γ is saturated and $i^{\mathcal{B}} \neq j^{\mathcal{B}}$, we must have $\mathsf{read}(a, j)^{\mathcal{B}} = \mathsf{read}(b, j)^{\mathcal{B}}$. But the first is equal to $\mathsf{read}(c, j)^{\mathcal{B}}$ by saturation and rule RIntro1, and the second is equal to $b^{\mathcal{B}}(\iota)$ by the definition of $\mathsf{read}^{\mathcal{B}}$. Thus, $f_{\mathsf{write}}(\iota) = f_a(\iota)$.

- A similar case is when $x = \mathsf{read}(c, j) \in \Gamma$ with $b \approx_a^{\Gamma} c$ and $j^{\mathcal{B}} = \iota$. Here, we have $f_{\mathsf{write}}(\iota) = \mathsf{read}(c, j)^{\mathcal{B}}$ by definition, and we can again conclude that $\mathsf{read}(a, j)^{\mathcal{B}} = \mathsf{read}(b, j)^{\mathcal{B}}$ by rule RIntro2. But the first is equal to $f_a(\iota)$ by the definition of read and the second is equal to $\mathsf{read}(c, j)^{\mathcal{B}}$ and thus to $f_{\mathsf{write}}(\iota)$.

- Finally, when neither of the previous cases hold, the definitions of $a^{\mathcal{B}}$ and $b^{\mathcal{B}}$ ensure that $f_a(\iota) = a^{\mathcal{B}}(\iota) = e_k = b^{\mathcal{B}}(\iota) = f_{\mathsf{write}}(\iota)$. The default interpretations are both equal to the same e_k since from $a = \mathsf{write}(b, i, v) \in \Gamma$ we know that $(a, b) \in \mathcal{P}_a(\Gamma)$ and thus both arrays have the same representative a_k.

Since $\mathcal{B}$ satisfies the axioms and each of the literals in Γ, this shows that Γ is $T_{\mathtt{arr}}$-satisfiable and concludes the proof. □

In the previous Theorem, we've taken particular care to interpret the arrays from different classes in $\mathcal{P}_a(\Gamma)$ to be different. This flexibility allows us to show the following two results.

Corollary 3.1. *A set of flat $T_{\mathtt{arr}}$-literals ϕ is $T_{\mathtt{arr}}$-satisfiable iff the following formula is also $T_{\mathtt{arr}}$-satisfiable*

$$\phi \cup \{\ a \neq b \ \mid \ (a, b) \notin \mathcal{P}_a(\phi)\ \}\ .$$

Proof. If ϕ is satisfiable, we know from Theorem 3.17 that there is a $\mathcal{D}_{\mathtt{arr}}$-saturated set $\Gamma \supseteq \phi$ that is also satisfiable. Notice that during saturation we do not introduce any equalities among array variables or write terms, hence $\mathcal{P}_a(\phi) = \mathcal{P}_a(\Gamma)$. Moreover, in the proof of Theorem 3.18 we've constructed an interpretation for such Γ where, for two arrays a and b that belong to different equivalence classes in $\mathcal{P}_a(\Gamma)$, the default values of the functions corresponding to a and b are different. Since the read terms from Γ only constrain the interpretations of a and b at finite number of points, and we've chosen the interpretations of index points to be infinite, it must be that the interpretations of a and b differ in at least one point (in fact infinitely many points). Therefore this same interpretation also satisfies the set $\phi \cup \{\ a \neq b \mid (a,b) \notin \mathcal{P}_a(\phi)\ \}$. □

Corollary 3.2. *Let ϕ be a $T_{\mathtt{arr}}$-satisfiable set of flat $T_{\mathtt{arr}}$-literals. Then the relation $\mathcal{P}_a(A)$ is an over-approximation of all* array *equalities that are implied by ϕ, i.e.*

$$\phi \vDash_{T_{\mathtt{arr}}} a = b \implies (a,b) \in \mathcal{P}_a(\phi)\ .$$

3.6.2 Equality Propagator

Let ϕ be a set of flat literals and V a set of variables. Consider the following modified versions of the RIntro2 rule that are enabled only if one of the branches can be ruled out as unsatisfiable:

$$\mathsf{RIntro2a}\quad \frac{\Gamma[a = \mathsf{write}(b,i,v), x = \mathsf{read}(c,j)]}{\Gamma, i = j} \quad \text{if} \begin{cases} a \approx_a^\Gamma c \text{ or } b \approx_a^\Gamma c, \\ i \not\approx_a^\Gamma j, \text{ and} \\ \mathsf{read}(a,j) \neq_a^\Gamma \mathsf{read}(b,j) \end{cases}$$

$$\mathsf{RIntro2b}\quad \frac{\Gamma[a = \mathsf{write}(b,i,v), x = \mathsf{read}(c,j)]}{\Gamma, \mathsf{read}(a,j) = \mathsf{read}(b,j)} \quad \text{if} \begin{cases} a \approx_a^\Gamma c \text{ or } b \approx_a^\Gamma c, \\ i \neq_a^\Gamma j, \text{ and} \\ \mathsf{read}(a,j) \not\approx_a^\Gamma \mathsf{read}(b,j) \end{cases}$$

Let $\mathcal{D}'_{\mathtt{arr}}$ be obtained from $\mathcal{D}_{\mathtt{arr}}$ by replacing RIntro2 with the above rules. Since these rules mimic RIntro2 when they are enabled, but are enabled less often, it is clear that $\mathcal{D}'_{\mathtt{arr}}$ remains sound and terminating. Let Γ' be the result of applying $\mathcal{D}'_{\mathtt{arr}}$ to Γ until no more rules apply (for the sake of determinism, assume that rules are applied in order of appearance when there is a

choice). We say that Γ' is $\mathcal{D}'_{\texttt{arr}}$-saturated. We define the equality propagator as:

$$\begin{aligned}\mathfrak{P}_{\texttt{arr}}[\![V]\!](\Gamma) = \{\ w = z \mid w, z \in V,\ (w, z) \in \mathcal{E}_a(\Gamma')\ \} \\ \cup \{\ w \neq z \mid w, z \in V,\ (w, z) \in \mathcal{N}_a(\Gamma')\ \}\ .\end{aligned}$$

It is easy to see that $\mathfrak{P}_{\texttt{arr}}[\![\cdot]\!]$ satisfies the requirements for a propagator.

3.6.3 Care Function

Let ϕ be a set of flat literals and V a set of variables. First, since a simple propagator cannot compute all equalities between array variables, we will ensure that the relationships between all pairs of array variables in V have been determined. To do so, we define the set E_a^ϕ of pairs of array variables in V that are not yet known equal or dis-equal. Let $V_a = \mathsf{vars}_{\mathsf{array}}(V)$. Then,

$$E_a^\phi = (V_a \times V_a) \setminus (\mathcal{E}_a(\phi) \cup \mathcal{N}_a(\phi))\ .$$

Next, since the inference rules can introduce new read terms, we compute the smallest set R^ϕ that includes all possible such terms, i.e.,

- if $x = \mathsf{read}(a, i) \in \phi$ or $a = \mathsf{write}(b, i, v) \in \phi$, then $\mathsf{read}(a, i) \in \mathsf{R}^\phi$,
- if $a = \mathsf{write}(b, i, v) \in \phi$, $\mathsf{read}(c, j) \in \mathsf{R}^\phi$, $i \not\approx_a^\phi j$, and $a \approx_a^\phi c$ or $b \approx_a^\phi c$, then both $\mathsf{read}(a, j) \in \mathsf{R}^\phi$ and $\mathsf{read}(b, j) \in \mathsf{R}^\phi$,
- if $a \neq b \in \phi$, then both $\mathsf{read}(a, k_{a,b}) \in \mathsf{R}^\phi$ and $\mathsf{read}(b, k_{a,b}) \in \mathsf{R}^\phi$.

Crucial in the introduction of the above read terms is the set of index variables whose equality could affect the application of the RIntro2 rule. We capture these variables by defining the set E_i^ϕ as the set of all pairs (i, j) such that:

- $i \not\approx_a^\phi j$ and not $i \neq_a^\phi j$
- $\exists\, a, b, c, v.\ a = \mathsf{write}(b, i, v) \in \phi, \mathsf{read}(c, j) \in \mathsf{R}^\phi,$ and $a \approx_a^\phi c$ or $b \approx_a^\phi c$.

Finally, we claim that with the variables in E_a^ϕ and E_i^ϕ decided, we can essentially use the same care function as for $T_{\texttt{euf}}$, treating the read function symbol as uninterpreted. We therefore define the third set E_r^ϕ to be the set of all pairs $(i, j) \in V \times V$ of undecided indices ($i \not\approx_a^\phi j$ and not $i \neq_a^\phi j$) such that

- there are terms $\mathsf{read}(a, i'), \mathsf{read}(b, j') \in \mathsf{R}^\phi$ with $\mathsf{read}(a, i') \not\approx_a^\phi \mathsf{read}(b, j')$,
- a and b could be equal and are not known to be disequal, i.e. $(a, b) \in \mathcal{P}_a(\phi)$ and $a \neq_a^\phi b$,
- the variables i and j are relevant, i.e. $i \approx_a^\phi i'$ and $j \approx_a^\phi j'$.

It is important to note that, in the definition of E_r^ϕ, we use the over-approximation $\mathcal{P}_a(a, b)$ of the entailed equalities over arrays.

With the definitions above, we can define the care function as $\mathfrak{C}_{\mathtt{arr}}[\![V]\!](\phi) = \mathbf{G} = \langle V, E\rangle$, where the set of edges is defined as:

$$E = \begin{cases} E_a^\phi & \text{if } E_a^\phi \neq \varnothing, \\ E_i^\phi & \text{if } E_i^\phi \neq \varnothing, \text{ and} \\ E_r^\phi & \text{otherwise.} \end{cases}$$

Note that as defined, E_i^ϕ may include pairs of index variables, one or more of which are not in V. Unfortunately, the care function fails if E_i^ϕ is not a subset of $V \times V$. We can ensure that it is either by expanding the set V until it includes all variables in E_i^ϕ or doing additional case-splitting up front on pairs in E_i^ϕ, adding formulas to ϕ, until $E_i^\phi \subseteq V \times V$.

Example 3.9. Consider the following constraints involving arrays and bit-vectors of size m, where $\times_m$ denotes unsigned bit-vector multiplication:

$$\bigwedge_{k=1}^{n} (\mathsf{read}(a_k, i_k) = \mathsf{read}(a_{k+1}, i_{k+1}) \wedge i_k = x_k \times_m x_{k+1}) \quad . \tag{3.6}$$

Assume that only the index variables are shared, i.e. $V = \{i_1, \ldots, i_{n+1}\}$. In this case, both E_a^ϕ and E_i^ϕ will be empty and the only read terms in R^ϕ will be those appearing in the formula. Since none of these are reading from equivalent arrays, the empty care graph is a fix-point for our care function, and we do not need to guess an arrangement.

Note that in the case when V contains array variables, the care graph requires us to split on all pairs of these variables (i.e. we use the trivial care function over these variables). Fortunately,

in practice it appears that index and element variables are typically shared, and only rarely are array variables shared.

Lemma 3.2. *Let ϕ be a set of flat $T_{\mathtt{arr}}$-literals, and suppose that $\mathfrak{C}_{\mathtt{arr}}[\![V]\!](\phi) = \langle V, \varnothing\rangle$. If Γ is a satisfiable set of literals obtained from ϕ via a sequence of $\mathcal{D}_{\mathtt{arr}}$-inferences, and $\mathsf{read}(a,i)$ appears in Γ, then $\mathsf{read}(a,i) \in \mathsf{R}^{\phi}$.*

Proof. The proof is by induction on inference rule applications. For the base case, suppose $\Gamma = \phi$. The first rule defining R^{ϕ} ensures that $\mathsf{read}(a,i) \in \mathsf{R}^{\phi}$.

For the inductive case, suppose every term $\mathsf{read}(a,i)$ apearing in Γ is in R^{ϕ} and let Γ' be obtained by applying an inference rule to Γ. Suppose the inference rule is RIntro1. This introduces a term of the form $\mathsf{read}(a,i)$. It also requires that we have an equality $a = \mathsf{write}(b,i,v) \in \Gamma$. But no rule introduces such equalities, so it must have been in ϕ originally. Again, the first rule defining R^{ϕ} then ensures that $\mathsf{read}(a,i) \in \mathsf{R}^{\phi}$.

Next, suppose the inference rule is RIntro2. The right branch of this rule may introduce $\mathsf{read}(a,j)$ and $\mathsf{read}(b,j)$. In this case, we know there are equalities $a = \mathsf{write}(b,i,v)$ and $x = \mathsf{read}(c,j)$ in Γ with $i \not\approx_a^{\Gamma} j$, and either $a \approx_a^{\Gamma} c$ or $b \approx_a^{\Gamma} c$. As before, we must have $a = \mathsf{write}(b,i,v) \in \phi$, and by the inductive hypothesis, we know that $\mathsf{read}(c,j) \in \mathsf{R}^{\phi}$. Furthermore, because $E_a^{\phi} = \varnothing$, we know that all relationships between array variables are already determined by ϕ, so either $a \approx_a^{\phi} c$ or $b \approx_a^{\phi} c$; and we know from Lemma 3.1 that $i \not\approx_a^{\phi} j$. We can then see that the second rule defining R^{ϕ} ensures that $\mathsf{read}(a,j)$ and $\mathsf{read}(b,j)$ are in R^{ϕ}.

Finally, suppose that the inference rule is ArrDiseq. This rule may introduce $\mathsf{read}(a,k_{a,b})$ and $\mathsf{read}(b,k_{a,b})$. This can only happen if $a \neq b \in \Gamma$. Since no rules introduce dis-equalities between array variables, this implies that $a \neq b \in \phi$, and so the last rule defining R^{ϕ} ensures that $\mathsf{read}(a,k_{a,b}) \in \mathsf{R}^{\phi}$ and $\mathsf{read}(b,k_{a,b}) \in \mathsf{R}^{\phi}$. □

Theorem 3.19. *Let $T_{\mathtt{arr}}$ be the theory of arrays. $\mathfrak{C}_{\mathtt{arr}}[\![\cdot]\!]$ is a care function for $T_{\mathtt{arr}}$ with respect to the equality propagator $\mathfrak{P}_{\mathtt{arr}}[\![\cdot]\!]$ for all sets ϕ of literals and V of variables such that $E_i^{\phi} \subseteq V \times V$.*

Proof. Assume that we are given a set ϕ of flat $\Sigma_{\mathtt{arr}}$-literals and a set V of variables with $E_i^{\phi} \subseteq V \times V$. Let $\mathfrak{C}_{\mathtt{arr}}[\![V]\!](\phi) = \langle V, \varnothing\rangle$, and assume that ϕ is $T_{\mathtt{arr}}$-satisfiable and δ_V is a variable arrangement such that $\delta_V \supseteq \mathfrak{P}_{\mathtt{arr}}[\![V]\!](\phi)$.

Because ϕ is $T_{\texttt{arr}}$-satisfiable, we know from Theorems 3.17 and 3.18 that we can find a $\mathcal{D}_{\texttt{arr}}$-saturated set $\Gamma \supseteq \phi$ that is $T_{\texttt{arr}}$-satisfiable and $\alpha(\Gamma)$ is $T_{\texttt{euf}}$-satisfiable. We define a set δ_P of additional disequalities based on $\mathcal{P}_a(\Gamma)$ as

$$\delta_P = \{\ a \neq b \mid (a,b) \notin \mathcal{P}_a(\Gamma)\ \}$$

Moreover, from Corollary 3.1 we can conclude that $\Gamma \cup \delta_P$ is $T_{\texttt{arr}}$-satisfiable and therefore $\alpha(\Gamma) \cup \delta_P$ is $T_{\texttt{euf}}$-satisfiable. From here we can resort to correctness of the $T_{\texttt{euf}}$ care function if we can show the following

$$\mathfrak{C}_{\texttt{euf}}[\![V]\!](\alpha(\Gamma) \cup \delta_P) = \langle V, \varnothing \rangle \ , \tag{3.7}$$

$$\mathfrak{P}_{\texttt{euf}}[\![V]\!](\alpha(\Gamma) \cup \delta_P) \subseteq \delta_V \ . \tag{3.8}$$

If we can show the above two properties, we can then proceed as follows. Using Theorem 3.16 we can show that $\alpha(\Gamma) \cup \delta_P \cup \delta_V$ must be $T_{\texttt{euf}}$-satisfiable, and therefore $\alpha(\Gamma) \cup \delta_V$ is also $T_{\texttt{euf}}$-satisfiable. But $\alpha(\Gamma) \cup \delta_V = \alpha(\Gamma \cup \delta_V)$, so $\alpha(\Gamma \cup \delta_V)$ is also $T_{\texttt{euf}}$-satisfiable. Finally, since Γ is $\mathcal{D}_{\texttt{arr}}$-saturated, and δ_V can only add new equalities and disequalities between variables of sort index or elem, it is clear that $\Gamma \cup \delta_V$ must also be $\mathcal{D}_{\texttt{arr}}$-saturated. Therefore, by Theorem 3.18, $\Gamma \cup \delta_V$ is $T_{\texttt{arr}}$-satisfiable, from which we can conclude that $\phi \cup \delta_V$ is $T_{\texttt{arr}}$-satisfiable, which concludes the proof.

We start by showing a seemingly weaker property than (3.7), that is, we claim that

$$\mathfrak{P}_{\texttt{euf}}[\![V]\!](\alpha(\Gamma)) \subseteq \delta_V \ . \tag{3.9}$$

We know from the assumptions of the theorem that $\mathfrak{P}_{\texttt{arr}}[\![V]\!](\phi) \subseteq \delta_V$, so it suffices to show that $\mathfrak{P}_{\texttt{euf}}[\![V]\!](\alpha(\Gamma)) = \mathfrak{P}_{\texttt{arr}}[\![V]\!](\phi)$. Consider the set Γ' taken as the $\mathcal{D}'_{\texttt{arr}}$-saturated set obtained starting from ϕ. By definition of the propagator $\mathfrak{P}_{\texttt{arr}}[\![\cdot]\!]$ it follows directly that $\mathfrak{P}_{\texttt{arr}}[\![V]\!](\phi) = \mathfrak{P}_{\texttt{euf}}[\![V]\!](\alpha(\Gamma'))$. Thus, is enough to show that $\mathfrak{P}_{\texttt{euf}}[\![V]\!](\alpha(\Gamma)) = \mathfrak{P}_{\texttt{euf}}[\![V]\!](\alpha(\Gamma'))$. In fact, we can show that $\Gamma = \Gamma'$. Suppose not. The only way this could happen is if there is some $\mathcal{D}_{\texttt{arr}}$-derivation starting from ϕ in which rule $\mathsf{RIntro2}$ applies but rules $\mathsf{RIntro2a}$ and $\mathsf{RIntro2b}$ do not. Let Γ'' be the first set in the $\mathcal{D}_{\texttt{arr}}$-derivation from ϕ to Γ in which this is the case. In order for the rule to be enabled, we must have $a = \mathsf{write}(b,i,v), x = \mathsf{read}(c,j) \in \Gamma''$. As we have noted before, derivations do not introduce equalities containing applications of write, so we must have

$a = \mathsf{write}(b, i, v) \in \phi$. We also know by Lemma 3.2 that $\mathsf{read}(c, j) \in \mathsf{R}^\phi$. We also have $a \approx_a^{\Gamma''} c$ or $b \approx_a^{\Gamma''} c$, so it follows from the fact that $E_a^\phi = \varnothing$ that $a \approx_a^\phi c$ or $b \approx_a^\phi c$. But now, since $E_i^\phi = \varnothing$, clearly we must have $i \approx_a^\phi j$ or $i \not\approx_a^\phi j$. In the first case, we know that $i \approx_a^{\Gamma''} j$, so rule RIntro2 is not applicable, contradicting our assumption. In the second case, we know that $i \not\approx_a^{\Gamma''} j$ which means that RIntro2b *is* applicable, which also contradicts our assumption.

We have shown that (3.9) holds. Now consider the set $\mathfrak{P}_{\mathtt{euf}}[\![V]\!](\alpha(\Gamma) \cup \delta_P)$. First, since $E_a^\phi = 0$, we know that for all pairs (a, b) of array variables in V we already have either $a = b \in \phi$ or $a \neq b \in \phi$, and therefore the $T_{\mathtt{euf}}$ propagator $\mathfrak{P}_{\mathtt{euf}}[\![\cdot]\!]$ can not possibly learn any new equality information over array variables in V from the additional disequalities in δ_P. Moreover, the $T_{\mathtt{euf}}$ propagator also can not learn any new equality information over index and elem variables once we add the δ_P set of array dis-equalities. Therefore, we conclude that $\mathfrak{P}_{\mathtt{euf}}[\![V]\!](\alpha(\Gamma) \cup \delta_P) = \mathfrak{P}_{\mathtt{euf}}[\![V]\!](\alpha(\Gamma)) \subseteq \delta_V$ which directly shows (3.8).

Now, let $\mathbf{G} = \langle V, E\rangle = \mathfrak{C}_{\mathtt{euf}}[\![V]\!](\alpha(\Gamma) \cup \delta_P)$ be the $T_{\mathtt{euf}}$ care graph based on $\alpha(\Gamma) \cup \delta_P$. As (3.7) states, we claim that $E = \varnothing$. First note that because $E_a^\phi = \varnothing$, we know that for array variables $a, b \in \mathsf{vars}_{\mathsf{array}}(V)$, either $a \approx_a^\phi b$ or $a \not\approx_a^\phi b$, so it is impossible to have $(a, b) \in E$. Next, notice that since variables of sort elem cannot appear as arguments to functions in $\alpha(\Gamma)$, there are no elem pairs $(x, y) \in E$. Finally, suppose we have a pair of index variables (i, j) such that $(i, j) \in E$. By the definition of $\mathfrak{C}_{\mathtt{euf}}[\![\cdot]\!]$, we know that there exist a, b, i', j' such that

1. $\mathsf{read}(a, i')$ and $\mathsf{read}(b, j')$ appear in $\alpha(\Gamma) \cup \delta_P$,
2. $(\mathsf{read}(a, i'), \mathsf{read}(b, j')) \notin \mathcal{E}_c(\alpha(\Gamma) \cup \delta_P)$,
3. $(a, b) \notin \mathcal{N}_c(\alpha(\Gamma) \cup \delta_P)$, and
4. $(i, i') \in \mathcal{E}_c(\alpha(\Gamma) \cup \delta_P)$ and $(j, j') \in \mathcal{E}_c(\alpha(\Gamma) \cup \delta_P)$.

Using Lemma 3.2 we can conclude from property 1 that, since the terms must appear in $\alpha(\Gamma)$, and therefore in Γ, we must have $\underline{\mathsf{read}(a, i') \in \mathsf{R}^\phi \text{ and } \mathsf{read}(b, j') \in \mathsf{R}^\phi}$. Also, since $\mathcal{E}_c(\alpha(\Gamma)) \subseteq \mathcal{E}_c(\alpha(\Gamma) \cup \delta_P)$, from property 2, we know that $\mathsf{read}(a, i') \not\approx_a^\Gamma \mathsf{read}(b, j')$. Then, since $\phi \subseteq \Gamma$ we can use Lemma 3.1 to conclude that $\underline{\mathsf{read}(a, i') \not\approx_a^\phi \mathsf{read}(b, j')}$.

We next consider the implications of property 4. Notice that the only equalities between index variables that could have been introduced during the derivation from ϕ to Γ are those introduced

by rule RIntro2. But if this rule is enabled and could introduce $i = j$, then $(i, j) \in E_i^\phi$. But we know that $E_i^\phi = \varnothing$. It follows that no equalities between index variables are introduced in the derivation. So, if $i \approx_a^\Gamma i'$, then $i \approx_a^\phi i'$. Similarly, if $j \approx_a^\Gamma j'$, then $j \approx_a^\phi j'$. But, from property 4 we know that $(i, i') \in \mathcal{E}_c(\alpha(\Gamma) \cup \delta_P)$. Again, since the disequalities in δ_P do not influence equalities over index terms in the congruence relation, we can conclude that $(i, i') \in \mathcal{E}_c(\alpha(\Gamma))$ and therefore $i \approx_a^\Gamma i'$, and similarity $j \approx_a^\Gamma j'$. We've therefore established that $\underline{i \approx_a^\phi i' \text{ and } j \approx^\phi j'}$.

Finally, lets examine the following two cases on the relation between the arrays a and b:

- If $a \not\approx_a^\phi b$, which is equivalent to $(a, b) \in \mathcal{N}_c(\alpha(\phi))$, by monotonicity of the disequality relation we have that $(a, b) \in \mathcal{N}_c(\alpha(\phi)) \subseteq \mathcal{N}_c(\alpha(\Gamma)) \subseteq \mathcal{N}_c(\alpha(\Gamma) \cup \delta_P)$.

- If we have that $(a, b) \notin \mathcal{P}_a(\phi)$ then, since saturation does not add any equalities among array variables or write terms, we have that $\mathcal{P}_a(\Gamma) = \mathcal{P}_a(\phi)$. By definition of δ_P we therefore have that $(a, b) \in \mathcal{N}_c(\delta_P) \subseteq \mathcal{N}_c(\alpha(\Gamma) \cup \delta_P)$

Since property 4 ensures the opposite of the conclusions of the two cases above, i.e. $(a, b) \notin \mathcal{N}_c(\alpha(\Gamma) \cup \delta_P)$ we can now conclude that it is the case that $\underline{(a, b) \in \mathcal{P}_a(\phi) \text{ and not } a \not\approx_a^\phi b}$.

From the underlined properties above we can now see that all conditions are satisfied to ensure that $(i, j) \in E_r^\phi$. But, this is impossible by assumption of the $T_{\texttt{arr}}$ care graph being empty. Hence we conclude that the $T_{\texttt{euf}}$ care graph is also empty, proving (3.8). □

3.7 Experimental Evaluation

We implemented the new method in the cvc3 solver [7], and in the discussion below, we denote the new implementation as cvc3+c. We focused our attention on the combination of the theory of arrays and the theory of fixed-size bit-vectors (QF_AUFBV). This seemed like a good place to start because there are many benchmarks which generate a significant number of shared variables, and additional splits on shared bit-vector variables can be quite expensive. This allowed us to truly examine the merits of the new combination method. In order to evaluate our method against the current state-of-the-art, we compared to boolector [18], yices [41], cvc3, and mathsat [19], the top solvers in the QF_AUFBV category from the 2009 SMT-COMP competition (in

Table 3.1: Experimental results of the evaluation. For each solver, the first column reports the total time (in seconds) used by that solver on the problem instances it solved. The second column reports the number of instances solved. The best results for each benchmark are in bold.

	boolector		yices		mathsat		z3		cvc3		cvc3+c	
benchmark set	time (s)	solved	time (s)	solved	time (s)	solved	time (s)	solved	time (s)	solved	time (s)	solved
`crafted` (40)	2100.13	40	6253.32	34	468.73	30	112.88	40	388.29	9	**14.22**	**40**
`matrix` (11)	1208.16	10	683.84	6	474.89	4	927.12	11	831.29	11	**45.08**	**11**
`unconstr` (10)	**3.00**	**10**		0	706.02	3	54.60	2	185.00	5	340.27	8
`copy` (19)	11.76	19	**1.39**	**19**	1103.13	19	4.79	19	432.72	17	44.75	19
`sort` (6)	691.06	6	557.23	4	82.21	4	248.94	3	**44.89**	**6**	**44.87**	**6**
`delete` (29)	**3407.68**	**18**	1170.93	10	2626.20	14	1504.46	10	1766.91	17	1302.32	17
`member` (24)	2807.78	24	185.54	24	217.35	24	**112.23**	**24**	355.41	24	320.80	24
	10229.57	**127**	8852.25	97	5678.53	98	2965.02	109	4004.51	89	2112.31	125

order).[7] Additionally, we included the z3 solver [35] so as to compare to the model-based theory combination method [34]. All tests were conducted on a dedicated Intel Pentium E2220 2.4 GHz processor with 4GB of memory. Individual runs were limited to 15 minutes.

We crafted a set of new benchmarks based on Example 3.9 from Section 3.6, taking $n = 10, \ldots, 100$, with increments of 10, and $m = 32, \ldots, 128$, with increments of 32. We also included a selection of problems from the QF_AUFBV division of the SMT-LIB library.[8] Since most of the benchmarks in the library come from model-checking of software and use a flat memory model, they mostly operate over a single array representing the heap. Our method is essentially equivalent to the standard Nelson-Oppen approach for such benchmarks, so we selected only the benchmarks that involved constraints over at least two arrays. We anticipate that such problems will become increasingly important as static-analysis tools become more precise and are able to infer separation of the heap (in the style of Burstall, e.g. [78]). All the benchmarks and the cvc3 binaries used in the experiments are available from the authors' website.[9]

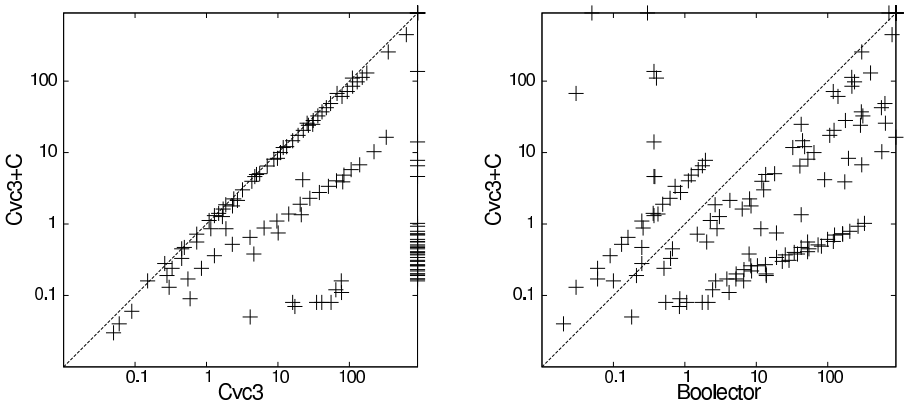
[7] `http://www.smtcomp.org/2009/`

[8] `http://combination.cs.uiowa.edu/smtlib/`

[9] `http://cs.nyu.edu/~dejan/sharing-is-caring/`

The combined results of our experiments are presented in Table 3.1, with columns reporting the total time (in seconds) that a solver used on the problem instances it solved (not including time spent on problem instances it was unable to solve), and the number of solved instances. Compared to cvc3, the new implementation cvc3+c performs uniformly better. On the first four classes of problems, cvc3+c greatly outperforms cvc3. On the last three classes of problems, the difference is less significant. After examining the benchmarks, we concluded that the multitude of arrays in these examples is artificial – the many array variables are just used for temporary storage of sequential updates on the same starting array – so there is not a great capacity for improvement using the care function that we described. A scatter-plot comparison of cvc3 vs cvc3+c is shown in Figure 3.1(a). Because the only difference between the two implementations is the inclusion of the method described in this paper, this graph best illustrates the performance impact this optimization can have.

When compared to the other solvers, we find that whereas cvc3 is not particularly competitive, cvc3+c is very competitive and in fact, for several sets of benchmarks, performs better than all of the others. This again emphasizes the strength of our results and suggests that combination methods can be of great importance for performance and scalability of modern solvers. Overall, on this set of benchmarks, boolector solves the most (solving 2 more than cvc3+c). However, cvc3+c is significantly faster on the benchmarks it solves. Figure 3.1(b) shows a scatter-plot comparison of cvc3+c against boolector.



(a) (b)

Figure 3.1: Comparison of cvc3, cvc3+c and boolector. Both axes use a logarithmic scale and each point represents the time needed to solve an individual problem.

Conclusion

The contributions of this thesis are threefold: we we proposed a new approach for solving linear integer problems, a new procedure for solving non-linear real problems and a new method for combination of decision procedure.

The proposed integer solver has all key ingredients that made CDCL-based SAT solver successful. Our solver justifies propagation steps using tightly-propagating inequalities that guarantee that any conflict detected by the search procedure can be resolved using only inequalities. We presented an approach to integrate Cooper's quantifier elimination algorithm in a model guided search procedure. Our first prototype is already producing encouraging results. We see many possible improvements and extensions to the integer procedure. A solver for Mixed Integer-Real problems is the first natural extension. One basic idea would be to make the real variables bigger than the integer variables in the variable order $\prec$, and use Fourier-Moztkin resolution (instead of Cooper's procedure) to explain conflicts on rational variables. Integrating our solver with a Simplex-based procedure is another promising possibility. The idea is to use Simplex to check whether the current state or the search is feasible in the rational numbers or not.

The new decision procedure for non-linear problems also performs a backtracking search for a model, where the backtracking is powered by a novel conflict resolution procedure. Our first prototype was consistently one of the best solvers for each problem set we tried, and, overall manages to solve the most problems, in much faster time. We expect even better results after several missing optimizations in the core algorithms are implemented. For example, our implementation does yet support full factorization of multivariate polynomials, or algebraic number computations using extension fields. We see many possible improvements and extensions to our procedure. We plan to design and experiment with different explain procedures. One possible idea is to try explain procedures that are more efficient, but do not guarantee termination. Heuristics for reordering variables and selecting a value from the feasible set should also be tried.

The new combination framework can be seen as a reformulation of the classic Nelson-Oppen method for combining theories. The most notable novel feature of the new method is the ability to leverage the structure of the individual problems in order to reduce the complexity of finding

a common arrangement over the interface variables. We do this by defining theory-specific care functions that determine the variable pairs that are relevant in a specific problem. We proved the method correct, and presented care functions for the theories of uninterpreted functions and arrays. The new method is asymmetric as only one of the theories takes charge of creating the arrangement graph over the interface variables. Since many theories we combine in practice are parametrized by other theories, the non-symmetric approach has an intuitive appeal. We draw intuition for the care functions and correctness proofs directly from the decision procedures for specific theories, leaving room for new care functions backed by better decision algorithms. Another benefit of the presented method is that it is orthogonal to the previous research on combinations of theories. For example, it would be easy to combine our method with a model-based combination approach–instead of propagating all equalities between shared variables implied by the model, one could restrict propagation to only the equalities that correspond to edges in the care graph, gaining advantages from both methods.

Appendices

A
nlsat Implementation Details

Acknowledging the importance that the details of a particular implementation play, in this appendix we describe which particular algorithms we used in our implementation, provide additional references, discuss alternatives, and analyze the impact of different optimizations we tried. Our procedure is based on several algorithms for manipulating polynomials and real algebraic numbers. Although most of the operations in these two modules have polynomial time complexity, they are the main bottleneck of our procedure. In our set of benchmarks, we identified two clear bottlenecks: the computation of *principal subresultant coefficients* (psc); and checking the sign of a multivariate polynomial in an irrational coordinate. In all benchmarks our prototype failed to solve, the computation was "stuck" in one of these two procedures.

A.1 Polynomials

We represent multivariate polynomials using a sparse representation of the sum-of-monomials normal form. Each monomial is a sorted vector of pairs of a variable and a degree. For example, the monomial $(x_1^3 x_3^2 x_4)$ is represented as the vector $\langle (1,3),(3,2),(4,1)\rangle$. We store monomials in a hash-table in order to have a unique reference for each monomial. A multivariate polynomial consists of two vectors, one containing the monomials and the other containing the coefficients with these monomials. The coefficients are arbitrary precision integers. The first monomial m in every non constant polynomial p always contains the maximal variable x in p, and is always the monomial of the highest degree in x, i.e $\mathsf{deg}(p,x) = \mathsf{deg}(m,x)$. Thus, the maximal variable of a polynomial and its degree can be quickly retrieved. Moreover, each polynomial has a unique integer identifier and a flag for marking whether the monomials are sorted using lexicographical order or not. Univariate polynomials are implemented separately and are represented as a dense vector of coefficients. For example, the polynomial $2x^5 + 3x + 1$ is represented using the vector $\langle 1,3,0,0,0,2\rangle$.

The polynomial arithmetic operations are implemented using the straightforward algorithms. Faster polynomial multiplication algorithms based on Fast-Fourier Transforms only outperform

the naïve algorithms for polynomials that are well beyond the current capabilities of our decision procedure. We use the standard polynomial pseudo-division algorithm ([25, 59]). In many algorithms (GCD, resultant, psc), exact multivariate polynomial division is used. We say the division of a polynomial p by a polynomial q is exact when there is a polynomial h such that $p = qh$. We use the exact division algorithm described at [9] (Algorithm 8.6) . We implemented three different multivariate polynomial GCD algorithms: subresultant GCD (Chapter 3 [25]), Brown's modular GCD and Zippel's sparse modular probabilistic GCD (Chapter 7 [48]). Although the resultant of two polynomials is formally defined as the determinant of the Sylvester-Habicht Matrix, we used the algorithm based on polynomial pseudo-division, GCD and exact division described at [25] (Algorithm 3.3.7). We also implemented the principal subresultant coefficient algorithm in a similar fashion. The resultant of two polynomials can also be computed using modular techniques similar to the ones used to compute the GCD of two polynomials. However, we did not implement the modular resultant procedure yet. To the best of our knowledge, QEPCAD uses this modular algorithm to control the coefficient growth in the resultant computation.

Regarding polynomial factorization, we perform square-free factorization of a polynomial f using the GCD and its derivative with respect to some variable in f. The polynomial is then put into the form $\prod f_k^k$, where each f_k is the product of all factors of degree k. We extract content and primitive parts of a polynomial using the GCD and exact division. We use the standard approach for univariate factorization, where we first factor a square free polynomial in a finite field $GF(p)$ for some small prime p s.t. the factorization is also square free. Then, the factorization is lifted using Hensel's lemma, and finally we search for factors in the resulting set of polynomials. Further details can be found in [25, 59]. In the current prototype, we do not have support for full multivariate factorization.

We implemented two algorithms for root isolation of univariate polynomials with integer coefficients. One is based on Sturm sequences, and another on the Descartes' rule of signs [21]. In both cases, the computations are performed using *binary rational numbers* [22], also known as *dyadic rationals*. The ring $\mathbb{Z}[1/2]$ of binary rational numbers is the smallest subring of $\mathbb{Q}$ that contains $\mathbb{Z}$ and $1/2$. Binary rationals are rational numbers of the form $a/2^k$. $\mathbb{Z}[1/2]$ is not a field, but it is closed under division by 2. We represent binary rationals using an arbitrary precision integer for a, and a machine unsigned integer for k. The procedures for computing with binary

rational numbers are more efficient than the equivalent ones for rational numbers.

A.2 Real algebraic numbers

In our implementation, a real algebraic number is either a rational number or a square free polynomial f in $\mathbb{Z}[x]$ and an isolating interval of binary rational numbers. Moreover, zero is not a root of f, and the isolating interval does not contain zero. Several algorithms for manipulating algebraic numbers are greatly simplified when square free polynomials are used. Recall that a square free polynomial for f can be computed using $\mathsf{exactdiv}(f, \mathsf{gcd}(f, f'))$, where f' is the first derivative of f. The arithmetical operations $+$, $-$, $\times$, / on algebraic numbers are implemented using resultants [25, 68]. To evaluate the sign of a polynomial $p(x_1, \ldots, x_k)$ at $(\alpha_1, \ldots, \alpha_k)$, we first use interval arithmetic. If the result interval does not contain zero, we are done. Otherwise, we replace all rational α_i's, and try to use interval arithmetic again. We also refine the intervals of each irrational algebraic number until the result interval does not contain zero or the α_i's intervals have size less that $1/2^{32}$. If the result interval still contains zero, let us assume without loss of generality that none of the α_i's are rationals. Then, we compute

$$R_1 = \mathsf{resultant}(y - p(x_1, \ldots, x_n), q_1, x_1)$$

$$\ldots$$

$$R_k = \mathsf{resultant}(R_{k-1}, q_k, x_k)$$

where $\mathsf{resultant}(p, q, x)$ is the resultant of polynomials p and q with respect to variable x, and q_i is the defining polynomial for α_i. R_k is a polynomial in y and, by resultant theory, the number $p(\alpha_1, \ldots, \alpha_k)$ is a root of R_k. Now, we compute a lower bound for the nonzero roots of R_k. This can be accomplished using the same algorithm used to compute a upper bound for the roots of a polynomial. We use the polynomial root upper bound algorithm described in [22]. Using this bound we can keep refining the α_i's intervals until the result interval for $p(\alpha_1, \ldots, \alpha_k)$ does not contain zero, or it is smaller than the lower bound for nonzero roots. In the second case, we have shown that $p(\alpha_1, \ldots, \alpha_k)$ is zero.

For isolating the roots of $p(\alpha_1, \ldots, \alpha_k, y)$, we use a similar approach. We compute

$$R_1 = \mathsf{resultant}(p(x_1, \ldots, x_k, y), q_1, x_1)$$

$$\ldots$$

$$R_k = \mathsf{resultant}(R_{k-1}, q_k, x_k)$$

However, R_k vanishes if $p(x_1, \ldots, x_n, y)$ vanishes for some roots of $q_1, \ldots, q_k$. For example, let $p(x_1, x_2, y) = x_1 y + x_2 y$, and $\alpha_1 = \alpha_2 = (x^2 - 2, (0, 2))$. That is, α_1 and α_2 are the $\sqrt{2}$. However, p vanishes for $p(\sqrt{2}, -\sqrt{2}, y)$. Thus, R_2 is the zero polynomial. To cope with this issue, we use a technique described in [87]. The basic idea is to use algebraic number arithmetic to evaluate the coefficients of p until we find one that does not vanish, or we prove that $p(\alpha_1, \ldots, \alpha_n, y)$ is the zero polynomial.

A.3 Analysis

In this section, we analyze the impact of different algorithms and optimizations we tried. For that, we used an extended set of benchmarks containing 8928 problems. It was not computationally feasible to execute all other systems on this extended set. We remark that all benchmarks that our procedure could not solve or took more than one millisecond to solve are included in the results described in Section 2.4.

Benchmarks. The first observation is that most benchmarks can be solved with very few conflict resolution steps. Only 23 problems required more than 1000 conflict resolutions to be solved. The number of psc chain computations is also very small. Only 17 problems required more than 1000 psc computations. In our prototype, if possible we select a rational number in the rule Lift-Stage. Therefore, many benchmarks can be solved without using any irrational algebraic number computation. Only 1826 benchmarks required irrational algebraic number computations to be solved.

Sparse modular GCD. The use of the sparse modular GCD algorithm instead of the subresultant GCD greatly improved the performance of our procedure. For 43 Meti-Tarski and Zankl benchmarks, we observed a two order of magnitude speedup.

Factorization. A standard technique used in CAD consists in factoring the polynomials obtained using the projection operator. If we disable factoring, 30 benchmarks from Meti-Tarski, Zankl and Hong families cannot be solved anymore, and another 18 benchmarks suffer from a two orders of magnitude slowdown. This suggests we may obtain even better performance results after we implement full multivariate polynomial factorization in our procedure.

Minimal polynomials. The minimal polynomial f of an algebraic number α is the unique irreducible polynomial of smallest degree with integer coefficients such that $f(\alpha) = 0$. Minimal polynomials are obtained using univariate polynomial factorization. Note that every minimal polynomial is square-free. By default, we use minimal polynomials for representing algebraic numbers. If we just use arbitrary square-free polynomials (that are not necessarily minimal) for encoding algebraic numbers, our procedure fails to solve 5 Meti-Tarski benchmarks, and suffers a two orders of magnitude slowdown on 12 other Meti-Tarski benchmarks.

Root isolation. By default, our procedure uses the Descartes' rule of signs procedure for isolating the roots of univariate polynomials. If we switch to a procedure based on Sturm sequences, the performance impact is negligible. Only one Meti-Tarski benchmark suffers from one order of magnitude slowdown.

Full dimensional. We say a problem is *full dimensional* if it contains only strict inequalities. A satisfiable full dimensional problem always has rational solutions. A standard optimization used in CAD-based procedures consists in ignoring sections when processing existential problems. This optimization is justified by the fact that in a full dimensional problem adding a constraint of the form $f \neq 0$, for some nonzero polynomial f, does not change the satisfiability of the problem. To the best of our knowledge, both QEPCAD and Mathematica use this optimization. We implement this approach in our procedure by simply using polynomial constraints of the form $y_k \leq_r \mathsf{root}(f, i, y_k)$ and $y_k \geq_r \mathsf{root}(g, j, y_k)$ instead of $y_k <_r \mathsf{root}(f, i, y_k)$ and $y_k <_r \mathsf{root}(g, j, y_k)$ when a problem is in the full dimensional fragment. With this optimization our prototype solved an extra 12 problems.

Variable reordering. Variable order has a dramatic impact on CAD-based procedures. Mathematica uses heuristics for ordering variables, but we could not find any reference describing the actual heuristics used. We used a simple variable reordering heuristic. First, we compute the maximal degree maxdeg of each variable in the initial set of constraints. Then, before starting our procedure, we sort the variables using the total order

$$x_i \prec x_j \quad \Leftrightarrow \quad \mathsf{maxdeg}(x_i) > \mathsf{maxdeg}(x_j) \vee (\mathsf{maxdeg}(x_i) = \mathsf{maxdeg}(x_j) \wedge i < j) \ .$$

With this simple heuristic, our prototype can solve 54 (35 from the Meti-Tarski, and 15 from the Zankl set) problems that it could not solve. However, the heuristic also prevents our procedure from solving 3 (2 from Meti-Tarski, and 1 from the Zankl set) that could be solved without using it. These results suggest that further work should be invested in developing variable reordering techniques. Dynamically variable reordering during the search is also a promising possibility. However, to guarantee termination it should be eventually disabled.

B
More on Theory Combination

In order to preserve the flow of the main text we've omitted some more technical results from the chapter on combination of theories. For completeness we present these additional results here.

B.1 Being Flat is General Enough

When proving that a Σ-theory is smooth or finitely witnessable with respect to a set of sorts S, we can restrict ourselves to conjunctions of flat Σ-literals. The following two lemmas show that this can indeed be done without loss of generality. The proofs are simple and already presented in [79], but we reiterate them here since they are affected by the change in the definition of finite witnessability.

Lemma B.1. *Let Σ be a signature, let $S \subseteq \Sigma^{\mathbb{S}}$ be a set of sorts, and let T be a Σ-theory. Assume that:*

- *for every T-satisfiable conjunction of flat Σ-literals ψ,*
- *for every T-interpretation $\mathcal{A}$ satisfying ψ,*
- *for all choices of cardinal numbers κ_σ, such that $\kappa_\sigma \geq |A_\sigma|$ for all $\sigma \in S$,*

there exists a T-interpretation $\mathcal{B}$ satisfying ψ such that $|B_\sigma| = \kappa_\sigma$, for all $\sigma \in S$. Then T is smooth with respect to S.

Proof. Assume that a quantifier-free Σ-formula ϕ is satisfiable in a T-interpretation $\mathcal{A}$ and we are given cardinal numbers κ_σ, such that $\kappa_\sigma \geq |A_\sigma|$ for all $\sigma \in S$. We can transform ϕ into its disjunctive normal form $DNF(\phi) = \psi_1 \vee \ldots \vee \psi_m$. Since ϕ and $DNF(\phi)$ are equivalent, T-interpretation $\mathcal{A}$ will satisfy one of the disjuncts ψ_k, for some $1 \leq k \leq m$. We can transform ψ_k into a conjunction of flat literals ψ by introducing fresh variables $\overrightarrow{v}$, such that ψ_k is logically equivalent to $\exists \overrightarrow{v}.\psi$. It follows that there exists a T-interpretation $\mathcal{A}'$ equivalent to $\mathcal{A}$ except in its interpretation of $\overrightarrow{v}$ such that $\mathcal{A}'$ satisfies ψ.

From here, we use the assumptions to obtain a new T-interpretation $\mathcal{B}$ satisfying ψ such that $|B_\sigma| = \kappa_\sigma$, for all $\sigma \in S$. $\mathcal{B}$ will also satisfy $\exists \overrightarrow{v}.\psi$ and, by equivalence, also ψ_k, $DNF(\phi)$ and ϕ. This shows that T is smooth with to respect to S. □

Lemma B.2. *Let Σ be a signature, let $S \subseteq \Sigma^{\mathbb{S}}$ be a set of sorts, and let T be a Σ-theory. Assume there exists a computable function, $witness_F$, which, for every conjunction of flat Σ-literals ϕ, returns a quantifier-free Σ-formula $\psi = witness_F(\phi)$ such that*

- *ϕ and $(\exists \overrightarrow{w})\psi$ are T-equivalent, where $\overrightarrow{w} = \mathsf{vars}(\psi) \setminus \mathsf{vars}(\phi)$ are fresh variables;*
- *if $\psi \wedge \delta_V$ is T-satisfiable, for an arrangement δ_V, where V is a set of variables of sorts in S, then there exists a T-interpretation $\mathcal{A}$ satisfying $\psi \wedge \delta_V$ such that $A_\sigma = [\mathsf{vars}_\sigma(\psi \wedge \delta_V)]^{\mathcal{A}}$, for all $\sigma \in S$.*

Then T is finitely witnessable with respect to S.

Proof. We want to define a witness function *witness* on all quantifier-free Σ-formulas by using the function $witness_F$ as a black box. Let ϕ be a quantifier-free Σ-formula, we compute *witness* using the following steps

1. convert ϕ into a T-equivalent disjunctive normal form $DNF(\phi) = \psi_1 \vee \ldots \psi_m$;
2. transform each disjunct ψ_i into a conjunction of flat Σ-literals ψ'_i by introducing fresh variables;
3. let $witness(\phi) = witness_F(\psi'_1) \vee \ldots \vee witness_F(\psi'_m)$.

If $\overrightarrow{v}_i$ are the fresh variables introduced by flattening ψ_i, we know that ψ_i and $\exists \overrightarrow{v}_i.\psi'_i$ are logically equivalent. Since ψ'_i is T-equivalent to $\exists \overrightarrow{w}_i.witness_F(\psi'_i)$, where $\overrightarrow{w}_i$ are the fresh variables introduced by applying the witness function, we can conclude that $\exists \overrightarrow{v}_i \exists \overrightarrow{w}_i.witness_F(\psi'_i)$ is T-equivalent to $\exists \overrightarrow{v}_i.\psi'_i$, and hence also T-equivalent to ψ_i. Since we can move existential quantifiers over disjunctions (maintaining logical equivalence), we can also conclude that ϕ is T-equivalent to $\exists \overrightarrow{v}_1 \exists \overrightarrow{w}_1 \ldots \exists \overrightarrow{v}_m \exists \overrightarrow{w}_m.witness(\phi)$. This proves the first requirement of the witness function.

For the second requirement, let $\psi = witness(\phi)$ and assume that $\psi \wedge \delta_V$ is T-satisfiable in a T-interpretation $\mathcal{A}$, for an arrangement δ_V, where V is a set of variables of sorts in S. This

implies that one of the disjuncts, say $witness_F(\psi'_k)$, together with δ_V, is satisfied in $\mathcal{A}$, for some $1 \leq k \leq m$. Of course, it is likely the case that $\mathsf{vars}_S(witness_F(\psi'_k) \wedge \delta_V)$ does not include all the variables present in $\mathsf{vars}_S(\psi \wedge \delta_V)$, but we can add the missing variables to our arrangement δ_V [1], while keeping compatibility with $\mathcal{A}$, thus obtaining a stronger arrangement δ'.

Using the assumptions we can therefore get a T-interpretation $\mathcal{B}$ that satisfies $witness_F(\psi'_k) \wedge \delta'$ such that

$$B_\sigma = [\mathsf{vars}_\sigma(witness_F(\psi'_k) \wedge \delta')]^{\mathcal{B}} = [\mathsf{vars}_\sigma(witness(\phi) \wedge \delta_V)]^{\mathcal{B}} \quad ,$$

for all $\sigma \in S$. Since δ' includes δ_V, $\mathcal{B}$ will also satisfy $witness_F(\psi'_k) \wedge \delta_V$, and hence also $witness(\phi) \wedge \delta_V$. This proves that T is indeed finitely witnessable. □

B.2 Preservation of Theory Properties under Combination

We show that stable-infiniteness is preserved when combining theories.

Proposition B.1. *Let Σ_1 and Σ_2 be signatures. If*

- *T_1 is a Σ_1-theory stably-infinite with respect to $S_1 \subseteq \Sigma_1^{\mathbb{S}}$,*
- *T_2 is a Σ_2-theory stably-infinite with respect to $S_2 \subseteq \Sigma_2^{\mathbb{S}}$,*
- *$\Sigma_1^{\mathbb{S}} \cap \Sigma_2^{\mathbb{S}} = S_1 \cap S_2$,*

then $T_1 \oplus T_2$ is a $(\Sigma_1 \cup \Sigma_2)$-theory and is stably-infinite with respect to $S_1 \cup S_2$.

Proof. Let $\Sigma = \Sigma_1 \cup \Sigma_2$, $T = T_1 \oplus T_2$, $S = S_1 \cap S_2$. Assume ϕ is a Σ-formula satisfied in a T-interpretation $\mathcal{A}$. As in the Nelson-Oppen algorithm, we can separate ϕ into the Σ_1-part ϕ_1 and the Σ_2-part ϕ_2. We can assume wlog that $\mathcal{A}$ is an interpretation over the variables $\mathsf{vars}(\phi_1) \cup \mathsf{vars}(\phi_2)$ and that $\mathcal{A} \models \phi_1 \wedge \phi_2$. Let $V = \mathsf{vars}_S(\phi_1) \cap \mathsf{vars}_S(\phi_2)$. Let δ_V be the arrangement over these variables that agrees with $\mathcal{A}$. We have that $\mathcal{A}^{\Sigma_1, \mathsf{vars}(\phi_1)} \models \phi_1 \cup \delta_V$ and $\mathcal{A}^{\Sigma_2, \mathsf{vars}(\phi_2)} \models \phi_2 \cup \delta_V$.

[1] This is the reason why the definition of finite witnessability includes the arrangement over an *arbitrary* set of variables V instead of just an arrangement over $\mathsf{vars}_S(\psi_2)$.

Since T_1 is stably-infinite with respect to S_1, we can conclude that there is a T_1-interpretation $\mathcal{B}$ such that the cardinalities $|B_\sigma|$ are infinite for $\sigma \in S_1$ and $\mathcal{B} \models \phi_1 \cup \delta_V$. Similarly, there is a T_2-interpretation $\mathcal{C}$ with infinite cardinalities $|C_\sigma|$, $\sigma \in S_2$ and $\mathcal{C} \models \phi_2 \cup \delta_V$. By the downward Löwenheim-Skolem theorem in the many-sorted setting,[2] we can assume that the cardinalities $|B_\sigma|$ and $|C_\sigma|$ are the same for $\sigma \in S$.

It is easy to see that $\mathcal{B}$ and $\mathcal{C}$ satisfy all the conditions of Theorem 3.5, so we can conclude that there is a T-interpretation $\mathcal{D}$ that satisfies $\phi_1 \wedge \phi_2$ (and hence ϕ) such that the cardinalities $|D_\sigma|$ are infinite for $\sigma \in S_1 \cup S_2$. □

Proposition 3.1 shows how to combine two theories, one of which is polite. However, the theorem tells us nothing about the politeness of the resulting (combined) theory. In particular, if we want to combine more than two theories by iterating the combination method, we cannot assume that the result of applying Proposition 3.1 is a theory that is polite with respect to any (non-empty) set of sorts.

In this section, we show that the combination described in Proposition 3.1 does preserve politeness over some of the sorts. This lays the foundation for the more general combination theorem described in Section B.3.

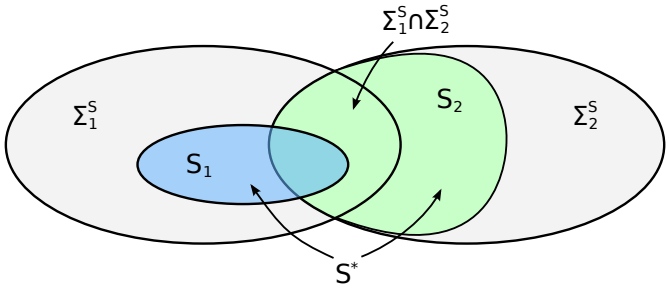


Figure B.1: Diagram for Theorem B.1.

Theorem B.1. *Let Σ_1 and Σ_2 be signatures and let $S = \Sigma_1^{\mathbb{S}} \cap \Sigma_2^{\mathbb{S}}$. If*

1. *T_1 is a Σ_1-theory polite with respect to $S_1 \subseteq \Sigma_1^{\mathbb{S}}$,*

[2] See Theorem 9 in [93] for the statement, or [94] for a full proof. The theorem there is in the more general setting of order-sorted logic, of which ordinary many-sorted logic is a simple instance.

2. *T_2 is a Σ_2-theory polite with respect to $S_2 \subseteq \Sigma_2^{\mathbb{S}}$,*

3. *$S \subseteq S_2$,*

then $T_1 \oplus T_2$ is polite with respect to $S^* = S_1 \cup \left(S_2 \setminus \Sigma_1^{\mathbb{S}}\right)$.

Proof. First we prove that the combined theory is smooth with respect to S^*. Let $T = T_1 \oplus T_2$, $\Sigma = \Sigma_1 \cup \Sigma_2$, and assume that ϕ is a conjunction of flat Σ-literals satisfiable in a T-interpretation $\mathcal{A}$. We are given cardinalities $\kappa_\sigma \geq |A_\sigma|$, for all $\sigma \in S^*$, and we must construct a new T-interpretation satisfying ϕ that obeys the given cardinalities.

We can separate ϕ into a Σ_1-part ϕ_1 and a Σ_2-part ϕ_2 such that $\phi = \phi_1 \wedge \phi_2$. Let $V_i = \mathsf{vars}(\phi_i)$. We know that:

$$\mathcal{A}^{\Sigma_1, V_1} \models_{T_1} \phi_1 \ ,$$
$$\mathcal{A}^{\Sigma_2, V_2} \models_{T_2} \phi_2 \ .$$

Since T_2 is finitely witnessable, we know there is a witness function $\mathit{witness}_2$ such that ϕ_2 is T_2-equivalent to $(\exists \overrightarrow{w})\psi_2$ for $\psi_2 = \mathit{witness}_2(\phi_2)$. Since the variables in $\overrightarrow{w}$ are fresh, we can assume without loss of generality that $\mathcal{A} \models \psi_2$. If we then let $V_2' = V_2 \cup \mathsf{vars}(\overrightarrow{w})$, we have:

$$\mathcal{A}^{\Sigma_1, V_1} \models_{T_1} \phi_1 \ ,$$
$$\mathcal{A}^{\Sigma_2, V_2'} \models_{T_2} \psi_2 \ .$$

Now, let $V_S = \mathsf{vars}_S(\psi_2)$ and $V_{\overline{S}} = \mathsf{vars}_{S_2 \setminus S}(\psi_2)$ and let δ_{V_S} and $\delta_{V_{\overline{S}}}$ be the (unique) arrangements of these sets of variables that are satisfied by $\mathcal{A}$. We can add these arrangements (letting $V_1' = V_1 \cup \mathsf{vars}_S(\psi_2)$) to obtain:

$$\mathcal{A}^{\Sigma_1, V_1'} \models_{T_1} \phi_1 \wedge \delta_{V_S} \ ,$$
$$\mathcal{A}^{\Sigma_2, V_2'} \models_{T_2} \psi_2 \wedge \delta_{V_S} \wedge \delta_{V_{\overline{S}}} \ .$$

Because T_1 is smooth with respect to S_1, we can lift the domains of sorts $\sigma \in S_1$ to cardinalities κ_σ, i.e. there is a T_1-interpretation $\mathcal{B}$ that satisfies $\phi_1 \wedge \delta_{V_S}$ with $|B_\sigma| = \kappa_\sigma$ for $\kappa \in S_1$. We can't assume anything about the rest of the sorts, so let $\kappa'_\sigma = |B_\sigma|$ for $\sigma \in S \setminus S_1$.

Next, because T_2 is finitely witnessable with respect to S_2, we know that there is a T_2-interpretation $\mathcal{C}$ that satisfies $\psi_2 \wedge \delta_{V_S} \wedge \delta_{V_{\overline{S}}}$ such that for $\sigma \in S_2$ we have

$$C_\sigma = \left[\mathsf{vars}_\sigma(\psi_2 \wedge \delta_{V_S} \wedge \delta_{V_{\overline{S}}})\right]^{\mathcal{C}} = \left[\mathsf{vars}_\sigma(V_S \cup V_{\overline{S}})\right]^{\mathcal{C}} \quad .$$

Consider the sizes of the domains in $\mathcal{C}$:

- for $\sigma \in S \cap S_1$, we have that $|C_\sigma| \leq |B_\sigma| = \kappa_\sigma$, since both $\mathcal{C}$ and $\mathcal{B}$ agree on δ_{V_S};
- for $\sigma \in S \setminus S_1$, we have that $|C_\sigma| \leq |B_\sigma| = \kappa'_\sigma$, since both $\mathcal{C}$ and $\mathcal{B}$ agree on δ_{V_S};
- for $\sigma \in S_2 \setminus S$, we have that $|C_\sigma| \leq |A_\sigma| \leq \kappa_\sigma$, since both $\mathcal{C}$ and $\mathcal{A}$ agree on $\delta_{V_{\overline{S}}}$.

Finally, because T_2 is smooth with respect to S_2, we know there is a T_2-structure $\mathcal{D}$ that satisfies $\psi_2 \wedge \delta_{V_S} \wedge \delta_{V_{\overline{S}}}$ such that:

- for $\sigma \in S \cap S_1$, we have that $|D_\sigma| = |B_\sigma| = \kappa_\sigma$;
- for $\sigma \in S \setminus S_1$, we have that $|D_\sigma| = |B_\sigma| = \kappa'_\sigma$;
- for $\sigma \in S_2 \setminus S$, we have $|D_\sigma| = \kappa_\sigma$.

Since the structures $\mathcal{B}$ and $\mathcal{D}$ agree on the sizes of the shared sorts, and they agree on the arrangement of the common variables, we know from Theorem 3.5 that we can combine them into a T-interpretation $\mathcal{F}$ that satisfies $\phi_1 \wedge \psi_2$ and has the required cardinalities. This interpretation also satisfies ϕ, so T is smooth with respect to S^*.

The second part of the proof requires showing that T is finitely witnessable with respect to S^*. Let ϕ be a conjunction of flat Σ-literals. Again, we can separate ϕ into $\phi_1 \wedge \phi_2$ such that ϕ_1 is a Σ_1-formula and ϕ_2 is a Σ_2-formula. Since T_1 and T_2 are both finitely witnessable (with respect to S_1 and S_2 respectively), there are computable witness functions $witness_1$ and $witness_2$. We define the witness function $witness$ for T using the following steps:

1. compute the T_2 witness function $\psi_2 = witness_2(\phi_2)$;
2. let $\mathcal{E}$ be the set of all equivalence relations over $V_S = \mathsf{vars}_S(\psi_2)$;
3. compute the T_1-part of the witness function

$$\psi_1 = \bigvee_{E \in \mathcal{E}} witness_1(\phi_1 \wedge \delta(E)) \ ;$$

4. let $\psi = \mathit{witness}(\phi) = \psi_1 \wedge \psi_2$.

To show the first requirement of Definition 3.7, suppose we have a T-interpretation $\mathcal{A}$ such that

$$\mathcal{A} \vDash \phi_1 \wedge \phi_2 \ .$$

We can use the T_2-equivalence of applying $\mathit{witness}_2$ to obtain

$$\mathcal{A} \vDash \phi_1 \wedge \exists \overrightarrow{w}_2 . \psi_2 \ ,$$

where $\overrightarrow{w}_2 = \mathsf{vars}(\psi_2) \setminus \mathsf{vars}(\phi_2)$. It follows that we can find a suitable vector of elements $\overrightarrow{a}_2$ such that

$$\mathcal{A}\{\overrightarrow{w}_2 \leftarrow \overrightarrow{a}_2\} \vDash \phi_1 \wedge \psi_2 \ .$$

Now, let E_S be the unique equivalence relation over $V_S = \mathsf{vars}_S(\psi_2)$ compatible with $\mathcal{A}\{\overrightarrow{w}_2 \leftarrow \overrightarrow{a}_2\}$. Adding the arrangement $\delta(E_S)$ and using the T_1-equivalence of applying $\mathit{witness}_1$ we further obtain

$$\begin{aligned} &\mathcal{A}\{\overrightarrow{w}_2 \leftarrow \overrightarrow{a}_2\} \vDash \phi_1 \wedge \delta(E_S) \wedge \psi_2 \ , \\ &\mathcal{A}\{\overrightarrow{w}_2 \leftarrow \overrightarrow{a}_2\} \vDash \exists \overrightarrow{w}_1 . \mathit{witness}_1(\phi_1 \wedge \delta(E_S)) \wedge \psi_2 \ , \end{aligned}$$

where $\overrightarrow{w}_1 = \mathsf{vars}(\mathit{witness}_1(\phi_1 \wedge \delta(E_S))) \setminus \mathsf{vars}(\phi_1 \wedge \delta(E_S))$. Since we always assume that variables introduced by witness functions are fresh, we can safely conclude that $\overrightarrow{w}_1$ and $\overrightarrow{w}_2$ are disjoint and thus there is a suitable $\overrightarrow{a}_1$ such that

$$\mathcal{A}\{\overrightarrow{w}_1 \leftarrow \overrightarrow{a}_1, \overrightarrow{w}_2 \leftarrow \overrightarrow{a}_2\} \vDash \mathit{witness}_1(\phi_1 \wedge \delta(E_S)) \wedge \psi_2 \ .$$

Let $\overrightarrow{w}_3$ be the variables from $\mathsf{vars}(\psi) \setminus \mathsf{vars}(\phi)$ not already in $\overrightarrow{w}_1$ or $\overrightarrow{w}_2$. Clearly there is an $\overrightarrow{a}_3$ such that

$$\mathcal{A}\{\overrightarrow{w}_1 \leftarrow \overrightarrow{a}_1, \overrightarrow{w}_2 \leftarrow \overrightarrow{a}_2, \overrightarrow{w}_3 \leftarrow \overrightarrow{a}_3\} \vDash \mathit{witness}_1(\phi_1 \wedge \delta(E_S)) \wedge \psi_2 \ . \tag{B.1}$$

Finally, since $\mathit{witness}_1(\phi_1 \wedge \delta(E_S))$ entails ψ_1, we can conclude

$$\mathcal{A}\{\overrightarrow{w}_1 \leftarrow \overrightarrow{a}_1, \overrightarrow{w}_2 \leftarrow \overrightarrow{a}_2, \overrightarrow{w}_3 \leftarrow \overrightarrow{a}_3\} \vDash \bigvee_{E \in \mathcal{E}} \mathit{witness}_1(\phi_1 \wedge \delta(E)) \wedge \psi_2 \ , \tag{B.2}$$

and thus

$$\mathcal{A} \vDash \exists \overrightarrow{w}_1 \exists \overrightarrow{w}_2 \exists \overrightarrow{w}_3 . \psi \ .$$

To show the implication in the other direction, each step is straightforward except the step from equation B.2 to equation B.1. Notice however, that because of the first property of witness functions, if a T_1 interpretation satisfies $witness_1(\phi_1 \wedge \delta(E))$, then it also satisfies $\delta(E)$. Now, since exactly one arrangement $\delta(E)$ is true in a particular interpretation, this means that exactly one of the disjuncts holds.

To see that the second requirement of Definition 3.7 is also satisfied, let $\mathcal{A}$ be a T-interpretation satisfying $\psi \wedge \delta_{S^*}$, where δ_{S^*} is an arrangement over a set V of variables with sorts in S^*. We will assume that $\mathsf{vars}_{S^*}(\psi) \subseteq V$, as we can always add the extra variables from ψ to the arrangement while keeping compatibility with $\mathcal{A}$. This does not affect the correctness of our argument: we will show that there is an interpretation $\mathcal{E}$ such that $\mathcal{E} \models_T \psi \wedge \delta_{S^*}$ and $E_\sigma = [\mathsf{vars}_\sigma(\psi \wedge \delta_{S^*})]^{\mathcal{E}}$; notice that if extra variables from ψ were included in δ_{S^*}, we can remove them and the same interpretation $\mathcal{E}$ still has the desired properties.

In the following, for any set U of sorts, we will abbreviate $\delta_{\mathsf{vars}_U(V)}$ as δ_U. Note that δ_{S^*} can be decomposed into:

$$\delta_{S^*} = \delta_{S_1 \setminus S} \wedge \delta_{S_1 \cap S} \wedge \delta_{S_2 \setminus S} \ .$$

We can construct an additional variable arrangement $\delta_{S \setminus S_1}$ over the variables $\mathsf{vars}_{S \setminus S_1}(\psi_2)$ that is compatible with $\mathcal{A}$. These arrangements are all true in $\mathcal{A}$, so letting $V_i = \mathsf{vars}_{\Sigma_i^{\mathcal{S}}}(\psi \wedge \delta_{S^*})$ we have:

$$\mathcal{A}^{\Sigma_1, V_1} \models_{T_1} \left(\psi_1 \wedge \delta_{S_1 \setminus S} \wedge \delta_{S_1 \cap S} \wedge \delta_{S \setminus S_1}\right) = \Psi_1 \ ,$$
$$\mathcal{A}^{\Sigma_2, V_2} \models_{T_2} \left(\psi_2 \wedge \delta_{S_2 \setminus S} \wedge \delta_{S_1 \cap S} \wedge \delta_{S \setminus S_1}\right) = \Psi_2 \ .$$

Expanding the first equation (and dropping the last arrangement) we get that

$$\mathcal{A}^{\Sigma_1, V_1} \models_{T_1} \bigvee_{E \in \mathcal{E}} witness_1(\phi_1 \wedge \delta(E)) \wedge \delta_{S_1 \setminus S} \wedge \delta_{S_1 \cap S} \ .$$

Note that exactly one of the arrangements $\delta(E)$ is satisfied by $\mathcal{A}^{\Sigma_1, V_1}$. Call this arrangement $\delta(E_S)$. Because of the T_1-equivalence of applying $witness_1$, we have

$$\mathcal{A}^{\Sigma_1, V_1} \models_{T_1} witness_1(\phi_1 \wedge \delta(E_S)) \wedge \delta_{S_1 \setminus S} \wedge \delta_{S_1 \cap S} = \Psi_1' \ .$$

Now, because T_1 is finitely witnessable over S_1, we can obtain a T_1-interpretation $\mathcal{B}$ such that

$$\mathcal{B} \models_{T_1} \Psi_1' \ ,$$

and for all $\sigma \in S_1$ we have $B_\sigma = [\mathsf{vars}_\sigma(\Psi_1')]^{\mathcal{B}}$. Note that though Ψ_1' and Ψ_1 differ, we have that $\mathsf{vars}(\Psi_1') \subseteq \mathsf{vars}(\Psi_1)$. We can thus extend $\mathcal{B}$ arbitrarily to interpret all of the variables in $\mathsf{vars}(\Psi_1)$ so that $B_\sigma = [\mathsf{vars}_\sigma(\Psi_1)]^{\mathcal{B}}$ for $\sigma \in S_1$.

Because $\mathcal{B} \models \Psi_1'$, we know that $\mathcal{B}$ will also satisfy $\delta(E_S)$ (by the first property of $witness_1$). Now, since $\delta(E_S)$ includes all the variables in $\delta_{S \setminus S_1}$ by definition (they both only arrange the variables from ψ_2), and because both $\delta(E_S)$ and $\delta_{S \setminus S_1}$ are satisfied by the same interpretation $\mathcal{A}$, we know that $\mathcal{B}$ also satisfies $\delta_{S \setminus S_1}$:

$$\mathcal{B} \vDash_{T_1} witness_1(\phi_1 \wedge \delta(E_S)) \wedge \delta_{S_1 \setminus S} \wedge \delta_{S_1 \cap S} \wedge \delta_{S \setminus S_1} \ .$$

Since one disjunct of ψ_1 is satisfied, we can conclude that

$$\mathcal{B} \vDash_{T_1} \Psi_1 \ .$$

Now, let's consider Ψ_2. Because T_2 is finitely witnessable over S_2, we can obtain a T_2-interpretation $\mathcal{C}$ satisfying Ψ_2 such that for $\sigma \in S_2$, we have $C_\sigma = [\mathsf{vars}_\sigma(\Psi_2)]^{\mathcal{C}}$.

Since both $\mathcal{B}$ and $\mathcal{C}$ satisfy the arrangement $\delta_{S_1 \cap S}$ and this arrangement contains all the variables of sorts $S_1 \cap S$ from ψ_1 and ψ_2, it follows that for $\sigma \in S_1 \cap S$, we have $|B_\sigma| = |C_\sigma|$. For the other shared sorts $\sigma \in S \setminus S_1$, we have that $|C_\sigma| \leq |B_\sigma|$ because Ψ_1 and Ψ_2 agree on $\delta_{\{\sigma\}}$, and we know that C_σ does not interpret any elements beyond those named by variables in $\delta_{\{\sigma\}}$ (since we chose $\delta_{S \setminus S_1}$ to include $\mathsf{vars}_{S \setminus S_1}(\psi_2)$).

As in the proof of smoothness, we now proceed to combine the two structures. By smoothness of T_2 with respect to S_2 we can lift the structure $\mathcal{C}$ to a structure $\mathcal{D}$ that satisfies Ψ_2, such that

- $|D_\sigma| = |B_\sigma| = |C_\sigma|$ for $\sigma \in S_1 \cap S_2$,
- $|D_\sigma| = |B_\sigma| \geq |C_\sigma|$ for $\sigma \in S \setminus S_1$,
- $|D_\sigma| = |C_\sigma|$ for $\sigma \in S_2 \setminus S$.

Interpretations $\mathcal{B}$ and $\mathcal{D}$ agree on the arrangements $\delta_{S \cap S_1} \wedge \delta_{S \setminus S_1}$. These arrangements include all of the shared variables of Ψ_1 and Ψ_2. For sorts in $S \cap S_1$, this follows by our assumption that δ_{S^*} includes all the variables in ψ of sorts in S^*. For sorts in $S \setminus S_1$, this follows from the fact that $\delta_{S \setminus S_1}$ includes all the variables in $\mathsf{vars}_{S \setminus S_1}(\psi_2)$.

Finally, by Theorem 3.5, given interpretations $\mathcal{B}$ and $\mathcal{D}$, we can find an interpretation $\mathcal{E}$ satisfying $\Psi_1 \wedge \Psi_2$, because they agree on the arrangement over the shared variables of Ψ_1 and Ψ_2, and have the same cardinalities over the shared sorts. Moreover, since we are keeping the cardinalities of $\mathcal{B}$ over S_1 and $\mathcal{C}$ over $S_2 \setminus S$, and these cardinalities are determined by the arrangements in δ_S^*, and $\mathsf{vars}_{S^*}(\Psi_1 \wedge \Psi_2) = \mathsf{vars}(\delta_{S^*})$, we will also have that for all $\sigma \in S^*$, $E_\sigma = [\mathsf{vars}_\sigma(\Psi_1 \wedge \Psi_2)]^{\mathcal{E}} = [\mathsf{vars}_\sigma(\psi \wedge \delta_{S^*})]^{\mathcal{E}}$, as required. This concludes the proof of finite witnessability and shows that $T_1 \oplus T_2$ is polite with respect to S^*. □

We illustrate the application of the theorem with an example using two theories of arrays.

Example B.1. Let $T_{\mathsf{array},1}$ and $T_{\mathsf{array},2}$ be two theories of arrays over the following sets of sorts respectively

$$S_1 = \{\mathsf{array}_1, \mathsf{index}_1, \mathsf{elem}_1\} \ ,$$
$$S_2 = \{\mathsf{array}_2, \mathsf{index}_2, \mathsf{array}_1\} \ .$$

These two theories together model two-dimensional arrays with indices in index_1 and index_2, and elements in elem_1.

We know that the theory $T_{\mathsf{array},1}$ is polite with respect to $S_1^* = \{\mathsf{index}_1, \mathsf{elem}_1\}$, and the theory $T_{\mathsf{array},2}$ is polite with respect to $S_2^* = \{\mathsf{index}_2, \mathsf{array}_1\}$. Using Theorem B.1, we know that we can combine them into a theory T_{array} that is polite with respect to the set

$$S_1^* \cup (S_2^* \setminus \{\mathsf{array}_1\}) = \{\mathsf{index}_1, \mathsf{index}_2, \mathsf{elem}_1\} \ .$$

This means that we can combine the theory of two-dimensional arrays with any other theories that operate over the elements and indices, even if they are not stably-infinite (such as bit-vectors for example).

An interesting corollary of Theorem B.1 is that, if both theories are polite with respect to the shared sorts then, analogously to Proposition B.1, we get a theory that is polite with respect to the union of the sorts.

Corollary B.1. *Let Σ_1 and Σ_2 be signatures. If*

- *T_1 is a Σ_1-theory polite with respect to $S_1 \subseteq \Sigma_1^{\mathbb{S}}$,*
- *T_2 is a Σ_2-theory polite with respect to $S_2 \subseteq \Sigma_2^{\mathbb{S}}$,*
- *$\Sigma_1^{\mathbb{S}} \cap \Sigma_2^{\mathbb{S}} = S_1 \cap S_2$,*

then $T_1 \oplus T_2$ is polite with respect to $S_1 \cup S_2$.

B.3 Combining Multiple Polite Theories

Given a Σ_1-theory T_1, polite with respect to sorts S_1, and a Σ_2-theory T_2, polite with respect to sorts S_2, we will denote their combination using the combination framework for polite theories as $T_1 \oplus_{\mathsf{p}} T_2$. Here, $\oplus_{\mathsf{p}}$ is a partial, asymmetric operator: $T_1 \oplus_{\mathsf{p}} T_2$ is defined as $T_1 \oplus T_2$ if $\Sigma_1^{\mathbb{S}} \cap \Sigma_2^{\mathbb{S}} \subseteq S_2$ and is undefined otherwise. Note that if defined, $T_1 \oplus_{\mathsf{p}} T_2$ is polite with respect to $S_1 \cup (S_2 \setminus \Sigma_1^{\mathbb{S}})$ by Theorem B.1.

Because of the asymmetry in its definition, it is not obvious whether $\oplus_{\mathsf{p}}$ is associative or commutative. When dealing with several theories there might be several ways we could try to combine them: given theories T_1, T_2 and T_3, we could first combine T_1 and T_2 into $T_1 \oplus_{\mathsf{p}} T_2$ and then combine the result with T_3 to obtain $(T_1 \oplus_{\mathsf{p}} T_2) \oplus_{\mathsf{p}} T_3$. Or we might opt to combine T_2 and T_3 first and then combine T_1 with $T_2 \oplus_{\mathsf{p}} T_3$ to get the same theory $T_1 \oplus_{\mathsf{p}} (T_2 \oplus_{\mathsf{p}} T_3)$. Some of these operations might not be defined, and if they are, it is not obvious whether Theorem B.1 ensures that the resulting theories are polite with respect to the same set of sorts.

Since, as explained above, there are several ways of obtaining a combined theory using the combination framework, we will write $T_1 \leftrightarrow_p T_2$ to denote that T_1 and T_2 are either both undefined or both defined, and in the latter case that T_1 and T_2 are polite with respect to the same sets of sorts.

Lemma B.3. *Let T_i be a Σ_i-theory, polite with respect to sorts S_i, for $i = 1, 2, 3$. Then*

$$T_1 \oplus_{\mathsf{p}} (T_2 \oplus_{\mathsf{p}} T_3) \leftrightarrow_p (T_1 \oplus_{\mathsf{p}} T_2) \oplus_{\mathsf{p}} T_3 \ . \tag{B.3}$$

Proof. The proof of this statement primarily relies on simple manipulations in basic set theory. For convenience, it might be easier to understand the result by looking at Figure B.2.

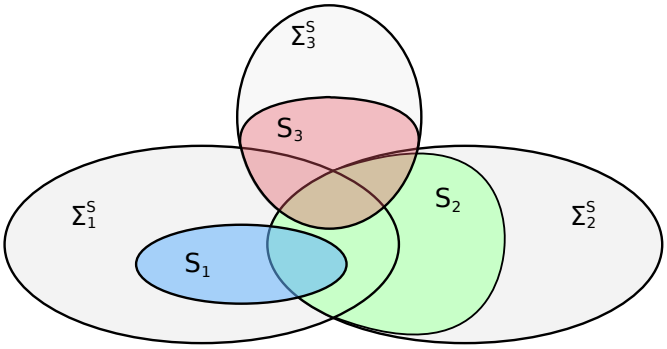


Figure B.2: Diagram for Lemma B.3.

We first note that the theory combination operator $\oplus$ is clearly associative. Thus, it suffices to show that if one side of (B.3) is defined, then the other is also, and that they are polite with respect to the same sets of sorts.

Assume that the right-associative combination $T_1 \oplus_{\mathsf{p}} (T_2 \oplus_{\mathsf{p}} T_3)$ is defined. This implies that we have

$$\Sigma_2^{\mathbb{S}} \cap \Sigma_3^{\mathbb{S}} \subseteq S_3 \ . \tag{B.4}$$

Using Theorem B.1 we know that $T_2 \oplus T_3$ is polite with respect to $S_2 \cup (S_3 \setminus \Sigma_2^{\mathbb{S}})$. Then, since we can combine T_1 with $T_2 \oplus T_3$, we must have $\Sigma_1^{\mathbb{S}} \cap (\Sigma_2^{\mathbb{S}} \cup \Sigma_3^{\mathbb{S}}) \subset S_2 \cup (S_3 \setminus \Sigma_2^{\mathbb{S}})$ which is equivalent to the following

$$\Sigma_1^{\mathbb{S}} \cap \Sigma_2^{\mathbb{S}} \subset S_2 \cup (S_3 \setminus \Sigma_2^{\mathbb{S}}) \ , \tag{B.5}$$

$$\Sigma_1^{\mathbb{S}} \cap \Sigma_3^{\mathbb{S}} \subset S_2 \cup (S_3 \setminus \Sigma_2^{\mathbb{S}}) \ . \tag{B.6}$$

It follows from (B.5) (intersecting both sides with $\Sigma_2^{\mathbb{S}}$) that

$$\Sigma_1^{\mathbb{S}} \cap \Sigma_2^{\mathbb{S}} \subseteq S_2 \ ,$$

which is enough to conclude that we can combine T_1 and T_2 into $T_1 \oplus T_2$. To be able to combine $T_1 \oplus T_2$ with T_3 we must show that

$$(\Sigma_1^{\mathbb{S}} \cup \Sigma_2^{\mathbb{S}}) \cap \Sigma_3^{\mathbb{S}} \subseteq S_3,$$

which is equivalent to

$$\Sigma_1^{\mathbb{S}} \cap \Sigma_3^{\mathbb{S}} \subseteq S_3 \ , \tag{B.7}$$

$$\Sigma_2^{\mathbb{S}} \cap \Sigma_3^{\mathbb{S}} \subseteq S_3 \ . \tag{B.8}$$

We have that (B.4) and (B.8) are the same. To show (B.7), it is sufficient to show both of the following

$$(\Sigma_1^{\mathbb{S}} \cap \Sigma_3^{\mathbb{S}}) \cap \Sigma_2^{\mathbb{S}} \subseteq S_3 \cap \Sigma_2^{\mathbb{S}} \ , \tag{B.9}$$

$$(\Sigma_1^{\mathbb{S}} \cap \Sigma_3^{\mathbb{S}}) \setminus \Sigma_2^{\mathbb{S}} \subseteq S_3 \setminus \Sigma_2^{\mathbb{S}} \ . \tag{B.10}$$

Equation (B.9) follows directly from (B.8) (intersect both sides with $\Sigma_2^{\mathbb{S}}$ and then intersect just the left side with $\Sigma_1^{\mathbb{S}}$). Equation (B.10) is obtained by subtracting $\Sigma_2^{\mathbb{S}}$ from both sides of (B.6). This shows that the left-associative combination $(T_1 \oplus_{\mathsf{p}} T_2) \oplus_{\mathsf{p}} T_3$ is defined.

In the opposite direction, assume that the left-associative combination $(T_1 \oplus_{\mathsf{p}} T_2) \oplus_{\mathsf{p}} T_3$ is defined. For this to be possible we need to combine T_1 and T_2 first, so it must be the case that

$$\Sigma_1^{\mathbb{S}} \cap \Sigma_2^{\mathbb{S}} \subseteq S_2 \ . \tag{B.11}$$

Next, to combine $T_1 \oplus T_2$ with T_3, we must have

$$(\Sigma_1^{\mathbb{S}} \cup \Sigma_2^{\mathbb{S}}) \cap \Sigma_3^{\mathbb{S}} \subseteq S_3 \ . \tag{B.12}$$

This is equivalent to

$$\Sigma_1^{\mathbb{S}} \cap \Sigma_3^{\mathbb{S}} \subseteq S_3 \ , \tag{B.13}$$

$$\Sigma_2^{\mathbb{S}} \cap \Sigma_3^{\mathbb{S}} \subseteq S_3 \ . \tag{B.14}$$

From (B.14) we immediately get that we can combine theories T_2 and T_3 into $T_2 \oplus T_3$ which is polite with respect to $S_2 \cup (S_3 \setminus \Sigma_2^{\mathbb{S}})$. To be able to combine T_1 with $T_2 \oplus T_3$ we need to show that

$$\Sigma_1^{\mathbb{S}} \cap (\Sigma_2^{\mathbb{S}} \cup \Sigma_3^{\mathbb{S}}) \subseteq S_2 \cup (S_3 \setminus \Sigma_2^{\mathbb{S}}) \ .$$

This is in turn equivalent to

$$\Sigma_1^{\mathbb{S}} \cap \Sigma_2^{\mathbb{S}} \subseteq S_2 \cup (S_3 \setminus \Sigma_2^{\mathbb{S}}) \ , \tag{B.15}$$

$$\Sigma_1^{\mathbb{S}} \cap \Sigma_3^{\mathbb{S}} \subseteq S_2 \cup (S_3 \setminus \Sigma_2^{\mathbb{S}}) \ . \tag{B.16}$$

From (B.11) we immediately get (B.15). To show (B.16), it is sufficient to show both of the following

$$(\Sigma_1^{\mathbb{S}} \cap \Sigma_3^{\mathbb{S}}) \cap \Sigma_2^{\mathbb{S}} \subseteq S_2 \ , \tag{B.17}$$

$$(\Sigma_1^{\mathbb{S}} \cap \Sigma_3^{\mathbb{S}}) \setminus \Sigma_2^{\mathbb{S}} \subseteq S_3 \setminus \Sigma_2^{\mathbb{S}} \ . \tag{B.18}$$

Equation (B.17) follows directly from (B.11). Equation (B.18) is obtained by subtracting $\Sigma_2^{\mathbb{S}}$ from both sides of (B.13). This proves that the right-associative combination $T_1 \oplus_{\mathsf{p}} (T_2 \oplus_{\mathsf{p}} T_3)$ is defined.

To show that the order of combination has no impact on the resulting sets of polite sorts, we compute the sets for both cases. If we consider the combination $T_1 \oplus_{\mathsf{p}} (T_2 \oplus_{\mathsf{p}} T_3)$, we would first get that $T_2 \oplus T_3$ is polite with respect to $S_2 \cup (S_3 \setminus \Sigma_2^{\mathbb{S}})$. Combining the resulting theory with T_1 gives the final set of polite sorts

$$S_1 \cup (S_2 \cup (S_3 \setminus \Sigma_2^{\mathbb{S}})) \setminus \Sigma_1^{\mathbb{S}} = S_1 \cup (S_2 \setminus \Sigma_1^{\mathbb{S}}) \cup (S_3 \setminus (\Sigma_1^{\mathbb{S}} \cup \Sigma_2^{\mathbb{S}})) \ . \tag{B.19}$$

Combining in the other direction, we first get that $T_1 \oplus T_2$ is polite with respect to $S_1 \cup S_2 \setminus \Sigma_1^{\mathbb{S}}$. Combining the result with T_3 gives the set of sorts (B.19). □

Lemma B.3 gives us the associativity of $\oplus_{\mathsf{p}}$. The next lemma shows that we can also achieve commutativity if both theories are polite with respect to at least the shared sorts.

Lemma B.4. *Let T_i be a Σ_i-theory polite with respect to the set of sorts $S_i \subseteq \Sigma_i^{\mathbb{S}}$, for $i = 1, 2$. Then the following are equivalent*

1. $T_1 \oplus_{\mathsf{p}} T_2 \leftrightarrow_p T_2 \oplus_{\mathsf{p}} T_1$;
2. $\Sigma_1^{\mathbb{S}} \cap \Sigma_2^{\mathbb{S}} = S_1 \cap S_2$.

Proof. If both $T_1 \oplus_{\mathsf{p}} T_2$ and $T_2 \oplus_{\mathsf{p}} T_1$ are defined, then we can use either T_1 or T_2 as the polite theory in the combination framework. This means that both $\Sigma_1^{\mathbb{S}} \cap \Sigma_2^{\mathbb{S}} \subseteq S_1$ and $\Sigma_1^{\mathbb{S}} \cap \Sigma_2^{\mathbb{S}} \subseteq S_2$, which implies that $\Sigma_1^{\mathbb{S}} \cap \Sigma_2^{\mathbb{S}} = S_1 \cap S_2$. In the other direction, if $\Sigma_1^{\mathbb{S}} \cap \Sigma_2^{\mathbb{S}} = S_1 \cap S_2$ holds, then (1) is a consequence of Corollary B.1. □

Now we give a general theorem for combining multiple theories in a sequential manner.

Theorem B.2. *Let T_i be a Σ_i-theory, for $1 \leq i \leq n$. Assume that*

- *theories T_i have no function or predicate symbols in common;*
- *the quantifier-free satisfiability problem of T_i is decidable, for $1 \leq i \leq n$;*
- *T_i is polite with respect to S_i, for $1 \leq i \leq n$;*
- *$\Sigma_i^{\mathbb{S}} \cap \Sigma_j^{\mathbb{S}} \subseteq S_j$, for $1 \leq i < j \leq n$.*

Then the quantifier-free satisfiability problem for $T = T_1 \oplus \cdots \oplus T_n$ is decidable. Moreover, the resulting theory T is polite with respect to the set of sorts

$$S = \bigcup_{j=1}^{n} \left(S_j \setminus (\bigcup_{i<j} \Sigma_i^{\mathbb{S}}) \right) \quad .$$

Proof. We prove the statement by induction on the number of theories n. In the base case, when $n = 2$, this directly follows from Proposition 3.1 and Theorem B.1, i.e. if we have that $\Sigma_1^{\mathbb{S}} \cap \Sigma_2^{\mathbb{S}} \subseteq S_2$, then we know how to devise the decision procedure for $T_1 \oplus T_2$ using the algorithm from [79]. Moreover, the resulting theory is polite with respect to $S_1 \cup (S_2 \setminus \Sigma_1^{\mathbb{S}})$.

Assume that the statement holds for $n > 1$ and consider the case for $n + 1$. By the inductive hypothesis, we have that the theory $T = T_1 \oplus \cdots \oplus T_n$ over the signature $\Sigma = \Sigma_1 \cup \ldots \cup \Sigma_n$ is decidable and polite with respect to

$$S = \bigcup_{j=1}^{n} \left(S_j \setminus (\bigcup_{i<j} \Sigma_i^{\mathbb{S}}) \right) \quad .$$

We have that $\Sigma_i^{\mathbb{S}} \cap \Sigma_{n+1}^{\mathbb{S}} \subseteq S_{n+1}$, for $1 \leq i \leq n$. Taking the union of these we get that

$$(\Sigma_1^{\mathbb{S}} \cup \ldots \cup \Sigma_n^{\mathbb{S}}) \cap \Sigma_{n+1}^{\mathbb{S}} = \Sigma^{\mathbb{S}} \cap \Sigma_{n+1}^{\mathbb{S}} \subseteq S_{n+1}$$

Since quantifier-free satisfiability in both T and T_{n+1} are decidable and the theories satisfy the conditions of Proposition 3.1 and Theorem B.1, we know that quantifier-free satisfiability is decidable in the combination $T \oplus T_{n+1} = T_1 \oplus \cdots \oplus T_{n+1}$. Furthermore, the combination is polite

with respect to the set

$$S = \bigcup_{j=1}^{n} \left(S_j \setminus (\bigcup_{i<j} \Sigma_i^{\mathbb{S}}) \right) \cup (S_{n+1} \setminus \bigcup_{k=1}^{n} \Sigma_k^{\mathbb{S}}) = \bigcup_{j=1}^{n+1} \left(S_j \setminus (\bigcup_{i<j} \Sigma_i^{\mathbb{S}}) \right) \ .$$

This concludes the proof. □

Example B.2. Assume we have a theory of arrays $T_{\mathsf{array},1}$ over the sorts

$$\Sigma^{\mathbb{S}}_{\mathsf{array},1} = \{\mathsf{array}_1, \mathsf{index}_1, \mathsf{elem}\} \ ,$$

as well as theories of arrays $T_{\mathsf{array},k}$ over the sorts

$$\Sigma^{\mathbb{S}}_{\mathsf{array},k} = \{\mathsf{array}_k, \mathsf{index}_k, \mathsf{array}_{k-1}\} \ ,$$

for $k \geq 2$. These theories represent different layers in the theory of n-dimensional arrays. The theories satisfy the assumption of Theorem B.2 and thus we can combine them into the full theory

$$T_{\mathsf{array}} = T_{\mathsf{array},1} \oplus T_{\mathsf{array},2} \oplus \cdots \oplus T_{\mathsf{array},n} \ .$$

This theory is polite with respect to the union of all indices and elements

$$S = \{\mathsf{index}_1, \mathsf{index}_2, \ldots, \mathsf{index}_n, \mathsf{elem}\} \ .$$

Note that, although we are combining theories in a straightforward fashion, we could not have used Theorem 14 from [80] to achieve this combination, since the common intersection of the polite sets of sorts is empty, and the pairwise intersection of sorts is not. More importantly, we are able to easily deduce the politeness of the resulting theory.

We finish this section with a theorem that gives an easy complete method for checking whether we can combine a set of theories in the framework of multiple polite theories.

Theorem B.3. *Let T_1, T_2, ..., T_n be pairwise signature-disjoint theories such that individual quantifier-free T_i-satisfiability problems are decidable. The quantifier-free satisfiability problem of $T = T_1 \oplus \cdots \oplus T_n$ is decidable by iterating the polite combination method for two theories if and only if there is a reordering of the theories T_i that satisfies the conditions of Theorem B.2.*

Proof. The if direction is obvious. In the other direction, assume there is a way to combine the theories T_i using the framework. Then there exists some expression combining the T_i's using $\oplus_{\mathsf{p}}$ that is defined. Using the associativity of $\oplus_{\mathsf{p}}$ (from Lemma B.3), we can transform the expression into a sequential combination $(\cdots (T_{p_1} \oplus_{\mathsf{p}} T_{p_2}) \oplus_{\mathsf{p}} T_{p_3}) \oplus_{\mathsf{p}} \cdots \oplus_{\mathsf{p}} T_{n-1}) \oplus_{\mathsf{p}} T_n$ that satisfies the requirements of Theorem B.2. □

B.4 Theory Instantiations

The way theories are defined in Definition 3.1 is meant to be general, i.e. the sorts can be interpreted in any domain. But, sometimes we are interested in a variant of a theory obtained by identifying some of the sorts. For example, consider a theory of arrays with elements and indices, i.e. $\Sigma^{\mathbb{S}}_{\mathsf{array}} = \{\mathsf{array}, \mathsf{elem}, \mathsf{index}\}$. In practice, we often deal with a closely related theory of arrays in which the indices and the elements are from the same sort. Note that these two theories are indeed different – in the general theory of arrays, the well-sortedness prevents us from comparing indices with elements (the term $\mathsf{read}(a, i) \neq i$ is not well-sorted, for example). We will call this merging of sorts *theory instantiation by sort equality.*

Definition B.4 (Signature Instantiation). Let $\Sigma = (S, F, P)$ be a signature. We call $\Sigma_s^{\sigma_1 = \sigma_2} = (S', F', P')$ a signature instantiation by sort equality $\sigma_1 = \sigma_2$, for sorts $\sigma_1, \sigma_2 \in S$ and $s \notin S$, if the following holds:

- $S' = (S \setminus \{\sigma_1, \sigma_2\}) \cup \{s\}$;
- F' contains the same function symbols as F except that we replace σ_1 and σ_2 with s in every arity;
- P' contains the same predicate symbols as P except that we replace σ_1 and σ_2 with s in every arity.

To enable the translation of formulas from the instantiated signature to the original signature and vice versa, we will use the satisfiability-preserving (see Lemma B.6) syntactic formula transformation α that maps conjunctions of flat $\Sigma_s^{\sigma_1=\sigma_2}$-literals into formulas from the signature Σ. Given such a conjunction $\phi = \bigwedge_{1\leq k\leq m} l_k$, with $\mathsf{vars}_s(\phi) = \{v_1, v_2, \ldots, v_n\}$, we first introduce fresh variables $v_i^{\sigma_1}$ of sort σ_1, and $v_i^{\sigma_2}$ of sort σ_2, for $i = 1, \ldots, n$. The function α transforms the formula ϕ into

$$\alpha(\phi) \quad \triangleq \bigwedge_{1\leq k\leq m} \alpha_l(l_k) \quad \wedge \bigwedge_{1\leq i<j\leq n} \left(v_i^{\sigma_1} =_{\sigma_1} v_j^{\sigma_1} \leftrightarrow v_i^{\sigma_2} =_{\sigma_2} v_j^{\sigma_2}\right) \quad ,$$

The transformation α_l acts on the individual literals as follows:

- Literals of the form $x =_\sigma y$ and $x \neq_\sigma y$, where $\sigma \neq s$, are left unchanged.
- Literals of the form $x =_s y$ and $x \neq_s y$ are transformed into $x^{\sigma_1} =_{\sigma_1} y^{\sigma_1}$ and $x^{\sigma_1} \neq_{\sigma_1} y^{\sigma_1}$ respectively.[3]
- Literals of the form $x =_\sigma f(y_1, \ldots, y_n)$, where $\sigma \neq s$, are transformed into $x =_\sigma f(y_1^*, \ldots, y_n^*)$. The variables y_i^* are taken to comply with the original arity of f in Σ, i.e.

$$y_i^* = \begin{cases} y_i^{\sigma_1} & \text{if } y_i \text{ should be of sort } \sigma_1 \text{ in the arity of } f \text{ in } \Sigma, \\ y_i^{\sigma_2} & \text{if } y_i \text{ should be of sort } \sigma_2 \text{ in the arity of } f \text{ in } \Sigma, \\ y_i & \text{otherwise.} \end{cases}$$

- Literals of the form $x =_s f(y_1, \ldots, y_n)$ are transformed into either $x^{\sigma_1} =_{\sigma_1} f(y_1^*, \ldots, y_n^*)$ or $x^{\sigma_2} =_{\sigma_2} f(y_1^*, \ldots, y_n^*)$, depending on the sort of the co-domain of f in Σ.
- Literals of the form $p(y_1, \ldots, y_n)$ and $\neg p(y_1, \ldots, y_n)$ are transformed in a similar manner.

In the other direction, we define a transformation γ_V, where V is a set of variables of sort s, from Σ-formulas to $\Sigma_s^{\sigma_1=\sigma_2}$-formulas, as follows

$$\gamma_V(\phi) = \phi \wedge \bigwedge_{v\in V} (v^{\sigma_1} = v \wedge v^{\sigma_2} = v) \quad .$$

In the new formula variables formerly of sort σ_1 or σ_2 are now of sort s.

[3]The choice of σ_1 over σ_2 is arbitrary, as the right part of $\alpha(\phi)$ will force the same on the dual variables.

Definition B.5 (Theory Instantiation). Let Σ be a signature and $T = (\Sigma, \mathbf{A})$ be a Σ-theory. We call a theory $T_s^{\sigma_1=\sigma_2} = (\Sigma_s^{\sigma_1=\sigma_2}, \mathbf{B})$ the theory instantiated by sort equality $\sigma_1 = \sigma_2$, for sorts $\sigma_1, \sigma_2 \in \Sigma^{\mathbb{S}}$ and $s \notin \Sigma^{\mathbb{S}}$, when $\mathcal{B} \in \mathbf{B}$ iff

- there exists an $\mathcal{A} \in \mathbf{A}$ such that $B_s = A_{\sigma_1} = A_{\sigma_2}$, and $B_\sigma = A_\sigma$ for $\sigma \neq s$; and
- all the predicate and function symbols in $\Sigma_s^{\sigma_1=\sigma_2}$ are interpreted in $\mathcal{B}$ exactly the same as they are interpreted in $\mathcal{A}$.

The above definition simply restricts the original theory structures to those in which the sorts σ_1 and σ_2 are interpreted by the same domain. The lemma below shows that the result, $T_s^{\sigma_1=\sigma_2}$, is indeed a theory.

As we did with formulas, we define a transformation on structures (which we will also call α) that maps $\Sigma_s^{\sigma_1=\sigma_2}$-interpretations into Σ-interpretations. Given a $\Sigma_s^{\sigma_1=\sigma_2}$-interpretation $\mathcal{A}$, we construct the transformed structure $\mathcal{B} = \alpha(\mathcal{A})$ as follows. For sorts $\sigma \in \Sigma^{\mathbb{S}} \setminus \{\sigma_1, \sigma_2\}$, we define $B_\sigma = A_\sigma$. For the sorts σ_1 and σ_2, we define $B_{\sigma_1} = B_{\sigma_2} = A_s$. The set of variables interpreted by $\mathcal{B}$ includes all of those interpreted by $\mathcal{A}$, without the variables of sort s, and we define $v^{\mathcal{B}} = v^{\mathcal{A}}$. Finally, since $B_{\sigma_1} = B_{\sigma_2} = A_s$, we can simply define $f^{\mathcal{B}} = f^{\mathcal{A}}$ and $p^{\mathcal{B}} = p^{\mathcal{A}}$ for each function symbol f and predicate symbol p. Additionally, it is clear that if $\mathcal{A}$ is a $T_s^{\sigma_1=\sigma_2}$-interpretation, then $\alpha(\mathcal{A})$ will be a T-interpretation.

Lemma B.5. *Let T and $\mathbf{B}$ be as in Definition B.5, and let $\mathbf{Ax}$ be the set of closed Σ-formulas that defines T. The class $\mathbf{B}$ is exactly the set of $\Sigma_s^{\sigma_1=\sigma_2}$-structures that satisfies the set of formulas $\gamma_\varnothing(\mathbf{Ax}) = \{\gamma_\varnothing(\phi) \mid \phi \in \mathbf{Ax}\}$.*

Proof. First, for every $\mathcal{B} \in \mathbf{B}$ there is a Σ-structure $\mathcal{A} \in \mathbf{A}$ such that $\mathcal{A} \vDash \mathbf{Ax}$. By the definition of γ, we also have $\mathcal{B} \vDash \gamma_\varnothing(\mathbf{Ax})$. In the other direction, let $\mathcal{B}$ be a $\Sigma_s^{\sigma_1=\sigma_2}$-structure satisfying $\gamma_\varnothing(\mathbf{Ax})$. We define a Σ-structure $\mathcal{A} = \alpha(B)$. It follows that $\mathcal{A} \vDash \mathbf{Ax}$. This implies that $\mathcal{A} \in \mathbf{A}$ and hence $\mathcal{B} \in \mathbf{B}$. □

Our motivating example is the theory of arrays where we restrict the sorts elem and index to be equal to each other and to bv, i.e. we are interested in the theory $T_{\mathsf{array}}^{\mathsf{bv}} = (T_{\mathsf{array}})_{\mathsf{bv}}^{\mathsf{elem}=\mathsf{index}}$. We know that T_{array} is polite with respect to the sorts elem and index. We want to know whether it is also the case that $T_{\mathsf{array}}^{\mathsf{bv}}$ is polite with respect to the sort bv.

The main result of this section is to show that by merging two sorts σ_1 and σ_2 in a theory, we preserve the politeness of the theory: the new theory will be polite with respect to the same set of sorts as the original theory, modulo renaming of the instantiated sorts σ_1 and σ_2. Before proving this, we need the following lemma.

Lemma B.6. *Let Σ be a signature such that $\sigma_1, \sigma_2 \in \Sigma^{\mathbb{S}}$ and $s \notin \Sigma^{\mathbb{S}}$, and ϕ be a conjunction of flat $\Sigma_s^{\sigma_1=\sigma_2}$-literals. Furthermore, let $S \subseteq \Sigma^{\mathbb{S}}$ be such that $\sigma_1, \sigma_2 \in S$ and $S' = S \setminus \{\sigma_1, \sigma_2\} \cup \{s\}$. Then the following are equivalent:*

1. *ϕ is satisfiable in a $T_s^{\sigma_1=\sigma_2}$-interpretation $\mathcal{A}$ with $|A_\sigma| = \kappa_\sigma$ for $\sigma \in S'$;*
2. *$\alpha(\phi)$ is satisfiable in a T-interpretation $\mathcal{B}$ with $|B_{\sigma_1}| = |B_{\sigma_2}| = \kappa_s$, and $|B_\sigma| = \kappa_\sigma$ for $\sigma \in S \setminus \{\sigma_1, \sigma_2\}$.*

Proof. Assume that ϕ is satisfiable in a $T_s^{\sigma_1=\sigma_2}$-interpretation $\mathcal{A}$ with $|A_\sigma| = \kappa_\sigma$ for $\sigma \in S'$. Let $\mathcal{B} = \alpha(\mathcal{A})$. Then it is easy to see that the domains of B have the required sizes and $\exists \overrightarrow{v}.\alpha(\phi)$ will be satisfied by $\mathcal{B}$, where v is the vector of fresh variables introduced by α. Hence there is an interpretation $\mathcal{B}'$ that satisfies $\alpha(\phi)$ such that $|B'_{\sigma_1}| = |B'_{\sigma_2}| = \kappa_s$ and $|B_{sigma}| = \kappa_\sigma, \sigma \in S \setminus \{\sigma_1, \sigma_2\}$.

In the other direction, assume that $\alpha(\phi)$ is satisfiable in a T-interpretation $\mathcal{B}$ with $|B_\sigma| = \kappa_\sigma$, for $\sigma \in S \setminus \{\sigma_1, \sigma_2\}$, and $|B_{\sigma_1}| = |B_{\sigma_2}| = \kappa_s$. The domains B_{σ_1} and B_{σ_2} are of the same size κ_s. They also agree on the arrangement of the dual variables of sorts σ_1 and σ_2 as $\alpha(\phi)$ enforces it. Let $V_{\sigma_1} = \mathsf{vars}_{\sigma_1}(\alpha(\phi))$ and $V_{\sigma_2} = \mathsf{vars}_{\sigma_2}(\alpha(\phi))$. Because α introduced these variables, and because α enforces the same arrangement on on the dual variables, we have that $[V_{\sigma_1}]^{\mathcal{B}} = [V_{\sigma_2}]^{\mathcal{B}}$.

Now, let $h : V_{\sigma_1}^{\mathcal{B}} \mapsto V_{\sigma_2}^{\mathcal{B}}$ be defined as follows

$$h((v^{\sigma_1})^{\mathcal{B}}) \triangleq (v^{\sigma_2})^{\mathcal{B}} \quad .$$

This function is a bijection and is well-defined since $\mathcal{B}$ satisfies $\alpha(\phi)$. Because $|B_{\sigma_1}| = |B_{\sigma_2}|$, we can extend h to a full bijection $h_{\sigma_1} : B_{\sigma_1} \mapsto B_{\sigma_2}$. Let h_σ be the identity function for $\sigma \neq \sigma_1$.

We use this family of functions to define an interpretation $\mathcal{B}'$ isomorphic to $\mathcal{B}$ as follows. $\mathcal{B}'$ interprets all the domains of the sorts $\sigma \neq \sigma_1$, as $B'_\sigma = B_\sigma$, and the domain of the sort σ_1 as $B'_{\sigma_1} = B_{\sigma_2}$. For each variable v of sort σ, $v^{\mathcal{B}'} \triangleq h_\sigma(v^{\mathcal{B}})$. For each function symbol f, we

define $f^{\mathcal{B}'}(b_1, \ldots, b_n) \triangleq h_{\tau_{n+1}}(f^{\mathcal{B}}(h_{\tau_1}^{-1}(b_1), \ldots, h_{\tau_n}^{-1}(b_n)))$ where τ_i is chosen to match the i^{th} sort in the arity of f. Similarly, we define $p^{\mathcal{B}'}(b_1, \ldots, b_n)$ iff $p^{\mathcal{B}}(h_{\tau_1}^{-1}(b_1), \ldots, h_{\tau_n}^{-1}(b_n))$. It is easy to see that the resulting interpretation $\mathcal{B}'$ is indeed isomorphic to $\mathcal{B}$, and as a result, $\mathcal{B}'$ is also a T-interpretation and satisfies $\alpha(\phi)$.

Finally, let $\mathcal{A}$ be a $T_s^{\sigma_1=\sigma_2}$-interpretation obtained from $\mathcal{B}'$ as in Definition B.5 (i.e. $A_\sigma = B'_\sigma$ for $\sigma \in S' \setminus \{s\}$, $A_s = B'_{\sigma_1} = B'_{\sigma_2}$, and the function and predicate symbols are interpreted the same in $\mathcal{A}$ as in $\mathcal{B}$). It is easy to see that we have $|A_\sigma| = \kappa_\sigma$ for $\sigma \in S'$. It remains to say how variables are interpreted in $\mathcal{A}$. For variables v of sort $\sigma \in S' \setminus \{s\}$, we let $v^{\mathcal{A}} = v^{\mathcal{B}'}$. In addition, for each variable $v \in \mathsf{vars}_s(\phi)$, we let $v^{\mathcal{A}} = (v^{\sigma_1})^{\mathcal{B}'}$. Note that because of the way $\mathcal{B}'$ was constructed, we also have $v^{\mathcal{A}} = (v^{\sigma_2})^{\mathcal{B}'}$. Because $\mathcal{A}$ interprets the variables v of sort s in ϕ the same as both v^{σ_1} and v^{σ_2} in $\mathcal{B}'$, $\mathcal{A}$ interprets everything else exactly the same as in $\mathcal{B}'$, and because $\mathcal{B}'$ satisfies $\alpha(\phi)$, it follows that $\mathcal{A}$ satisfies ϕ. □

Now we can prove the main theorem.

Theorem B.6. *Let Σ be a signature, $\sigma_1, \sigma_2 \in \Sigma^{\mathbb{S}}$, and $s \notin \Sigma^{\mathbb{S}}$. If Σ-theory T is polite with respect to S, where $\sigma_1, \sigma_2 \in S$ and $s \notin S$, then $T_s^{\sigma_1=\sigma_2}$ is polite with respect to $S' = S \setminus \{\sigma_1, \sigma_2\} \cup \{s\}$. Furthermore, if witness is a witness function for theory T, then an acceptable witness function for $T_s^{\sigma_1=\sigma_2}$ is*

$$witness_s^{\sigma_1=\sigma_2}(\phi) = (\gamma_{\mathsf{vars}_s(\phi)} \circ witness \circ \alpha)(\phi) \ .$$

Proof. First we show that $T_s^{\sigma_1=\sigma_2}$ is smooth with respect to S'. Let ϕ be a conjunction of flat $\Sigma_s^{\sigma_1=\sigma_2}$-literals satisfiable in a $T_s^{\sigma_1=\sigma_2}$-structure $\mathcal{A}$. We are given cardinalities $\kappa_\sigma \geq |A_\sigma|$, for $\sigma \in S'$. By Lemma B.6 we know that $\alpha(\phi)$ is satisfiable in a T-interpretation $\mathcal{B}$ such that $|B_\sigma| = |A_\sigma|, \sigma \in S' \setminus \{s\}$, and $|B_{\sigma_1}| = |B_{\sigma_2}| = |A_s|$. By smoothness of T there is a T-interpretation $\mathcal{B}'$ that satisfies $\alpha(\phi)$, such that $|B'_\sigma| = \kappa_\sigma$, for $\sigma \in S' \setminus \{s\}$, and $|B'_{\sigma_1}| = |B'_{\sigma_2}| = \kappa_s$. Then, applying Lemma B.6 one more time (in the other direction), we get that ϕ is satisfiable in a $T_s^{\sigma_1=\sigma_2}$-interpretation $\mathcal{A}'$ such that $|A'_\sigma| = \kappa_\sigma$, for $\sigma \in S'$, which proves smoothness.

Next, we need to show that $T_s^{\sigma_1=\sigma_2}$ is also finitely witnessable. Let ϕ be a conjunction of flat $T_s^{\sigma_1=\sigma_2}$-literals. Because T is finitely witnessable with respect to S, it has a witness function *witness*. We define the witness function of the instantiated theory as

$$witness_s^{\sigma_1=\sigma_2}(\phi) = (\gamma_{\mathsf{vars}_s(\phi)} \circ witness \circ \alpha)(\phi) \ .$$

Among the fresh variables introduced by $witness_s^{\sigma_1=\sigma_2}$ we will distinguish $\overrightarrow{w}$, the fresh variables introduced by *witness*, and $\overrightarrow{v}^{\sigma_1}$ and $\overrightarrow{v}^{\sigma_2}$, the fresh variables introduced by transformation α (i.e. the variables v^{σ_1} and v^{σ_2}, corresponding to variables $v \in \mathsf{vars}_s(\phi)$).

First we need to show that if $\psi = witness_s^{\sigma_1=\sigma_2}(\phi)$, then $\exists \overrightarrow{v}^{\sigma_1} \exists \overrightarrow{v}^{\sigma_2} \exists \overrightarrow{w}.\psi$ and ϕ are $T_s^{\sigma_1=\sigma_2}$-equivalent. This follows from the equivalence of the following statements:

$$\begin{aligned}
&\mathcal{A} \vDash \phi \\
&\alpha(\mathcal{A})\{\overrightarrow{v}^{\sigma_1} \leftarrow \overrightarrow{v}^{\mathcal{A}}, \overrightarrow{v}^{\sigma_2} \leftarrow \overrightarrow{v}^{\mathcal{A}}\} \vDash \alpha(\phi) && \text{(definition of } \alpha\text{)} \\
&\alpha(\mathcal{A})\{\overrightarrow{v}^{\sigma_1} \leftarrow \overrightarrow{v}^{\mathcal{A}}, \overrightarrow{v}^{\sigma_2} \leftarrow \overrightarrow{v}^{\mathcal{A}}\} \vDash \exists \overrightarrow{w}.witness(\alpha(\phi)) && (T \text{ finitely witnessable}) \\
&\alpha(\mathcal{A})\{\overrightarrow{v}^{\sigma_1} \leftarrow \overrightarrow{v}^{\mathcal{A}}, \overrightarrow{v}^{\sigma_2} \leftarrow \overrightarrow{v}^{\mathcal{A}}, \overrightarrow{w} \leftarrow \overrightarrow{a}\} \vDash witness(\alpha(\phi)) \\
&\mathcal{A}\{\overrightarrow{v}^{\sigma_1} \leftarrow \overrightarrow{v}^{\mathcal{A}}, \overrightarrow{v}^{\sigma_2} \leftarrow \overrightarrow{v}^{\mathcal{A}}, \overrightarrow{w} \leftarrow \overrightarrow{a}\} \vDash \gamma_{\mathsf{vars}_s(\phi)}(witness(\alpha(\phi))) && \text{(definition of } \gamma\text{)} \\
&\mathcal{A} \vDash \exists \overrightarrow{v}^{\sigma_1} \exists \overrightarrow{v}^{\sigma_2} \exists \overrightarrow{w}.\psi
\end{aligned}$$

To show that the defined witness function satisfies the second requirement of Definition 3.7, let

$$\mathcal{E} = \{\ E_\sigma \ \mid \ \sigma \in S'\ \}$$

be a family of equivalence relations over a set V of variables with sorts in S', and $\delta_V(\mathcal{E})$ be the arrangement induced by $\mathcal{E}$. Now, assume that there is a $T_s^{\sigma_1=\sigma_2}$-interpretation $\mathcal{A}$ such that

$$\mathcal{A} \vDash \overbrace{(\gamma_{\mathsf{vars}_s(\phi)} \circ witness \circ \alpha)(\phi)}^{\psi} \wedge \delta_V(\mathcal{E})\ .$$

For convenience, we will let $\psi' = (witness \circ \alpha)(\phi)$ (so that $\psi = \gamma_{\mathsf{vars}_s(\phi)}(\psi')$). As in the proof of Theorem B.1 (page 136), we can assume wlog that $\mathsf{vars}_{S'}(\psi) \subseteq V$, i.e. the arrangement $\delta_V(\mathcal{E})$ covers all the variables of ψ with sorts in S' . We will make use of the following sets of variables:

- $V_s = \mathsf{vars}_s(V)$
- $V_\phi = \mathsf{vars}_s(\phi)$
- $V_\alpha =$ superscripted variables in $\alpha(\phi)$ introduced by α (i.e. v^{σ_1} and v^{σ_2} for each $v \in V_\phi$).
- $W_s =$ variables of sort σ_1 or σ_2 in ψ' introduced by the *witness* function for T.

- $\Delta_s = V_s \setminus (V_\phi \cup V_\alpha \cup W_s)$

Note that because γ reinterprets variables of sorts σ_1 and σ_2 as variables of sort s, all of the variables in the above sets are of sort s in the formula $\psi \wedge \delta_V$.

The definition of γ also guarantees that, for all $v \in V_\phi$, the variables v, v^{σ_1}, and v^{σ_2} must be equal in $\mathcal{A}$ and (since $\mathcal{A}$ also satisfies δ_V) belong to the same equivalence class in E_s. We can thus construct a new family of equivalence relations $\mathcal{E}'$ in which the variables from V_α do not appear, while keeping the same number of equivalence classes for each sort. Concretely, let

$$V'_s = V_s \setminus V_\alpha \ ,$$

$$E'_\sigma = \begin{cases} E_\sigma & \text{for } \sigma \neq s \ , \\ E_s \cap V'_s \times V'_s & \text{for } \sigma = s \ , \end{cases}$$

$$\mathcal{E}' = \{ \ E'_\sigma \ \mid \ \sigma \in S' \ \} \ .$$

Also, let $V' = V \setminus V_\alpha$, and for an equivalence relation E, let $Q(E)$ denote the quotient set of E (i.e. the set of all equivalence classes in E). It is clear that $\mathcal{A} \vDash \psi \wedge \delta_{V'}(\mathcal{E}')$ and $|Q(E_\sigma)| = |Q(E'_\sigma)|$ for each $\sigma \in S'$.

In order to switch to reasoning in the signature Σ (as opposed to $\Sigma_s^{\sigma_1=\sigma_2}$), we need to modify the equivalence relations so that variables of different sorts (when considered in the signature Σ) are not in the same equivalence class (so that the induced arrangement is well-sorted). To this end, we define the variable mappings β_{σ_1} and β_{σ_2} as follows. For $v \in V'_s$,

$$\beta_{\sigma_1}(v) = \begin{cases} v^{\sigma_1} \text{ if } v \in V_\phi \ , \\ v \text{ if } v \in W_s \text{ and } v \text{ is of sort } \sigma_1 \ , \\ v' \text{ if } v \in W_s \text{ and } v \text{ is of sort } \sigma_2 \ , \\ v \text{ if } v \in \Delta_s \end{cases} \qquad \beta_{\sigma_2}(v) \quad = \begin{cases} v^{\sigma_2} \text{ if } v \in V_\phi \ , \\ v' \text{ if } v \in W_s \text{ and } v \text{ is of sort } \sigma_1 \ , \\ v \text{ if } v \in W_s \text{ and } v \text{ is of sort } \sigma_2 \ , \\ v' \text{ if } v \in \Delta_s \end{cases}$$

In the above, the primed variables v' are to be understood as fresh variables of the appropriate sort. In addition, we (arbitrarily) choose to interpret variables in Δ_s to be of sort σ_1 when working in Σ. Note that both functions are injective. The purpose of these mappings is to ensure that every variable in V' has some corresponding variable of sorts σ_1 and σ_2 when working in Σ.

We can now construct a new family of equivalence relations as follows. First, let

$$E''_{\sigma_1} = \{\ (\beta_{\sigma_1}(v_1), \beta_{\sigma_1}(v_2))\ \mid\ (v_1, v_2) \in E'_s\ \}\ ,$$
$$E''_{\sigma_2} = \{\ (\beta_{\sigma_2}(v_1), \beta_{\sigma_2}(v_2))\ \mid\ (v_1, v_2) \in E'_s\ \}\ ,$$

For the other sorts $\sigma \in S \setminus \{\sigma_1, \sigma_2\}$, we simply let $E''_\sigma = E'_\sigma = E_\sigma$. We then set $\mathcal{E}'' = \{\ E''_\sigma\ \mid\ \sigma \in S\ \}$. Let V'' be the set of variables appearing in $\mathcal{E}''$. Note that as desired, variables in the same equivalence class have the same sort. In addition, with the exception of the variables in V_ϕ, all variables appearing in $\mathcal{E}'$ also appear in $\mathcal{E}''$. The fresh primed variables are used just as temporary place-holders of the appropriate sort. It is easy to see that the number of equivalence classes is preserved, i.e. $|Q(E''_{\sigma_1})| = |Q(E''_{\sigma_2})| = |Q(E'_s)| = |Q(E_s)|$.

Now, it is not hard to construct a T-interpretation $\mathcal{B}$ starting from $\mathcal{A}$ such that

$$\mathcal{B} \vDash \psi' \wedge \delta_{V''}(\mathcal{E}'')\ .$$

We do this as follows. The domains of the Σ-structure $\mathcal{B}$ will mimic those in $\mathcal{A}$, except that $B_{\sigma_1} = B_{\sigma_2} = A_s$. We also keep the same interpretations of all function and predicate symbols, while moving back to the original signature, and hence use the new domains where necessary. It follows that $\mathcal{B}$ is a T-structure. We interpret the variables of sorts in $S \setminus \{\sigma_1, \sigma_2\}$ as they were, and the variables of sorts σ_1 and σ_2 as $(\beta_{\sigma_1}(v))^{\mathcal{B}} = (\beta_{\sigma_2}(v))^{\mathcal{B}} = (v)^{\mathcal{A}}$. By the definition of γ and due to the way we constructed $\mathcal{E}''$, it is clear that $\psi' \wedge \delta_{V''}(\mathcal{E}'')$ is indeed satisfied by $\mathcal{B}$.

Now we can apply the finite witnessability of T to obtain a T-interpretation $\mathcal{C}$ satisfying $\psi' \wedge \delta_{V''}(\mathcal{E}'')$ such that for all $\sigma \in S$ we have

$$C_\sigma = [\mathsf{vars}_\sigma(\psi' \wedge \delta_{V''}(\mathcal{E}''))]^{\mathcal{C}}\ .$$

Since all of the variables in ψ' are also in V'', we have that

$$|C_{\sigma_1}| = |C_{\sigma_2}| = |Q(E''_{\sigma_1})| = |Q(E''_{\sigma_2})| = |Q(E'_s)| = |Q(E_s)|\ .$$

Similarly, for $\sigma \in S \setminus \{\sigma_1, \sigma_2\}$, $|C_\sigma| = |Q(E_\sigma)|$. Now, define $g_{\sigma_1} : Q(E'_s) \mapsto C_{\sigma_1}$ as $g_{\sigma_1}([v]) = (\beta_{\sigma_1}(v))^{\mathcal{C}}$. This is well-defined since $\mathcal{C}$ satisfies $\delta_{V''}(\mathcal{E}'')$. For the same reason, g_{σ_1} is injective. Finally, it must be surjective because $|Q(E'_s)| = |C_{\sigma_1}|$. Define the bijection g_{σ_2} similarly.

Now, let $h : C_{\sigma_1} \mapsto C_{\sigma_2} = g_{\sigma_2} \circ g_{\sigma_1}^{-1}$. Clearly, h is a bijection. As in the proof of Lemma B.6, we can extend h to a family of bijections that forms an isomorphism into a T-interpretation $\mathcal{D}$ such that:

- for $\sigma \in S \setminus \{\sigma_1, \sigma_2\}$, $D_\sigma = C_\sigma$,
- $D_{\sigma_1} = D_{\sigma_2} = C_{\sigma_2}$,
- for $v \in \mathsf{vars}_{\sigma_1}(V'')$, $v^{\mathcal{D}} = h(v^{\mathcal{C}})$,
- for $v \in V'' \setminus \mathsf{vars}_{\sigma_1}(V'')$, $v^{\mathcal{D}} = v^{\mathcal{C}}$,
- $\mathcal{D} \vDash \psi' \wedge \delta_{V''}(\mathcal{E}'')$.

Note that for $v \in V'_s$, we have:

$$(\beta_{\sigma_1}(v))^{\mathcal{D}} = h((\beta_{\sigma_1}(v))^{\mathcal{C}}) = (\beta_{\sigma_2}(v))^{\mathcal{C}} = (\beta_{\sigma_2}(v))^{\mathcal{D}} \tag{B.20}$$

Finally, we construct a $T_s^{\sigma_1=\sigma_2}$-structure $\mathcal{F}$ from $\mathcal{D}$ by using the construction of Definition B.5. We interpret in $\mathcal{F}$ only the variables $v \in V$ as follows:

$$v^{\mathcal{F}} = \begin{cases} (v^{\sigma_2})^{\mathcal{D}} \text{ if } v \in V_\phi \ , \\ v^{\mathcal{D}} \text{ otherwise } . \end{cases}$$

We also claim that for $v \in V'_s$,

$$v^{\mathcal{F}} = (\beta_{\sigma_2}(v))^{\mathcal{D}} . \tag{B.21}$$

This follows easily from the definition for $v \in V_\phi$. Otherwise, we have $v^{\mathcal{F}} = v^{\mathcal{D}}$ with $v \in W_s \cup \Delta_s$. But notice that in this case we know that either β_{σ_1} or β_{σ_2} is the identity function. The claim (B.21) then follows from (B.20).

Since interpretations in $\mathcal{F}$ follow those in $\mathcal{D}$ except for variables in V_ϕ, and since no variables from V_ϕ appear in $\psi' \wedge \delta_{V''}(\mathcal{E}'')$, it should be clear that $\mathcal{F} \vDash \gamma_\varnothing(\psi' \wedge \delta_{V''}(\mathcal{E}''))$. Furthermore, for $v \in V_\phi$, we have

$$\begin{aligned} v^{\mathcal{F}} &= (v^{\sigma_2})^{\mathcal{D}} \ , \\ (v^{\sigma_2})^{\mathcal{F}} &= (v^{\sigma_2})^{\mathcal{D}} \ , \\ (v^{\sigma_1})^{\mathcal{F}} &= (v^{\sigma_1})^{\mathcal{D}} = (\beta_{\sigma_1}(v))^{\mathcal{D}} = (\beta_{\sigma_2}(v))^{\mathcal{D}} = (v^{\sigma_2})^{\mathcal{D}}, \end{aligned}$$

and thus

$$\mathcal{F} \vDash v^{\sigma_1} = v \wedge v^{\sigma_2} = v \ . \tag{B.22}$$

It follows that $\mathcal{F} \vDash \psi$. It remains to show that $\mathcal{F} \vDash \delta_V(\mathcal{E})$. Recall that for each $v \in V_\phi$, we know that v, v^{σ_1}, and v^{σ_2} must all be in the same equivalence class in $\mathcal{E}$. It follows that if $\mathcal{F} \vDash \delta_{V'}(\mathcal{E}')$, then by (B.22), we will have $\mathcal{F} \vDash \delta_V(\mathcal{E})$.

To show $\mathcal{F} \vDash \delta_{V'}(\mathcal{E}')$, consider a pair of variables $v_1, v_2 \in V'$. Suppose the sorts of v_1 and v_2 are $\sigma \neq s$. We know that $E'_\sigma = E''_\sigma$, so $(v_1, v_2) \in \mathcal{E}'$ iff $(v_1, v_2) \in \mathcal{E}''$ iff $\mathcal{F} \vDash v_1 = v_2$ (since $\mathcal{F} \vDash \gamma_\varnothing(\delta_{V''}(\mathcal{E}''))$ and γ has no effect in this case).

Finally, suppose that v_1, v_2 have sort s, so that $v_1, v_2 \in V'_s$. We have

$$\begin{array}{lll} (v_1, v_2) \in \mathcal{E}' & \text{iff} \quad (\beta_{\sigma_2}(v_1), \beta_{\sigma_2}(v_2)) \in \mathcal{E}'' & \text{by def of } \mathcal{E}'' \text{ and since } \beta_{\sigma_2} \text{ is injective} \\ & \text{iff} \quad ((\beta_{\sigma_2}(v_1))^{\mathcal{D}} = (\beta_{\sigma_2}(v_2))^{\mathcal{D}} & \text{since } \mathcal{D} \vDash \delta_{V''}(\mathcal{E}'') \\ & \text{iff} \quad v_1^{\mathcal{F}} = v_2^{\mathcal{F}} & \text{by (B.21)} \end{array}$$

Thus, $\mathcal{F} \vDash \psi \wedge \delta_V(\mathcal{E})$.

The last step is to show that $F_\sigma = [\mathsf{vars}_\sigma(\psi \wedge \delta_V(\mathcal{E}))]^{\mathcal{F}}$, for $\sigma \in S'$. Since $\mathsf{vars}_{S'}(\psi) \subseteq V$, it suffices to show that $F_\sigma = [\mathsf{vars}_\sigma(\delta_V(\mathcal{E}))]^{\mathcal{F}}$. For this to hold, it suffices to know that $|F_\sigma| = |Q(E_\sigma)|$. For $\sigma \neq s$, we have that $|F_\sigma| = |D_\sigma| = |C_\sigma| = |Q(E_\sigma)|$. Similarly, we have $|F_s| = |D_{\sigma_2}| = |C_{\sigma_2}| = |Q(E_s)|$. This concludes the proof. □

Example B.3. Consider again example B.2, i.e. we have a theory of arrays T_{array} that operates over the sorts

$$\Sigma^{\mathbb{S}} = \{\mathsf{array}_1, \ldots, \mathsf{array}_n, \mathsf{index}_1, \ldots \mathsf{index}_n, \mathsf{elem}_1\}$$

and is polite with respect to the index and element sorts

$$\Sigma^{\mathbb{S}} = \{\mathsf{index}_1, \ldots \mathsf{index}_n, \mathsf{elem}_1\} \ .$$

Using Theorem B.6, we can now safely replace the sorts index_1, index_2 and elem_1 with the sort of bit-vectors bv, obtaining a theory $T_{\mathsf{array(bv)}}$ of n-dimensional arrays where the elements and the indices are of the same bit-vector sort. This theory $T_{\mathsf{array(bv)}}$ of arrays over bit-vectors

is polite with respect to the sort bv, and therefore we can safely combine it with the theory of bit-vectors T_{bv}.

Using the combination method for polite theories, we can therefore get a *sound and complete decision procedure for deciding the the theory of n-dimensional arrays over bit-vectors*, given a decision procedure and witness function for the theory of arrays T_{array} and a decision procedure for the theory of bit-vectors T_{bv}.

Theorem B.3 together with Theorem B.6 give a practical modular approach for reasoning about and deciding combinations of polite theories.

Bibliography

[1] Tobias Achterberg. *SCIP: Solving constraint integer programs.* PhD thesis, TU Berlin, 2007.

[2] Tobias Achterberg, Thorsten Koch, and Alexander Martin. MIPLIB 2003. *Operations Research Letters*, 34(4):361–372, 2006.

[3] Behzad Akbarpour and Lawrence C. Paulson. MetiTarski: An automatic theorem prover for real-valued special functions. *Journal of Automated Reasoning*, 44(3):175–205, 2010.

[4] Clark W. Barrett, David L. Dill, and Jeremy Levitt. Validity checking for combinations of theories with equality. In *Formal Methods In Computer-Aided Design*, pages 187–201. Springer, 1996.

[5] Clark W. Barrett, Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. Splitting on demand in SAT Modulo Theories. In *Logic for Programming, Artificial Intelligence, and Reasoning*, volume 4246 of *LNCS*, pages 512–526. Springer, 2006.

[6] Clark W. Barrett, Roberto Sebastiani, Sanjit A. Seshia, and Cesare Tinelli. Satisfiability modulo theories. In *Handbook of Satisfiability*. IOS Press, 2009.

[7] Clark W. Barrett and Cesare Tinelli. CVC3. In *Computer Aided Verification*, pages 298–302. Springer, 2007.

[8] Peter Barth. A Davis-Putnam based enumeration algorithm for linear pseudo-Boolean optimization. Technical Report MPI-I-95-2-003, Max-Planck-Institut für Informatik, 1995.

[9] Saugata Basu, Richard Pollack, and Marie-Françoise Roy. *Algorithms in real algebraic geometry.* Springer, 2006.

[10] Sergey Berezin, Vijay Ganesh, and David L. Dill. An online proof-producing decision procedure for mixed-integer linear arithmetic. In Hubert Garavel and John Mark Hatcliff, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 2619 of *Lecture Notes in Computer Science*, pages 521–536. Springer, 2003.

[11] Daniel Le Berre and Anne Parrain. The Sat4j library, release 2.2 system description. *Journal on Satisfiability, Boolean Modeling and Computation*, 7:59–64, 2010.

[12] Marco Bozzano, Roberto Bruttomesso, Alessandro Cimatti, Tommi Junttila, Silvio Ranise, Peter van Rossumd, and Roberto Sebastiani. Efficient theory combination via Boolean search. *Information and Computation*, 204(10):1493–1525, 2006.

[13] Aaron R. Bradley and Zohar Manna. *The calculus of computation: decision procedures with applications to verification*, volume 374. Springer, 2007.

[14] Christopher W. Brown. *Solution formula construction for truth invariant CAD's*. PhD thesis, University of Delaware, 1999.

[15] Christopher W. Brown. Improved projection for cylindrical algebraic decomposition. *Journal of Symbolic Computation*, 32(5):447–465, 2001.

[16] Christopher W. Brown. QEPCAD B: a program for computing with semi-algebraic sets using CADs. *ACM SIGSAM Bulletin*, 37(4):97–108, 2003.

[17] William S. Brown and Joseph F Traub. On Euclid's algorithm and the theory of subresultants. *Journal of the ACM*, 18(4):505–514, 1971.

[18] Robert Brummayer and Armin Biere. Boolector: An efficient SMT solver for bit-vectors and arrays. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 5505 of *LNCS*, pages 174–177. Springer, 2009.

[19] Roberto Bruttomesso, Alessandro Cimatti, Anders Franzén, Alberto Griggio, and Roberto Sebastiani. The MathSAT 4 SMT solver. In *Computer Aided Verification*, volume 5123 of *LNCS*, pages 299–303. Springer, 2008.

[20] Roberto Bruttomesso, Alessandro Cimatti, Anders Franzen, Alberto Griggio, and Roberto Sebastiani. Delayed theory combination vs. Nelson-Oppen for satisfiability modulo theories: A comparative analysis. *Annals of Mathematics and Artificial Intelligence*, 55(1):63–99, 2009.

[21] Bruno Buchberger, George Edwin Collins, Rüdiger Loos, and Rudolf Albrecht, editors. *Computer algebra*. Symbolic and algebraic computation. Springer, 1982.

[22] Bob F. Caviness and Jeremy R. Johnson, editors. *Quantifier Elimination and Cylindrical Algebraic Decomposition*. Texts and Monographs in Symbolic Computation. Springer, 2004.

[23] Donald Chai and Andreas Kuehlmann. A fast pseudo-Boolean constraint solver. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24(3):305–317, 2005.

[24] Vašek Chvátal. Edmonds polytopes and a hierarchy of combinatorial problems. *Discrete Mathematics*, 4(4):305–337, 1973.

[25] Henri Cohen. *A Course in Computational Algebraic Number Theory*. Springer Verlag, 1993.

[26] George Edwin Collins. Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In *Automata Theory and Formal Languages 2nd GI Conference Kaiserslautern, May 20–23, 1975*, pages 134–183. Springer, 1975.

[27] George Edwin Collins and Hoon Hong. Partial cylindrical algebraic decomposition for quantifier elimination. *Journal of Symbolic Computation*, 12(3):299–328, 1991.

[28] Stephen Arthur Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158. ACM, 1971.

[29] David Charles Cooper. Theorem proving in arithmetic without multiplication. *Machine Intelligence*, 7(91-99):300, 1972.

[30] Scott Cotton. Natural domain SMT: A preliminary assessment. In Krishnendu Chatterjee and Thomas A. Henzinger, editors, *Formal Modeling and Analysis of Timed Systems*, volume 6246 of *Lecture Notes in Computer Science*, pages 77–91, 2010.

[31] Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397, 1962.

[32] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7(3):201–215, 1960.

[33] Leonardo de Moura and Nikolaj Bjørner. Relevancy propagation. Technical Report MSR-TR-2007-140, Microsoft Research, 2007.

[34] Leonardo de Moura and Nikolaj Bjørner. Model-based Theory Combination. In *5th International Workshop on Satisfiability Modulo Theories*, volume 198 of *Electronic Notes in Theoretical Computer Science*, pages 37–49. Elsevier, 2008.

[35] Leonardo de Moura and Nikolaj Bjørner. Z3: An efficient SMT solver. In Coimbatore Rajamani Ramakrishnan and Jakob Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340. Springer, 2008.

[36] Leonardo de Moura and Nikolaj Bjørner. Generalized, efficient array decision procedures. In *Formal Methods in Computer-Aided Design, 2009*, pages 45–52. IEEE, November 2009.

[37] René Descartes. *La Géométrie*. 1637.

[38] David Detlefs, Greg Nelson, and James B. Saxe. Simplify: a theorem prover for program checking. *Journal of the ACM*, 52(3):365–473, 2005.

[39] Isili Dillig, Thomas Dillig, and Alex Aiken. Cuts from proofs: A complete and practical technique for solving linear inequalities over integers. In Ahmed Bouajjani and Oded Maler, editors, *Computer Aided Verification*, volume 5643 of *Lecture Notes in Computer Science*, pages 233–247. Springer, 2009.

[40] Andreas Dolzmann and Thomas Sturm. Redlog: Computer algebra meets computer logic. *ACM SIGSAM Bulletin*, 31(2):2–9, 1997.

[41] Bruno Dutertre and Leonardo de Moura. A fast linear-arithmetic solver for DPLL(T). In Thomas Ball and Robert B. Jones, editors, *Computer Aided Verification*, volume 4144 of *Lecture Notes in Computer Science*, pages 81–94. Springer, 2006.

[42] Herbert B. Enderton. *A mathematical introduction to logic*. Academic press New York, 1972.

[43] Jean-Christophe Filliâtre, Sam Owre, Harald Ruess, and Natarajan Shankar. ICS: Integrated Canonizer and Solver. In *Computer Aided Verification*, pages 246–249. Springer, 2001.

[44] Martin Fränzle, Christian Herde, Tino Teige, Stefan Ratschan, and Tobias Schubert. Efficient solving of large non-linear arithmetic constraint systems with complex Boolean structure. *Journal on Satisfiability, Boolean Modeling and Computation*, 1(3-4):209–236, 2007.

[45] Carsten Fuhs, Jürgen Giesl, Aart Middeldorp, Peter Schneider-Kamp, René Thiemann, and Harald Zankl. Sat solving for termination analysis with polynomial interpretations. *Theory and Applications of Satisfiability Testing*, pages 340–354, 2007.

[46] Évariste Galois. Sur la théorie des nombres. *Bulletin des sciences mathématiques, physiques et chimiques*, 13:428–435, 1830.

[47] Carl Friedrich Gauss. *Werke*, volume 2. Königlichen Gesellschaft der Wissenschaften, 1876.

[48] Keith O. Geddes, Stephen R. Czapor, and George Labahn. *Algorithms for computer algebra*. Springer, 1992.

[49] Paul C. Gilmore. A proof method for quantification theory: its justification and realization. *IBM Journal of research and development*, 4(1):28–35, 1960.

[50] Ralph Edward Gomory. Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical Society*, 64(5):275–278, 1958.

[51] Alberto Griggio. A practical approach to satisfiability modulo linear integer arithmetic. *Journal on Satisfiability, Boolean Modeling and Computation*, 8:1–27, 2012.

[52] Hoon Hong. An improvement of the projection operator in cylindrical algebraic decomposition. In *Proceedings of the international symposium on Symbolic and algebraic computation*, pages 261–264. ACM, 1990.

[53] Hoon Hong. Comparison of several decision algorithms for the existential theory of the reals. 1991.

[54] Muḥammad ibn Mūsā al Khwārizmī. *Hisab al-jabr w'al-muqabala*. 820.

[55] Qin Jiushao. *Mathemathical Treatise in Nine Chapters*. 1247.

[56] Dejan Jovanović and Clark W. Barrett. Polite theories revisited. Technical Report TR2010-922, Department of Computer Science, New York University, January 2010.

[57] Dejan Jovanović and Clark W. Barrett. Polite theories revisited. In *Logic for Programming, Artificial Intelligence, and Reasoning*, volume 6397 of *LNCS*, pages 402–416. Springer Berlin / Heidelberg, 2010.

[58] Dejan Jovanović and Leonardo de Moura. Cutting to the chase: Solving linear integer arithmetic. In *Proceedings of the 23rd International Conference on Automated Deduction*, pages 338–353. Springer, 2011.

[59] Donald Ervin Knuth. *The Art of Computer Programming*, volume 2: Seminumerical Algorithms. Addison Wesley, third edition, 1997.

[60] Konstantin Korovin, Nestan Tsiskaridze, and Andrei Voronkov. Conflict resolution. In Ian P. Gent, editor, *Principles and Practice of Constraint Programming*, volume 509 of *Lecture Notes in Computer Science*, pages 509–523. Springer, 2009.

[61] Daniel Kroening and Ofer Strichman. *Decision procedures: an algorithmic point of view*. Springer-Verlag New York Inc, 2008.

[62] Sava Krstić and Amit Goel. Architecting solvers for SAT modulo theories: Nelson-Oppen with DPLL. In Boris Konev and Frank Wolter, editors, *Frontiers of Combining Systems*, volume 4720 of *Lecture Notes in Computer Science*, pages 1–27. Springer, 2007.

[63] Sava Krstić, Amit Goel, Jim Grundy, and Cesare Tinelli. Combined Satisfiability Modulo Parametric Theories. In Orna Grumberg and Michael Huth, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 13th International Conference, TACAS 2007*, volume 4424 of *Lecture Notes in Computer Science*, pages 602–617. Springer, 2007.

[64] Rüdiger Loos. Generalized polynomial remainder sequences. *Computer Algebra: Symbolic and Algebraic Computation*, pages 115–137, 1982.

[65] Scott McCallum. *An improved projection operation for cylindrical algebraic decomposition.* PhD thesis, University of Wisconsin-Madison, 1984.

[66] Sean McLaughlin and John Robert Harrison. A proof-producing decision procedure for real arithmetic. In *Proceedings of the 20th International Conference on Automated Deduction*, volume 3632, page 295. Springer, 2005.

[67] Kenneth L. McMillan, Andreas Kuehlmann, and Mooly Sagiv. Generalizing DPLL to richer logics. In Ahmed Bouajjani and Oded Maler, editors, *Computer Aided Verification*, volume 5643 of *Lecture Notes in Computer Science*, pages 462–476. Springer, 2009.

[68] Bhubaneswar Mishra. *Algorithmic algebra.* Springer, 1993.

[69] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: engineering an efficient SAT solver. In *Proceedings of the 38th annual Design Automation Conference*, pages 530–535. ACM, 2001.

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

[71] Isaac Newton. *Arithmetica universalis: sive de compositione et resolutione arithmetica liber.* 1732.

[72] Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. Solving SAT and SAT modulo theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(T). *Journal of the ACM*, 53(6):937–977, 2006.

[73] Derek C. Oppen. Complexity, convexity and combinations of theories. *Theoretical Computer Science*, 12(3):291–302, 1980.

[74] Christos H. Papadimitriou. On the complexity of integer programming. *Journal of the ACM*, 28(4):765–768, 1981.

[75] Grant O. Passmore. *Combined Decision Procedures for Nonlinear Arithmetics, Real and Complex*. PhD thesis, University of Edinburgh, 2011.

[76] André Platzer, Jan-David Quesel, and Philipp Rümmer. Real world verification. In *Proceedings of the 22nd International Conference on Automated Deduction*, pages 485–501. Springer, 2009.

[77] William Pugh. The Omega test: a fast and practical integer programming algorithm for dependence analysis. In *Proceedings of the 1991 ACM/IEEE conference on Supercomputing*, pages 4–13. ACM, 1991.

[78] Zvonimir Rakamarić and Alan J. Hu. A Scalable Memory Model for Low-Level Code. In *Verification, Model Checking, and Abstract Interpretation*, volume 5403 of *LNCS*, page 304. Springer-Verlag, 2009.

[79] Silvio Ranise, Christophe Ringeissen, and Calogero G. Zarba. Combining Data Structures with Nonstably Infinite Theories Using Many-Sorted Logic. In *Frontiers of Combining Systems*, volume 3717 of *LNCS*, pages 48–64. Springer, 2005.

[80] Silvio Ranise, Christophe Ringeissen, and Calogero G. Zarba. Combining Data Structures with Nonstably Infinite Theories using Many-Sorted Logic. Research Report RR-5678, INRIA, 2005.

[81] John Alan Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–41, 1965.

[82] Sanjit A. Seshia and Randal E. Bryant. Deciding quantifier-free Presburger formulas using parameterized solution bounds. In *Logic in Computer Science*, pages 100–109. IEEE, 2004.

[83] Robert E. Shostak. An algorithm for reasoning about equality. In *5th international joint conference on Artificial intelligence*, pages 526–527. Morgan Kaufmann Publishers Inc., 1977.

[84] Robert E. Shostak. Deciding combinations of theories. In *6th Conference on Automated Deduction*, pages 209–222. Springer, 1982.

[85] João P. Marques Silva and Karem A. Sakallah. GRASP – a new search algorithm for satisfiability. In *International Conference on Computer-Aided Design*, pages 220–227. ACM and IEEE Computer Society, 1996.

[86] João P. Marques Silva and Karem A. Sakallah. GRASP: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48(5):506–521, 1999.

[87] Adam W. Strzeboński. Computing in the field of complex algebraic numbers. *Journal of Symbolic Computation*, 24(6):647–656, 1997.

[88] Adam W. Strzeboński. Cylindrical algebraic decomposition using validated numerics. *Journal of Symbolic Computation*, 41(9):1021–1038, 2006.

[89] Jacques Charles François Sturm. *Mémoire sur la résolution des équations numériques.* 1835.

[90] Alfred Tarski. Über definierbare Mengen reeller Zahlen. *Annales de la Société Polonaise de Mathématique*, 9, 1930.

[91] Alfred Tarski. A decision method for elementary algebra and geometry. Technical Report R-109, Rand Corporation, 1951.

[92] Cesare Tinelli and Mehdi T. Harandi. A new correctness proof of the Nelson–Oppen combination procedure. In *Frontiers of Combining Systems*, Applied Logic, pages 103–120. Kluwer Academic Publishers, March 1996.

[93] Cesare Tinelli and Calogero G. Zarba. Combining decision procedures for sorted theories. In *Logic in Artificial Intelligence*, volume 3229 of *LNAI*, pages 641–653. Springer, 2004.

[94] Cesare Tinelli and Calogero G. Zarba. Combining decision procedures for theories in sorted logics. Technical Report 04-01, Department of Computer Science, The University of Iowa, February 2004.

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

[96] Ashish Tiwari. An algebraic approach for the unsatisfiability of nonlinear constraints. In *Computer Science Logic*, pages 248–262. Springer, 2005.

[97] Lou van den Dries. Alfred Tarski's elimination theory for real closed fields. *The Journal of Symbolic Logic*, 53(1):7–19, 1988.

[98] Alexandre Joseph Hidulphe Vincent. Note sur la résolution des équations numériques. *Journal de Mathématiques Pures et Appliquées*, 1:341–372, 1836.

[99] Volker Weispfenning. Quantifier elimination for real algebra - the quadratic case and beyond. *AAECC*, 8:85–101, 1993.

[100] Laurence A. Wolsey and George L. Nemhauser. *Integer and combinatorial optimization*. Wiley New York, 1999.

[101] Wen-Tsun Wu. *Mathematics mechanization: Mechanical Geometry Theorem-Proving, Mechanical Geometry Problem-Solving, and Polynomial Equations-Solving*. Kluwer Academic Publishers, 2001.

[102] Harald Zankl and Aart Middeldorp. Satisfiability of non-linear (ir)rational arithmetic. In *Logic for Programming, Artificial Intelligence, and Reasoning*, pages 481–500. Springer, 2010.