NELSON OPPEN COMBINATION AS A REWRITE THEORY

BY

NISHANT RODRIGUES

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Mathematics
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2018

Urbana, Illinois

Advisor:

Professor José Meseguer

Abstract

Solving Satisfiability Modulo Theories (SMT) problems in a key piece in automating tedious mathematical proofs. It involves deciding satisfiability of formulas of a decidable theory, which can often be reduced to solving systems of equalities and disequalities, in a variety of theories such as linear and non-linear real and integer arithmetic, arrays, uninterpreted and Boolean algebra. While solvers exist for many such theories or their subsets, it is common for interesting SMT problems to span multiple theories. SMT solvers typically use refinements of the Nelson-Oppen combination method, an algorithm for producing a solver for the quantifier free fragment of the combination of a number of such theories via cooperation between solvers of those theories, for this case. Here, we present the Nelson-Oppen algorithm adapted for an order-sorted setting as a rewriting logic theory. We implement this algorithm in the Maude System and instantiate it with the theories of real and integer matrices to demonstrate its use in automated theorem proving, and with hereditarily finite sets with reals to show its use with non-convex theories. This is done using both SMT solvers written in Maude itself via reflection (Variant-based satisfiability) and using external solvers (CVC4 and Yices). This work can be considered a first step towards building a rich ecosystem of cooperating SMT solvers in Maude, that modeling and automated theorem proving tools typically written using the Maude System can leverage.

Table of Contents

Chapter 1

Introduction

In 1928, David Hilbert posed the "Entscheidungsproblem" ("the decision problem") to the mathematical community: a challenge to mechanize mathematics; to find an algorithm that takes as input any first-order logic statement and return whether it is a true statement or not. Even though, in 1936, Alan Turing and Alonzo Church independently showed that such an algorithm is impossible, great progress has been made towards solving significant and profitable subsets of first-order logic formulae.

Given a theory and a first order logic formula in it's signature, the Satisfiability Modulo Theories problem is that of deciding whether there is an assigment of variables such that the interpretation of that forumla holds in some model of that theory. In this case, we say that the forumla is "satisfiable". Otherwise we say that the formula is "unsatisfiable". Validity, an important related concept, is the dual of satisfiability. A formula is "valid" in a theory, if in every model of the theory and for every possible assigment of variables, the formula holds. For example, the statement "every natural number factorizers uniquely into a set of prime numbers" is valid in Peano arithmetic, whereas any first order logic formulation of "Peano arithmetic is consistent" in the theory of Peano arithmetic is not, due to Gödel's Incompleteness Theorems. An algorithmic check for validity of arbitary formulae was Hilbert's dream and forms the core of SMT solving.

Although, alas, Hilbert's dream is impossible, and in general satisfiability is undecidable (e.g. for non-linear integer arithmetic), there are subsets of theories that are decidable and immensely useful for a variety of applications including solving optimization problems, program verification and automated theorem proving.

Over the years, efficient algorithms were devised for linear real and integer arithmetic, non-linear arithmetic, arrays (partial functions from the naturals) amongst others, as well as theory-generic algorithms (Meseguer 2016). Program verification and other applications, however, often involve working with a combination of two or more theories (e.g. verification of a sorting algorithm may involve using the combined theory of arrays and of total linear orders). Initially, solving satisfiability problems in a combination of theories involved manually working out the combined procedure and proving their correctness (Shostak 1979)(Suzuki and Jefferson 1980). In 1979, Greg Nelson and Derek Oppen published a generic method for composing SMT solvers for two theories into one for the quantifier free fragment of their union (Nelson and Oppen 1979). In (Shostak 1984), Shostak introduced a procedure for deciding combinations of "canonizable" and "solvable" theories, called Shostak theories. Today, most SMT solvers use the Nelson-Oppen algorithm, with refinements for handling Shostak, shiny, and polite theories, at their core.

In this thesis, we implement in rewriting logic an order-sorted Nelson-Oppen algorithm for composing satisfiability modulo theory (SMT) solvers for first order theories into an SMT solver for the quantifier-free fragment of their union. We build on the Tinelli and Zarba's work of extending the Nelson-Oppen algorithm to order-sorted logics (Tinelli and Zarba 2004) and refer to the notes of Meseguer (J. Meseguer 2017b), to implement this algorithm as an order-sorted rewrite theory using the Maude System.

Implementing this as a rewrite theory is particularly attractive for several reasons. Firstly, the inference rules translate almost directly into axioms of an equational theory (used as rewriting rules), making the algorithm much clearer than it would be in, e.g. C++. Secondly, many first order logic theories can be defined as equational theories with an initial algebra semantics (a subset of rewrite theories). This, in combination with rewriting logic being a reflective logic allows implementing theory generic SMT solvers such as Variant-Based Satisfiability (Meseguer 2016) and congruence closure possible. In particular, these solvers have been implemented in Maude (Skeirik and Meseguer 2016) taking advantage of reflection through Maude's `META-LEVEL`.

The Maude System is a programming language often used for modeling and verification of systems. It has been used to verify a wide spectrum of systems, from biological systems (Pathway Logic (Eker et al. 2004)), to Cryptographic Protocols (Maude NPA (Escobar, Meadows, and Meseguer 2006)), to concensus algorithms, to programming languages (KFramework (Şerbanuţă and Roşu 2010)), and so on (see (Meseguer 2012) for a comprehensive survey of such applications). The capabilities of many of these formal verification tools can be substantially increased through leveraging the power of SMT solvers. Besides the SMT solvers mentioned previously, Maude also offers access to CVC4 (Barrett et al. 2011) as well as Yices (Dutertre 2014), both state of the art solvers. While both CVC4 and Yices themselves implement the Nelson-Oppen algorithm internally, those implementations do not allow cooperation between the algorithms implemented in Maude as rewrite theories, or other solvers. Thus this implementation of the algorithm can be seen as a first step towards a rich, robust and extensable ecosystem of cooperating SMT solvers.

1.1 Satisfiability Modulo Theories (SMT)

SMT problems are decision problems for checking whether a first-order logic formula $\varphi(\vec{x})$ is satisfiable in a theory T, i.e. whether there is a model M of T such that $M \models \exists \vec{x} \varphi(\vec{x})$. Similarly, a formula is said to be valid if its negation is unsatisfiable.

Checking satisfiabilty and its dual, validity, have a wide range of applications, including logistics, optimization, software verification, program synthesis and automated theorem proving. In fact validity forms the core of automated theorem proving. Its importance has led to the standardization of a language, SMT-LIB for describing SMT problems, and the SMT-COMP competition, where the foremost solvers compete against each other for effectiveness and performance. This has created a virtuous cycle where difficult real world SMT problems posed by industry and academia are added to the benchmarks and the solvers compete at efficiently solving these problems, enabling further and more interesting applications.

SMT has come a long way since Hilbert posed his problem of "mechanising mathematics". In 1929, Persburger proved that linear integer arithmetic is indeed decidable, and although it was shown later by Fischer and Rabin that the algorithm must be worst case doubly exponential on the length of formulae, the Simplex Algorithm and its variations has proven to be an effective method of solving SMT for both real and integer quantifier free linear arithmetic efficiently. Efficient algorithms have also been found for a number of other theories, such as the theory of arrays, uninterpreted functions and more.

SMT problems in automated theorem proving and program verification commonly involve combinations of standard theories. For example, verifying a sorting algorithm may involve solving queries in the combined theories of lists and of total orders. Prior to 1979, this involved manually looking for a combined algorithm, and proving that it worked as promised. In 1979, Nelson and Oppen proposed a general algorithm for combining SMT solvers into one for the quantifier free fragment of the larger theory. Although this algorithm only applies to a class of theories called stably infinite (which intuitively means that models of both theories can be found having the same cardinality), this requirement is much easier to meet in a many-sorted or order-sorted context, and it has furthermore been relaxed in subsequent work (e.g., to so-called "polite" theories). As a consequence many important theories fall into this class or satisfy weaker requirements allowing them to be combined.

1.2 Logical foundations of Maude

The Maude System is a programming language and framework whose semantics is based on Rewriting Logic. Rewrite theories model concurrent systems. In particular, for model checking purposes they provide a very high level formalism for axiomatizing possibly infinite Kripke structures. This is exploited in Maude for formal analysis purposes, since concurrent systems specified as rewrite theories can be analyzed using Maude's LTL model checker and other model checkers and theorem proving tools in Maude's formal environment.

Since a rewrite theory is a triple (Σ, E, R) with (Σ, E) an equational theory with symbols Σ and (possibly conditional) equations E, and R the theory's rewrite rules axiomatizing system transitions, a rewrite theory defines over the elements of the initial algebra $T_{\Sigma/E}$ (which models the system states), a transition system. This transition system is intrinsically concurrent thanks to the logic's semantics, and captures naturally the non-determinism present in such systems.

1.2.1 Unsorted vs Many-Sorted vs Order-Sorted Logics

Traditionally, first order logic has been used in an unsorted setting, i.e. there is a single set of elements in the model that can be quantified over. This can however make representing some theories cumbersome. For example, in the theory of vector spaces there are two types of objects that are of interest to us: *vectors* and *scalars*. If we approach this by defining a signature whose terms can represent either vectors or scalars, along with predicates for checking whether an element is a vector or a scalar, functions on vectors would become partial. We could work around this by adding a third "type" of element to represent invalid results for these functions, but this quickly becomes cumbersome.

Many sorted logics offer a solution to this. A many sorted signature is a pair $\Sigma = (S, F)$ where S is a set of sorts, and F is a $S^* \times S$-indexed set of function symbols $F = \{F_{a,r} : (a, r) \in S^* \times S\}$. If $f \in F_{s_1 \times \ldots \times s_n, s}$, we write $f : s_1 \times \ldots \times s_n \to s$.

For a many-sorted signature Σ, a many-sorted Σ-algebra, is a pair $(A, \_\_A)$, where $A = \{A_s\}_{s \in S}$ is an S indexed set, and $\_\_A$ is the interpretation map, mapping each function symbol $f : s_1 \times \cdots \times s_n \to s$ to a function $f_A : A_{s_1} \times \cdots \times A_{s_n} \to S_s$ (J. Meseguer 2017a). Terms, formulae and sentences are defined as they traditionally are in first order logic. Now, for the theory of vector spaces, we can define a signature with two sorts: one for vectors and another for scalars and use it to axiomatize vector spaces concisely.

However, we can do better than many-sorted logic. Take the theory of lists. The head function takes a non-empty list and returns its first element. But, what happens when the list is empty? What does the head function return in the case of an empty list? The head function must be partial. Order-sorted signatures allow formallizing such partiality (Meseguer 2013).

An order sorted signature $\Sigma = ((S, \leq_s), F)$ is a triple where (S, F) is a many-sorted signature, and $\leq_s$ is a partial order on the set S. Models of order-sorted theories are order-sorted $\Sigma-$algebras. For an order-sorted signature Σ an order-sorted $\Sigma-$algebra is a many-sorted $\Sigma-$algebra, $(A, \_\_A)$, satisfying the following additional conditions:

1. If $s \leq_s s'$, then $A_s \subset A_{s'}$.

2. Given constant symbols $c : \epsilon \to s$ and $c : \epsilon \to s'$ with s and s', sorts in the same connected component under $\leq_s$, then their interpretations, $c_A^{\epsilon,s} \in A_a$ and $c_A^{\epsilon,s'} \in A_{s'}$, are the same element.

3. Given function symbols $f : a \to r$ and $f : a' \to r'$ with $a' = s'_1 \times \cdots \times s'_n$ and $a =$

$s_1 \times \cdots \times s_n$ and each s_i and s'_i, and r and r' in the same connected component (i.e. f is *subsort overloaded*) then for each $\vec{a} \in (A_{s_1} \times \cdots \times A_{s_n}) \cap (A_{s'_1} \times \cdots \times A_{s'_n})$ we have $f_{A, s_1 \times \cdots \times s_n \to s}(\vec{a}) = f_{A, s'_1 \times \cdots \times s'_n \to s'}(\vec{a})$ (i.e. subsort-overloaded functions agree on common elements).

In an order-sorted setting, we can define lists with distinct subsorts for the empty list and non-empty lists. The head function can then be defined as a total function with domain non-empty lists.

1.2.2 Equational Logic

A *signature* Σ is a set of function symbols and their arities. An *equational theory* is a pair (Σ, E), where E is a set of algebraic identities on the terms $T_\Sigma(X)$ constructed from the signature Σ with (sorted) variables in X. For example, the group $\mathbb{Z}_5$ could be described as the initial algebra of an equational theory for a signature $\Sigma = \{0, 1, \_+\_, -\_\}$ with the following equations E:

$x + 0 = x$	Additive Identity
$x + (y + z) = (x + y) + z$	Associativity
$x + y = y + x$	Commutativity
$1 + 1 + 1 + 1 + 1 = 0$	Characteristic 5
$x + (-x) = 0$	Inverses

Note that underscores in the signature indicate holes for subterms, and thus indicate the arity of the symbol.

This equational theory can be implemented as a Maude *functional module* as follows:

```
fmod Z5 is
    sorts Z5 .
    op 0 : -> Z5                             [ctor] .
    op 1 : -> Z5                             [ctor] .
    op _ + _ : Z5 Z5 -> Z5   [assoc comm id: 0 ctor] .
    op   - _ : Z5    -> Z5                          .

    vars x y : Z5                       . --- x and y are variables of sort Z5
    eq (-0)              = 0            . --- Inverse of 0
    eq (-1)              = 1 + 1 + 1 + 1 . --- Inverse of 1
    eq 1 + 1 + 1 + 1 + 1 = 0            . --- Characteristic 5
    eq -(x + y)          = (-x) + (-y)  . --- Inverse distribute
endfm
```

This program represents an equational theory $E = ((S, \leq_S), \Sigma, E \cup B)$. Here, $S = \{\mathtt{Z5}\}$ $\leq_{\mathtt{s}} = \{\}$ and $\Sigma = \{\mathtt{0}, \mathtt{1}, \_ + \_\}$. The `fmod Z5 is ... endfm` construct defines a *functional module* that describes an equational theory. The signature of this theory has a single sort `Z5`. The `op` declaration defines the terms and functions in the signature of that theory. These are of the form `op NAME : ARGUMENTS -> RESULT [ATTRIBUTES]`. For example, `_ + _` takes two terms of sort `Z5` and returns another of the same sort, while `0` and `1` are constants of sort `Z5`. The `ctor` attribute marks a term as part of the constructor signature of the theory. The `assoc`, `comm` and `id: 0` attributes mark the plus operator as being associative, commutative and having `0` as its identity. The `vars` declaration allows using the tokens `x` and `y` as variables in equations. Each `eq` construct represents an axiom in the equational theory.

Although ordinarily equations in equational theories are symmetric – in a proof we may replace equals by equals if a term matches either the left hand side or the right hand side – equations in Maude are only applied from left to right. This is to allow defining a terminating execution and also, by choosing equations carefully so that they are confluent, to ensure a unique result for every terminating execution of a term. Attributes like `assoc` and `comm` allow specifying common axioms that would otherwise be difficult to define in a terminating manner (and also make computations using Maude's efficient matching algorithms modulo such axioms considerably more expressive, with very succinct specifications.) Because of this directionality, the theories must be *confluent* for them to form a well-defined equational theory. i.e. the application of equations must yield the same final result irrespective of the order in which eqautions are applied. Although tools such as the Church-Rosser Checker and the Maude Termination Tool are provided to help check these, the burden of making sure that functional modules are confluent and terminating is ultimately on the programmer defining them. This orientation on the equations means that we will sometimes have to define equations that would otherwise be mathematically deducible. For example, if we had defined the functional module with the same equations as the equational theory, Maude would not have been able to deduce that $-3 = 2$. However, it is trivial to show that each set of equations can be derived from the other. In spite of this, it can be seen from the example above that the representational distance between an equational theory and its implementation in Maude very small.

Besides the syntax demonstrated above, Maude also supports conditional equations, i.e. an equation that fires only when some predicate expression holds for an equation's instance, and also an "otherwise" clause – an equation that will fire when no other equation holds.

1.2.3 Rewriting Logic

A rewrite theory $\mathcal{R}$ is a triple (Σ, E, R), where (Σ, E) is an equational theory and R the set of *one step rewrite rules* on the terms of the signature.

The rewrite rules R define a relation $\longrightarrow_R \subset T_\Sigma \times T_\Sigma$. This relation is obtained from the closure of R under *reflexivity*, $E-$*equality* (equality under the set of axioms E), *congruence* (if a subterm rewrites, then the rewrite "lifts" to all terms containing that subterm; $t \longrightarrow_R t' \implies f(\ldots, t, \ldots) \longrightarrow_R f(\ldots, t', \ldots)$), *replacement* (for any substitution θ, $t \longrightarrow_R t' \implies t\theta \longrightarrow_R t'\theta$) and *transitivity.* If $x \longrightarrow_R y$, we say "x rewrites to y".

This relation defines a transition system, where the system states are precisely the elements of the initial algebra $T_{\Sigma/E}$ associated to the equational theory (Σ, E). Execution of a program in Maude – reducing a concrete term via the rewrite relation $\longrightarrow_R$ – involves following the edges of this transition graph and terminates when the term it arrives at has no outward edges. Maude can also perform symbolic execution, i.e. reduce a term that has variables, as well as search the structure for terms matching a pattern or predicate. By defining state predicates by means of equations and adding these equations as well as the predicate symbols to the rewrite theory, the above transition system becomes a Kripke structure. Kripke Structures are commonly used in the implementation of model checking and are the structures over which Linear and Branching Temporal Logics are defined. Again, this makes the representational distance between the specification of the model and the data structures we use to reason over it minimal, making verification of correctness of model checkers and other tools that reason over these structures easier than that of model checkers where systems are specified in some imperative langaugee

Rewrite theories are defined in Maude through *system modules*. Since we implement the Nelson-Oppen combination algorithm purely as a functional module, we do not go into the details of the syntax for system modules here.

1.2.4 Reflective logic

Rewriting logic is a *reflective logic* – its meta theory can be represented at the object level in a consistent way. i.e. there is a *universal theory* U and a function $\overline{(\_\vdash\_)}$ such that for any theory T, $T\vdash\varphi \iff U\vdash\overline{T\vdash\varphi}$. This is particularly interesting because it allows us to implement both the models we work over, and the theorem proving and the model checking tools we use in the same language. In fact, the implementation of variant-based satisfiability by Stephen Sherik and of the Nelson-Oppen Combination Algorithm here crucially take advantage of this.

In Maude, the built-in module `META-LEVEL` is used to do this lifting. Terms are represented in the sort `Term`, and modules in the sort `Module`. The function `upModule : ModuleExpression Bool -> Module` takes a `ModuleExpression`, a quote followed by the module name (e.g. `'Z5`) and returns a term representing the module. Similarly, the function `upTerm : Universal -> Term` takes a term of any sort and returns a meta-term, i.e. a term of sort `Term`. Constants, function symbols and variables in a term are represented using quoted identifiers. Arguments of a function symbol in a term are placed in a comma separated list within square brackets. Constants and variables have their sorts annotated as part of the identifier. For example the term `1 + 1` is represented at the meta level as `'_+_[ '1.Z5, '1.Z5 ]`, while the variable `X` of sort `Z5` as `'X:Z5`. Meta-terms can be reduced in a reflective way by the reflected equations of a reflected functional module using the `metaReduce` function. `META-LEVEL`'s `upModule` function allows us to lift a theory and perform rewrites with it like any other term.

1.3 Decision Procedures in Maude

There are a several satisfiability procedures available in Maude, either implemented in Maude at the meta level, or in external tools and made accessible in Maude through their API. It is these tools that we shall use as the base solvers for the Nelson-Oppen combination method.

1.3.1 Variant-based Satisfiability

Variant-based satisfiability is a theory-generic procedure that applies to initial models of a large class of user-definable order-sorted equational theories. The equations of such theories must satisfy the *finite variant property* (FVP) (Escobar, Sasse, and Meseguer 2012)(Comon-Lundh and Delaune 2005) and may include axioms such as commutativity, associativity-commutativity, or identity.

Let $T = (\Sigma, E \cup B)$ where the equations E are confluent, terminating and B-coherent modulo axioms B. A $E, B-$variant of a term t is a pair (u, θ) such that $u =_B (t\theta)!_{\vec{E},B}$, where for any term u, $u!_{\vec{E},B}$ denotes the fully simplified term obtained by exhaustive simplification with the oriented equations $\vec{E}$ modulo B. Given variants (u, θ) and (v, γ) of t, (u, θ) is more general than (v, γ) iff there is a substitution ρ such that:

1. $\theta\rho =_B \gamma$ and
2. $u\rho =_B v$

A theory T has the finite variant property (FVP) iff for each term t there is a finite most general complete set of variants. If a theory $(\Sigma, E \cup B)$ is FVP and B has a finitary $B-$unification algorithm, then folding variant narrowing gives a finitary $E \cup B$-unification algorithm (Escobar, Sasse, and Meseguer 2012).

Furthermore, if $(\Sigma, E \cup B) \supseteq (\Omega, E_\Omega \cup B_\Omega)$ is a subsignature of constructors and $(\Omega, E_\Omega \cup B_\Omega)$ is OS-compact, then satisfiability of quantifier free formulae in the initial algebra of this theory is decidable by variant-based satisfiability. This has been implemented in Maude by Sherik and Meseguer (Skeirik and Meseguer 2016) and will be used for demonstrating the order-sorted Nelson-Oppen combination method. Refer to (Meseguer 2016) for a more in-depth description.

1.3.2 CVC4 and Yices

CVC4 is an industry-standard automatic theorem prover that supports many theories including rational and integer linear arithmetic, arrays, bitvectors and a subset of non-linear arithmetic (Barrett et al. 2011). Yices is another state of the art SMT solver that supports in addition tuples and scalar types and excels in non-linear real and integer arithmetic (Dutertre 2014). Maude allows interaction with these solvers via their respective C APIs.

Although CVC4 and Yices allow defining algebraic data types they do not allow terms in these data types to be identified by additional axioms or have any operations except term constructors and term selectors. Variant based satisfiablilty includes these simple algebraic data types as a special case, but it covers a much wider class of algebraic specifications: it allows user-defined functions (provided their equations are FVP), and structural axioms such as combinations of commutative, associative commutative, and identity axioms. Therefore, it complements CVC4, Yices and other SMT solvers by allowing a very wide range of user-definable decidable theories.

Chapter 2

Order Sorted Nelson Oppen as a rewrite theory

Given decision procedures for the quantifier free formulae in several theories the Order-Sorted Nelson-Oppen combination method gives us a decision procedure for the quantifier free fragment in the combination of these theories, provided that the theories are *disjoint*, *stably infinite* for their shared sorts and *optimally intersecting*. In theories stably infinite in a set of sorts, we can find models for each theory such that the cardinalities of the carrier sets of those sorts match.

Stably Infinite Let T be an order-sorted first-order theory with signature $\Sigma = ((S, \leq), F, P)$ and $s_1, s_2, \ldots s_n \in S$. Let $\mathcal{F} \subset \text{FirstOrderFormula}(\Sigma)$, be the set of first order formulae in Σ

T is stably infinite in sorts $s_1, s_2, \ldots s_n$ for $\mathcal{F}-$satisfiability iff every $T-$satisfiable formula $\varphi \in \mathcal{F}$, is also satisfiable in a model $\mathcal{B} = (B, \_\__B) \in \text{mod}(T)$ such that $|B_{s_i}| \geq \chi_0, 1 \leq i \leq n$.

For Nelson-Oppen combinations, requiring that both theories T_1 and T_2 are stably infinite intuitively means that we can always find models of both theories where the cardinalities of sorts $s_1, \ldots, s_n$ agree.

Notation: For sort s and signature Σ_i, let $[s]_i$ denote it's connected component of sorts in Σ_i

Optimally intersectable (J. Meseguer 2017b) The order-sorted signatures Σ_1 and Σ_2 are optimally intersectable iff:

1. **Functions and predicates sorts agree:** For each $f \in \text{fun}(\Sigma_1) \cap \text{fun}(\Sigma_2)$ (resp, $p \in \text{pred}(\Sigma_1) \cap \text{pred}(\Sigma_2)$), $\exists \{i, j\} \in \{1, 2\}$ such that:
   - $F_i(f) = F_j(f) \cap ([s_1]_i \times \cdots \times [s_m]_i) \times [s_i]$ (resp $P_i(p) = P_j(p) \cap ([s_1]_i \times \cdots \times [s_m]_i)$
   - $[s_l] \subset [s_l]_j, 1 \leq l \leq n$, and $[s]_i \subset [s]_j$ (resp. $[s_l]_i \subset [s_l]_j, 1 \leq l \leq n$).
2. **Intersection is a single component:** For every sort $s \in S_0$, we have $[s]_1 \cap S_2 = [s]_2 \cap [s]_1 = [s]_1 \cap [s]_2$
3. and, for any two sorts $s_i \in S_i$ and $s_j \in S_j$ any one of:
   1. **Intersection is empty**: $[s_i]_i \cap [s_j]_j = \emptyset$
   2. **Intersection is the top sort of one component:** $[s_i]_1 \cap [s_j]_2 = \{s_0\}$, where s_0 is the top-sort of at least one of the connected components.
   3. **Once component is subsumed in the other:**
      a. $\exists k \in \{1, 2\}$ and $[s_k]_k$ has a top sort, $[s_k]_k \subset [s_l]_l$ $\{k, l\} = \{1, 2\}$.
      b. $\leq_k \cap [s_k] = \leq_l \cap [s_k]_2^2$
      c. (downward closure): $\forall s \in [s_l]_l, \forall s' \in [s_k]_k, s \leq_l s' \implies s \in [s_k]_k$

Given two order-sorted, optimally intersecting, stably-infinite theories T_1 and T_2 with disjoint signatures Σ_1 and Σ_2 each with decision procedures for quantifier free

T_i-satisfiability we want to derive a decision procedure for quantifier free $T_1 \cup T_2$ satisfiability. We can transform any formula φ into an *equisatisfiable* formula in disjunctive normal form. Further, for each atom in such a formula we can apply "purification" to obtain a formula where each atom is in the signature of one of the two theories.

Now, our task has become to find a $T_1 \cup T_2$-model M_0 and an assignment $a :$ vars$(\varphi) \to M_0$ such that $M, a \models \varphi$. How can we decompose this satisfiability problem into similar subproblems for the theories T_1 and T_2? What follows summarizes more detailed arguments in (J. Meseguer 2017b) about the order-sorted Nelson-Oppen combination. Because of the stably infinite assumptions on the theories T_1, T_2, as well as the assumption that the corresponding signatures are optimally intersectable and disjoint, if φ is purified into an equisatisfiable conjunction $\varphi_1 \wedge \varphi_2$ of formulas φ_1 and φ_2 in the theories T_1 and T_2, we can always choose M so that the shared sorts in M have infinite cardinalities, so that M is the *amalgamation* of models M_1 and M_2 of theories T_1 and T_2 that satisfy φ_1 and φ_2, and that have the same cardinality for the shared sorts. The interesting question is the converse one: under what conditions, given models M_1 and M_2 with same cardinality in shared sorts, and assignments a_1 and a_2 such that $M_i, a_i \models \varphi_i$, $1 \leq i \leq 2$ can we amalgamate M_1 and M_2 into a single $T_1 \cup T_2$-model M with an assignment a such that $M, a \models \varphi$? The answer is that such an amalgamation M and an assignment a extending both a_1 and a_2 after a suitable bijective identification of the sets for the shared sorts will exist if and only if a_1 and a_2 generate the same equivalence relation among the variables of φ_1 and φ_2 that have shared sorts. Therefore, since the satisfaction of any such equivalence relation can be characterized by a conjuction ψ of equalities and disequalities among shared variables (called an "arrangement"), a "naive" order-sorted Nelson-Oppen algorithm amounts to finding such a ψ among all possible equivalence relations such that $\varphi_i \wedge \psi$ is T_i-satisfiable, $1 \leq i \leq 2$.

The question now becomes: how do we efficiently find such an arrangement of variables? Checking each equivalence class for satisfiability is infeasable as the number of equivalence classes grows exponentially with the number of variables, even in the order sorted case where we can restrict ourselves to equivalences compatible with the sort structure of the signatures (e.g. we cannot have an equality between a boolean and an integer variable). Instead of checking each of the possible partitions on the shared variables, we choose a Darwinian approach, pruning classes of equivalences from the search space if an identification of a single pair of variables implied by one theory is not satisfiable in another (equality propagation). In the case of non-convex theories, we may have $\varphi \longrightarrow (x_1 = y \vee x_2 = y)$ without either $\varphi \longrightarrow x_1 = y$ or $\varphi \longrightarrow x_2 = y$ individually holding. Thus if any theory implies the disjunction of all remaining identifications we branch our search, checking if at least one of the remaining identifications is satisfiable (split). We can think of each equality propagation step of the algorithm as pruning the search space (of arrangements) of unsatisfiable ones, and the split step dividing the search space into smaller groups where the split step can apply. The inference rules for the Equality Propagation and Split rules are given in Figure 2.1 where φ_E denotes the equalities between variables with shared sorts obtained so far by previous inference steps, and CE denotes the still uncommitted equalities between such shared sorts. These rules are similar to

$$\frac{x_m = x_n \in \mathrm{CE} \quad T_i \models (\varphi_i \wedge \varphi_E) \longrightarrow x_m = x_n \quad \mathrm{NelsonOppenSat}(\varphi_1 \wedge \varphi_2 \wedge \varphi_E, \mathrm{CE})}{\begin{array}{ll} & \mathrm{CheckSat}(\varphi_j \wedge \varphi_E \wedge x_m = x_n) \\ \wedge & \mathrm{NelsonOppenSat}(\varphi_1 \wedge \varphi_2 \wedge \varphi_E \wedge x_m = x_n, \mathrm{CE} \setminus \{x_m = x_n\}) \end{array}} \text{ Equality Propagation}$$

$$\frac{T_i \models (\varphi_i \wedge \varphi_E) \longrightarrow \bigvee \mathrm{CE} \quad \mathrm{NelsonOppenSat}(\varphi_1 \wedge \varphi_2 \wedge \varphi_E, \mathrm{CE})}{\bigvee_{x_m = x_n \in \mathrm{CE}} \left(\begin{array}{ll} & \mathrm{CheckSat}(\varphi_1 \wedge \varphi_E \wedge x_m = x_n) \\ \wedge & \mathrm{CheckSat}(\varphi_2 \wedge \varphi_E \wedge x_m = x_n) \\ \wedge & \mathrm{NelsonOppenSat}(\varphi_1 \wedge \varphi_2 \wedge \varphi_E, \mathrm{CE} \setminus \{x_m = x_n\}) \end{array} \right)} \text{ Split}$$

Figure 2.1: Inference rules for the Nelson-Oppen algorithm

rules presented in (Manna and Zarba 2003) for the unsorted case and have a similar proof of correctness.

Chapter 3

Implementation in Maude

The Nelson-Oppen algorithm is implemented in Maude as the function `nelson-oppen-sat`. Besides the names of the theories and the unpurified formulae, the algorithm also requires information about which function to use to check satisfiability, and whether the theory is convex. We use "tagged" formulae to represent this information. For example, the term `tagged('1.Nat ?= '2.Nat, (('mod > 'NAT), ('check-sat > 'var-sat)))` represents the formula "1 = 2" in the module of `NAT`, and that we should use the `var-sat` procedure to check its satisfiability. In the implementation in Maude, these tagged formula are represented by the sort `TaggedFormula` and sets of tagged formulae by the sort `TaggedFormulaSet`. For rewriting logic variables (not to be confused with variables part of the formula we are rewriting over) of the sort `TaggedFormula` we use the variables `TF1` and `TF2`, while for `TaggedFormulaSet` we use `TFS`.

```
op nelson-oppen-sat    : TaggedFormulaSet QFForm                -> Bool .
```

The `nelson-oppen-valid` function converts a validity check into a satisfiability check:

```
op nelson-oppen-valid  : TaggedFormulaSet QFForm -> Bool .
-----------------------------------------------------------
eq nelson-oppen-valid(TFS, PHI) = strictNot(nelson-oppen-sat(TFS, ~ PHI)) .
```

Given a quantifier free formula PHI in the set of theories TFS (each tagged with information regarding covexitivity, and information about which procedure to use for checking sat), we first convert it to disjunctive normal form (DNF) and simplify it (e.g. $\bot \wedge \varphi$ becomes $\bot$).

```
eq nelson-oppen-sat(TFS, PHI)
 = $nosat.dnf(TFS, simplify(toDNF(toNNF(simplify(PHI))))) .
```

The algorithm then considers each disjunction separately.

```
eq $nosat.dnf(TFS, CONJ \/ PHI)
 =  $nosat.dnf(TFS, CONJ) or-else $nosat.dnf(TFS, PHI)
 .
```

We then purify each mixed disjunct into a conjunction of "pure" atoms each wellformed in the signature of one of the theories, and tagged with the appropriate information.

```
ceq $nosat.dnf(TFS , CONJ)
  = $nosat.purified(TFS, purify(ME1, ME2, CONJ))
 if    ( tagged(tt, ('mod > ME1); TS1)
       , tagged(tt, ('mod > ME2); TS2))
    := TFS
  .
 eq $nosat.purified(TFS, CONJ)
  = $nosat.tagged(tagWellFormed(TFS, CONJ)) .
```

Next, we make sure each of the tagged formulae (TF1, TF2) are satisfiable on their own.

```
eq $nosat.tagged((TF1, TF2))
 = check-sat(TF1) and-then check-sat(TF2) and-then $nosat.basicSat(TF1, TF2)
   [print "Purified:\n\t" TF1 "\n\t" TF2]
 .
```

From the set of shared variables $X^{1,2} := \text{vars}(\varphi_1) \cap \text{vars}(\varphi_2)$ we define a set of candidate equalities.

$$\text{CE} := \{x_i = y_i | x_i, y_i \in X^{1,2}_{s_i}, x_i \not\equiv y_i\}$$

where $X^{1,2}_{s_i}$ is the subset of shared variables in the connected component of sort s_i.

```
ceq $nosat.basicSat(TFS)
  = $nosat.ep( TFS
             , candidate-equalities(in-module(moduleIntersect(ME1, ME2)
             , vars(PHI1) ; vars(PHI2)))
             )
 if ( tagged(PHI1, ('mod > ME1); _1:Tags)
    , tagged(PHI2, ('mod > ME2); _2:Tags))
    :=  TFS
  .
```

Next, we apply the equality propagation inference rule. If any identification of variables is implied by a theory, we propagate that identification to the other theories by replacing all occurrences of the variable in the left hand side with that on the right hand side in all formulae and the candidate equalities. Performing the substitution instead of merely adding the equality to the formula has the advantage of reducing the number of candidate equalities we need to try.

```
ceq $nosat.ep(( tagged(PHI1, ('mod > ME1); TS1)
              , tagged(PHI2, ('mod > ME2); TS2)), X1 ?= X2 \/ CANDEQ)
  =          check-sat(tagged(simplify(PHI2 << (X1 <- X2)), ('mod > ME2); TS2))
    and-then $nosat.ep(( tagged(simplify(PHI1 << (X1 <- X2)), ('mod > ME1); TS1)
                       , tagged(simplify(PHI2 << (X1 <- X2)), ('mod > ME2); TS2))
                      , simplify(CANDEQ << (X1 <- X2)))
 if check-valid(tagged(PHI1 => (X1 ?= X2), ('mod > ME1); TS1))
    [ print "EqualityProp: " ME1 ": => " X1 " ?= " X2 ] .
```

If, after checking each identification individually, there are none that are implied we apply the split rule.

```
eq $nosat.ep(TFS, CANDEQ) = $nosat.split(TFS, CANDEQ) [owise print "Split? " CANDEQ] .
```

If there are no variables left to identify, then the formula is satisfiable

```
eq $nosat.split(TFS, mtForm) = true .
```

However, if some disjunction of identifications is implied and we are in a non-convex theory, we "split". i.e. we try each of the possible identification left in turn and see if at least one of them is satisfiable.

```
ceq $nosat.split(TFS, CANDEQ)
  = $nosat.split.genEqs(TFS, CANDEQ, CANDEQ)
 if    ( tagged(PHI1, ('mod > ME1) ; ('convex > 'false) ; TS1)
       , tagged(PHI2, ('mod > ME2) ;                      TS2))
    := TFS
 /\ check-valid(tagged((PHI1) => (CANDEQ), ('mod > ME1); ('convex > 'false) ; TS1))
  .
```

Otherwise, since there are no implied identifications and the theories are stably-infinite, the equation is satisfiable.

```
eq $nosat.split(TFS, CANDEQ) = true [owise] .
```

We use `$nosat.split.genEqs` to generate this disequality of sat problems.

```
eq $nosat.split.genEqs((tagged(PHI1, ('mod > ME1); TS1), tagged(PHI2, ('mod > ME2); TS2))
                     , X1 ?= X2 \/ DISJ?1, X1 ?= X2 \/ DISJ?2)
  = (          check-sat(tagged(PHI1 /\ X1 ?= X2, ('mod > ME1); TS1))
      and-then check-sat(tagged(PHI2 /\ X1 ?= X2, ('mod > ME2); TS2))
      and-then $nosat.ep(( tagged(PHI1 /\ X1 ?= X2, ('mod > ME1); TS1)
                         , tagged(PHI2 /\ X1 ?= X2, ('mod > ME2); TS2))
                        , DISJ?2)
```

```
   )
   or-else $nosat.split.genEqs(( tagged(PHI1, ('mod > ME1); TS1)
                              , tagged(PHI2, ('mod > ME2); TS2))
                          , DISJ?1, X1 ?= X2 \/ DISJ?2)
    [print "Split: "  ME1 " : " X1 " ?= " X2 ]
 .

eq $nosat.split.genEqs(( tagged(PHI1, ('mod > ME1); TS1)
                      , tagged(PHI2, ('mod > ME2); TS2))
                  , mtForm, DISJ?2)
 = false
 .
```

Chapter 4

Examples

4.1 Matrices with real and integer entries

The specification that follows is not exactly the one used in the experiments, but is equivalent to it. There are two somewhat subtle issues about this example, namely: (i) the use of parameterization, and (ii) the use of definitional extensions, that can best be explained using Maude parameter theories, parameterized modules, and parameter instantiation by views.

We can define in Maude the theory of 2×2 matrices over a ring as the following module parameterized by the theory of rings as its parameter theory:

```
fth RING is
    sort Ring .
    op _+_ : Ring Ring -> Ring [assoc comm] .
    op _*_ : Ring Ring -> Ring [assoc comm] .
    op 0 : -> Ring .
    op 1 : -> Ring .
    op - : Ring -> Ring .
    vars x y z : Ring .
    eq x + 0 = x .
    eq 1 * x = x .
    eq x + -(x) = 0 .
    eq x * (y + z) = (x * y) + (x * z) .
endfth
```

```
fmod MATRIX{R :: RING} is
    sort Matrix .
    op matrix : R$Ring R$Ring R$Ring R$Ring -> Matrix [ctor] .

    vars A B C D : R$Ring .

    op m11 : Matrix -> R$Ring .
    op m12 : Matrix -> R$Ring .
    op m21 : Matrix -> R$Ring .
    op m22 : Matrix -> R$Ring .

    eq m11(matrix(A, B, C, D)) = A [variant] .
    eq m12(matrix(A, B, C, D)) = B [variant] .
    eq m21(matrix(A, B, C, D)) = C [variant] .
    eq m22(matrix(A, B, C, D)) = D [variant] .
endfm
```

Next, we define matrix multiplication, determinant and identity as *definitional extensions* of the theory of matrices. That is, these new functions are fully defined in terms of the theory of matrices itself and can always be "evaluated away." This is important to meet the Nelson-Oppen theory disjointness requirement, as explained below.

```
fmod MATRIX-OPS{R :: RING} is
    protecting MATRIX{R} .

    vars A1 B1 A2 B2 : R$Ring .
    vars A B : Matrix .

    op mulSum : R$Ring R$Ring R$Ring R$Ring -> R$Ring .
    eq mulSum(A1, B1, A2, B2) = (A1 * B1) + (A2 * B2) .

    op multiply : Matrix Matrix -> Matrix .
    eq multiply(A, B)
     = matrix(mulSum(m11(A),m11(B),m12(A),m21(B)),
             mulSum(m11(A),m12(B),m12(A),m22(B)),
             mulSum(m21(A),m11(B),m22(A),m21(B)),
             mulSum(m21(A),m12(B),m22(A),m22(B))) .

    op determinant : Matrix -> R*Ring .
    eq determinant(A)
     = (m11(A) * m22(A)) - (m12(A) * m21(A)) .

    op identity : -> Matrix .
    eq identity = matrix(1, 0, 0, 1) .
endfm
```

Next we instantiate the theory of rings to the module for the theory of Reals using a view:

```
view Real from RING to REAL is
    sort Ring to Real .
    op 0 to 0/1 .
    op 1 to 1/1 .
    op _+_ to _+_ .
    op _-  to _- .
    op _*_ to _*_ .
endv

fmod  MATRIX-REAL is
  protecting MATRIX-OPS{Real} .
endfm
```

What is crucial about this theory instantiation is that, since the operators in MATRIX-OPS are all definitional extensions, they can all be evaluated away to their righthand sides, i.e., to operators in the disjoint union of two theories: (i) the FVP theory MATRIX obtained by completely removing its RING parameter part, and (ii) the theory REAL to which the parameter theory RING is instantiated. Therefore, the order-sorted Nelson-Oppen algorithm can be invoked to decide validity and satisfiability of formulas in `MATRIX-REAL`, once we: (i) evaluate away all defined operations in `MATRIX-OPS` appearing in a formula, and (ii) purify the formula into its two disjoint parts.

We cannot, at the moment, use this specification as is, because the Nelson-Oppen implementation does not support views yet. Instead, we execute the the following query against an equivalent specification of real matrices:

```
reduce in MATRIX-TEST : nelson-oppen-valid(
    ( tagged(tt, (('mod > 'MATRIX-REAL); ('check-sat > 'var-sat)))
    , tagged(tt, (('mod > 'REAL);        ('check-sat > 'smt-sat)))
    ),
         (multiply('A:Matrix, 'B:Matrix) ?= identity('0/1.Real, '1/1.Real))
      => (determinant('A:Matrix) != '0/1.Real)
    ) .
```

The negation of this forumla (since we are checking validity) purifies to the following the formula in the theory of reals:

```
    '0:Real ?= '0/1.Real
 /\ '1:Real ?= '1/1.Real
 /\ 'p11:Real ?= '_+_['_*_['a11:Real, 'b11:Real],'_*_[ 'a12:Real, 'b21:Real]]
 /\ 'p12:Real ?= '_+_['_*_['a11:Real, 'b12:Real],'_*_[ 'a12:Real, 'b22:Real]]
 /\ 'p21:Real ?= '_+_['_*_['a21:Real, 'b11:Real],'_*_[ 'a22:Real, 'b21:Real]]
 /\ 'p22:Real ?= '_+_['_*_['a21:Real, 'b12:Real],'_*_[ 'a22:Real, 'b22:Real]]
 /\ '0/1.Real ?= '_-_['_*_['a11:Real, 'a22:Real],'_*_[ 'a12:Real, 'a21:Real]]
```

and, in the theory of Matrices:

```
    'a11:Real ?= 'm11['A:Matrix]  /\ 'b11:Real ?= 'm11['B:Matrix]
 /\ 'a12:Real ?= 'm12['A:Matrix]  /\ 'b12:Real ?= 'm12['B:Matrix]
 /\ 'a21:Real ?= 'm21['A:Matrix]  /\ 'b21:Real ?= 'm21['B:Matrix]
 /\ 'a22:Real ?= 'm22['A:Matrix]  /\ 'b22:Real ?= 'm22['B:Matrix]
 /\     'matrix['1:Real  ,'0:Real  , '0:Real ,'1:Real  ]
     ?= 'matrix['p11:Real,'p12:Real,'p21:Real,'p22:Real]
```

Next, each theory propagates equalities that are implied by each formula:

```
'MATRIX-REAL: => '0:Real   ?= 'p12:Real
'MATRIX-REAL: => '1:Real   ?= 'p11:Real
'REAL:        => 'p12:Real ?= 'p22:Real
'MATRIX-REAL: => 'p11:Real ?= 'p21:Real
'REAL:        => 'a11:Real ?= 'a21:Real
'REAL:        => 'a12:Real ?= 'a22:Real
'MATRIX-REAL: => 'p21:Real ?= 'p22:Real
```

But, this last identification is a contradiction in the theory of reals. p_{22} cannot equal p_{21} since $p_{22} = p_{12} = 0$, while $p_{21} = p_{11} = 1$. Thus, the negation is unsatisfiable and the original formula must be valid.

It turns out that if we combine this module with the Integers instead of the Reals, we can prove something stronger: that any invertible matrix must have determinant ± 1. Unfortunately, CVC4 is not able to solve the non-linear arithmetic needed to prove this. We must instead turn to the Yices solver, the other SMT solver available in Maude. Even so, the default configuration for Yices does not enable the solver for non-linear arithmetic (MCSAT), and running this example involved modifying the Maude C++ source code to enable that configuration. Even so, the computational difficulty involved in solving non-linear integer arithmetic forced us to restrict the proof to upper-triangular matrices.

```
reduce in MATRIX-TEST : nelson-oppen-valid(
       ( tagged(tt, (('mod > 'MATRIX-INTEGER);
            ('check-sat > 'var-sat); ('convex > 'true)))
       , tagged(tt, (('mod > 'INTEGER       );
            ('check-sat > 'smt-sat); ('convex > 'false)))
       ),
           (    multiply('A:Matrix, 'B:Matrix) ?= identity('0.Integer, '1.Integer)
             /\ 'm21['A:Matrix] ?= '0.Integer
```

```
          /\ 'm21['B:Matrix] ?= '0.Integer
        )
     => (    determinant('A:Matrix) ?= '1.Integer
          \/ determinant('A:Matrix) ?= '-_['1.Integer]
        )
  ) .
```

In the theory of integers this purifies to:

```
   '0:Integer ?= 'a21:Integer /\ '0:Integer ?= 'b21:Integer
/\ '0:Integer ?= '0.Integer   /\ '1:Integer ?= '1.Integer
/\ 'p11:Integer ?= '_+_[ '_*_['a11:Integer, 'b11:Integer]
                       , '_*_['a12:Integer, 'b21:Integer]]
/\ 'p12:Integer ?= '_+_[ '_*_['a11:Integer, 'b12:Integer]
                       , '_*_['a12:Integer, 'b22:Integer]]
/\ 'p21:Integer ?= '_+_[ '_*_['a21:Integer, 'b11:Integer]
                       , '_*_['a22:Integer, 'b21:Integer]]
/\ 'p22:Integer ?= '_+_[ '_*_['a21:Integer, 'b12:Integer]
                       , '_*_['a22:Integer, 'b22:Integer]]
/\ '1.Integer != '_-_['_*_[ 'a11:Integer, 'a22:Integer]
                       ,'_*_[ 'a12:Integer, 'a21:Integer]]
/\ '-_['1.Integer] != '_-_[ '_*_['a11:Integer, 'a22:Integer]
                          , '_*_[ 'a12:Integer, 'a21:Integer]]
```

and, in the theory of matrices to:

```
   '0:Integer ?= 'a21:Integer      /\ '0:Integer ?= 'b21:Integer
/\ 'a11:Integer ?= 'm11['A:Matrix] /\ 'b11:Integer ?= 'm11['B:Matrix]
/\ 'a12:Integer ?= 'm12['A:Matrix] /\ 'b12:Integer ?= 'm12['B:Matrix]
/\ 'a21:Integer ?= 'm21['A:Matrix] /\ 'b21:Integer ?= 'm21['B:Matrix]
/\ 'a22:Integer ?= 'm22['A:Matrix] /\ 'b22:Integer ?= 'm22['B:Matrix]
/\     'matrix[ '1:Integer,'0:Integer, '0:Integer,'1:Integer]
    ?= 'matrix['p11:Integer,'p12:Integer,'p21:Integer,'p22:Integer]
```

Similar equalities are propagated:

```
'INTEGER:        => '0:Integer   ?= 'p21:Integer
```

```
'INTEGER:        => 'p21:Integer ?= 'a21:Integer
'INTEGER:        => 'a21:Integer ?= 'b21:Integer
'MATRIX-INTEGER: => '1:Integer   ?= 'p11:Integer
'INTEGER:        => 'a11:Integer ?= 'b11:Integer
'MATRIX-INTEGER: => 'p11:Integer ?= 'p22:Integer
```

leading to a complex contradiction forcing some elements to be inverses of others in an impossible way, allowing us to conclude that the original formula is valid.

4.2 Hereditarily Finite Sets with Reals

In this example, we demonstrate the combination algorithm with non-convex theories – non-linear real arithmetic and hereditarily finite sets. Hereditarily finite sets is an example of a theory not currently definable in CVC4 or Yices2 because of its use of algebraic data types modulo axioms like associativity-commutativity and having FVP equations. Hereditarily finite sets (HFS) are a model of set theory without the axiom of infinity. Although hereditarily finite sets are expressive enough to encode constructs like the integers and the natural numbers, its initial model is a countable model and so cannot encode the real numbers.

We have three sorts, `X`, the parametric sort, `Set`s and `Magma`s. Both `X`s and `Set`s are `Magma`s.

```
    sorts X Set Magma .
    subsorts X Set < Magma .
```

The elements of a hereditarily finite set can be elements of the parameter sort `X` of "atomic elements", or can be other hereditarily constructed inductively from the following three constructors. First, `empty` is a `Set`:

```
    op empty :              -> Set                                       [ctor] .
```

Second, the union operator is an associative, commutative and idemopotent operator:

```
    op _ , _ : Magma Magma -> Magma                          [ctor assoc comm] .
    ----------------------------------------------------------------------------
    eq M , M , M' = M , M'                                            [variant] .
    eq M , M      = M                                                 [variant] .
```

Finally, a `Set` may be constructed from any `Magma` by enclosing it in braces.

```
    op { _ } : Magma       -> Set                                       [ctor] .
```

We also have a subset operator and the various equations (not detailed here) defining it:

```
    op _ C= _ : Magma Magma -> MyBool                                          .
```

We instantiate this module with `Real`s as a subsort of `X`:

```
fmod HFS-REAL is
    including HEREDITARILY-FINITE-SET .
    sorts Real .
    subsorts Real < X .

    op fake-0 :      -> Real  [ctor] .
    op fake-s : Real -> Real  [ctor] .
endfm
```

Finally, we check the satisfiability of the formula $\{x^2, y^2, z^2\} \subseteq \{a\} \wedge x \neq y.$ i.e. "is it possible for the set of squares of three numbers, two of which must be distinct, to be a subset of a set with a single element." This is indeed possible, since every positive real number has two distinct square roots. Since set union is idempotent, if the two distinct numbers are additive inverses of each other and the third is equal to either, then the proposition would indeed be satisfied.

Our query is:

```
reduce in NELSON-OPPEN-COMBINATION :
       nelson-oppen-sat( ( tagged(tt, ('mod > 'REAL)    ; ('check-sat > 'smt-sat))
                         , tagged(tt, ('mod > 'HFS-REAL); ('check-sat > 'var-sat))
                         )
                         , (  '_C=_[ '`{_`}['_`,_[ '_*_ [ 'Z:Real, 'Z:Real ]
                                                 , '_*_ [ 'X:Real, 'X:Real ]
                                                 , '_*_ [ 'Y:Real, 'Y:Real ]
                                                 ]]
                                   , '`{_`}['A:Real]]
                              ?= 'tt.MyBool
                           )
                           /\ 'X:Real != 'Y:Real
                         ) .
```

This purifies to:

```
    'x2:Real ?= '_*_['X:Real,'X:Real]
 /\ 'y2:Real ?= '_*_['Y:Real,'Y:Real]
 /\ 'z2:Real ?= '_*_['Z:Real,'Z:Real]
 /\ 'X:Real != 'Y:Real,
```

in the theory of the hereditarily finite sets, and to:

```
    'tt.MyBool ?= '_C=_['`{_`}['_`,_['z2:Real,'x2:Real,'y2:Real]],'`{_`}['A:Real]]
 /\ 'X:Real != 'Y:Real
```

in the theory of the reals.

Initially, a few equalities are propagated from the theory of hereditarily finite sets:

```
'HFS-REAL: => 'x2:Real ?= 'y2:Real
'HFS-REAL: => 'y2:Real ?= 'z2:Real
'HFS-REAL: => 'z2:Real ?= 'A:Real
```

Since no more identifications of variables are implied on their own and the theories are not convex, the algorithm must check whether a disjunction of identifications is implied by either of the theories, and indeed $x = z \vee y = z$ is implied. The algorithm splits the search space on the remaining candidate equalities ($a = x$, $a = y$, $a = z$, $x = y$, $z = z$ and $y = z$). It first tries the case where $a = x$ and finds that there are satisfiabile arrangements (this can happen when $a = x = 1$). It then splits the search space again, but finds that there are no arrangements $a = y$ possible (since that implies that $x = y$). However the case where $a = z$ is satisfiable. This causes the the equality $x = z$ to be propagated. Now, since no further equalities or disjunctions thereof hold, the algorithm concludes that the formula is satisfiable.

Chapter 5

Conclusion & Future work

The examples above have demonstrated the usefulness of Nelson-Oppen combination in Maude. Even so, the tool is still a prototype. As mentioned previously, the Nelson-Oppen method forms the keystone of general SMT solving. Other key pieces need to be implemented in Maude for the solver to be efficient and viable.

For example, in this implementation, prior to purification and to applying the Nelson-Oppen algorithm, we convert the formula into its DNF form. This can lead to an exponential blow up in the length of the formula. A more efficient solution would be to take advantage of a boolean structure by using a SAT solver by extension of the DPLL algorithm (Davis and Putnam 1960) to the so-called DPLL(T) algorithm (Nieuwenhuis, Oliveras, and Tinelli 2006)(Krstić and Goel 2007).

Being a prototype, little effort has been spent on optimization. For example, when working with the `var-sat` solver, the list of most general unifiers is computed repeatedly at every query to the solver. Computing this list can be expensive depending on the term and theory under consideration. For example, in an extreme case checking the satisfiability of the forumala `{ X:Magma, Y:Magma, Z:Magma } C= { X:Magma } ?= true` using `var-sat` takes tens of minutes. Most of this time is spend calculating this list of unifiers. If such a formula were to arise in the course of the Nelson-Oppen algorithm, the list of unifiers may need to be computed several times at prohibitive cost.

Stable infiniteness requires that the theory has infinite models. However, there are several important theories that are not stably infinite. For example, the theory of bit vectors ($\mathbb{Z}/2^n\mathbb{Z}$) can be used to model "machine integers" widely used by many programming languages. In (Tinelli and Zarba 2003), Tinelli and Zarba showed that this requirement can be reduced to the case where all but one of the theories is "shiny". Further work by Ranise, Ringeissen and Zarba (Ranise, Ringeissen, and Zarba 2005), and by Jovanovi and Barrett (Jovanović and Barrett 2010) provided an easier to compute alternative called strongly "polite" theories. Extending this implementation to handle these cases would greatly expand the usefulness of these theories.

Work also needs to be done to expand the the implementation to handle more than two theories at a time, and theories that share a sub-signature, though this work is mostly on the purification front.

In general, one can envision incrementally building up towards a flexible, efficient and powerful SMT infrastructure in Maude delegating subproblems both to external solvers as well as to tools that leverage the power and expressiveness of rewriting logic.

Chapter 6

References

Barrett, Clark, Christopher L. Conway, Morgan Deters, Liana Hadarean, Dejan Jovanovi'c, Tim King, Andrew Reynolds, and Cesare Tinelli. 2011. "CVC4." In *Proceedings of the 23rd International Conference on Computer Aided Verification (Cav '11)*, edited by Ganesh Gopalakrishnan and Shaz Qadeer, 6806:171–77. Lecture Notes in Computer Science. Springer. `http://www.cs.stanford.edu/~barrett/pubs/BCD+11.pdf`.

Comon-Lundh, Hubert, and Stéphanie Delaune. 2005. "The Finite Variant Property: How to Get Rid of Some Algebraic Properties." In *Term Rewriting and Applications*, edited by Jürgen Giesl, 294–307. Berlin, Heidelberg: Springer Berlin Heidelberg.

Davis, Martin, and Hilary Putnam. 1960. "A Computing Procedure for Quantification Theory." *Journal of the ACM* 7 (3): 201. `http://www.library.illinois.edu.proxy2.library.illinois.edu/proxy/go.php?url=http://search.ebscohost.com.proxy2.library.illinois.edu/login.aspx?direct=true&db=edb&AN=73453662&site=eds-live&scope=site`.

Dutertre, B. 2014. *Yices 2.2.* Vol. 8559 LNCS. Lecture Notes in Computer Science. Computer Science Laboratory, SRI International: Springer

Verlag. http://www.library.illinois.edu.proxy2.library.illinois.edu/proxy/go.php?url=http://search.ebscohost.com.proxy2.library.illinois.edu/login.aspx?direct=true&db=edselc&AN=edselc.2-52.0-84904793529&site=eds-live&scope=site.

Eker, Steven, Merrill Knapp, Keith Laderoute, Patrick Lincoln, and Carolyn Talcott. 2004. "Pathway Logic: Executable Models of Biological Networks." *Electronic Notes in Theoretical Computer Science* 71: 144–61. https://doi.org/https://doi.org/10.1016/S1571-0661(05)82533-2.

Escobar, Santiago, Catherine Meadows, and José Meseguer. 2006. "A Rewriting-Based Inference System for the Nrl Protocol Analyzer and Its Meta-Logical Properties." *Theoretical Computer Science* 367 (1): 162–202. https://doi.org/https://doi.org/10.1016/j.tcs.2006.08.035.

Escobar, Santiago, Ralf Sasse, and José Meseguer. 2012. "Folding Variant Narrowing and Optimal Variant Termination." *The Journal of Logic and Algebraic Programming* 81 (7): 898–928. https://doi.org/https://doi.org/10.1016/j.jlap.2012.01.002.

Jovanović, Dejan, and Clark Barrett. 2010. "Polite Theories Revisited." In *International Conference on Logic for Programming Artificial Intelligence and Reasoning*, 402–16. Springer.

Krstić, Sava, and Amit Goel. 2007. "Architecting Solvers for Sat Modulo Theories: Nelson-Oppen with Dpll." In *International Symposium on Frontiers of Combining Systems*, 1–27. Springer.

Manna, Zohar, and Calogero G. Zarba. 2003. "Combining Decision Procedures."

In *Formal Methods at the Crossroads. From Panacea to Foundational Support, 10th Anniversary Colloquium of Unu/Iist, the International Institute for Software Technology of the United Nations University, Lisbon, Portugal, March 18-20, 2002, Revised Papers*, 2757:381–422. Lecture Notes in Computer Science. Springer.

Meseguer, José. 2012. "Twenty Years of Rewriting Logic." *The Journal of Logic and Algebraic Programming* 81 (7): 721–81. `https://doi.org/https://doi.org/10.1016/j.jlap.2012.06.003`.

Meseguer, José. 2013. "Set Theory and Algebra in Computer Science." 2013. `https://courses.engr.illinois.edu/cs476/fa2017/readings/meseguer-set-theory-algebra-computer-science.pdf`.

———. 2016. "Variant-Based Satisfiability in Initial Algebras." In *Formal Techniques for Safety-Critical Systems*, edited by Cyrille Artho and Peter Csaba Ölveczky, 3–34. Cham: Springer International Publishing.

———. 2017a. "CS 476 - Fall 2017." 2017. `https://courses.engr.illinois.edu/cs476/fa2017/`.

———. 2017b. "CS 576 - Spring 2017." 2017. `https://courses.engr.illinois.edu/cs576/sp2017/`.

Nelson, Greg, and Derek C. Oppen. 1979. "Simplification by Cooperating Decision Procedures." *ACM Trans. Program. Lang. Syst.* 1 (2). New York, NY, USA: ACM: 245–57. `https://doi.org/10.1145/357073.357079`.

Nieuwenhuis, Robert, Albert Oliveras, and Cesare Tinelli. 2006. "Solving SAT and SAT Modulo Theories: from an Abstract Davis-Putnam-Logemann-Loveland Procedure to DPLL(T)." *Journal of the ACM* 53 (6): 937–77.

Ranise, Silvio, Christophe Ringeissen, and Calogero G. Zarba. 2005. "Combining Data Structures with Nonstably Infinite Theories Using Many-Sorted Logic." In *Frontiers of Combining Systems*, edited by Bernhard Gramlich, 48–64. Berlin, Heidelberg: Springer Berlin Heidelberg.

Şerbanuţă, Traian Florin, and Grigore Roşu. 2010. "K-Maude: A Rewriting Based Tool for Semantics of Programming Languages." In *Proceedings of the 8th International Workshop on Rewriting Logic and Its Applications (Wrla'10)*, 104–22. Springer. `https://doi.org/http://dx.doi.org/10.1007/978-3-642-16310-4_8`.

Shostak, Robert E. 1979. "A Practical Decision Procedure for Arithmetic with Function Symbols." *J. ACM* 26 (2). New York, NY, USA: ACM: 351–60. `https://doi.org/10.1145/322123.322137`.

———. 1984. "Deciding Combinations of Theories." *J. ACM* 31 (1). New York, NY, USA: ACM: 1–12. `https://doi.org/10.1145/2422.322411`.

Skeirik, Stephen, and Jose Meseguer. 2016. "Metalevel Algorithms for Variant Satisfiability."

Suzuki, Norihisa, and David Jefferson. 1980. "Verification Decidability of Presburger Array Programs." *J. ACM* 27 (1). New York, NY, USA: ACM: 191–205. `https://doi.org/10.1145/322169.322185`.

Tinelli, Cesare, and Calogero G Zarba. 2004. "Combining Decision Procedures for Sorted Theories." In *European Workshop on Logics in Artificial Intelligence*, 641–53. Springer.

Tinelli, Cesare, and Calogero G. Zarba. 2003. "Combining Non-Stably Infinite

Theories." *Electronic Notes in Theoretical Computer Science* 86 (1): 35–48. `https://doi.org/https://doi.org/10.1016/S1571-0661(04)80651-0`.