Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 198 (2008) 37–49
www.elsevier.com/locate/entcs

Model-based Theory Combination

Leonardo de Moura[1] Nikolaj Bjørner[2]

Microsoft Research, One Microsoft Way, Redmond, WA, 98074, USA

Abstract

Traditional methods for combining theory solvers rely on capabilities of the solvers to produce all implied equalities or a pre-processing step that introduces additional literals into the search space. This paper introduces a combination method that incrementally reconciles models maintained by each theory. We evaluate the practicality and efficiency of this approach.

Keywords: Theory Combination, Decision Procedures, SMT.

1 Introduction

A core problem of Satisfiability Modulo Theories is combining separate theory solvers for theories $\mathcal{T}_1$ and $\mathcal{T}_2$ to a combined solver for the union $\mathcal{T}_1 \cup \mathcal{T}_2$. The Nelson-Oppen combination method identifies sufficient conditions for combining two theories over disjoint signatures: only (disjunctions of) equalities over shared variables that are implied by one of the theories need to be communicated. Most existing implementations and optimizations of this method seek to efficiently implement the trigger:

$$\textbf{if } \mathcal{T}_i \cup \Gamma_i \models u \simeq v \textbf{ then } \text{propagate } u \simeq v,$$

to exhaustively enumerating all implied (disjunctions of) equalities for a theory $\mathcal{T}_i$ and constraints Γ_i that are asserted in its context. Another method [4] to obtain completeness is by enumerating equalities corresponding to the

[1] Email: leonardo@microsoft.comu
[2] Email: nbjorner@microsoft.com

1571-0661/$ – see front matter
doi:10.1016/j.entcs.2008.04.079

cross-product of all shared variable pairs and use the SAT solver for non-deterministically choosing a partition based on the cross-product. Common to these methods is that they are pessimistic about which equalities are propagated. A more optimistic approach is by inspecting a candidate model $\mathcal{M}_i$ for one of the theories $\mathcal{T}_i$ and propagate all equalities implied by the candidate model, hedging that other theories will agree. If not, use backtracking to fix the model. Thus, Model-based Theory Combination is based on a trigger of the form:

$$\textbf{if } \mathcal{M}_i \models \mathcal{T}_i \cup \Gamma_i \cup \{u \simeq v\} \textbf{ then } \text{propagate } u \simeq v \ .$$

The rationale for Model-based Theory Combination is practical: It tends to be much cheaper to enumerate equalities that are implied in a particular model than of all models; the number of inter-theory equalities that really matter is small in practice (intra-theory equality propagation on the other hand does matter); backtracking is relatively cheap with modern DPLL solvers; and finally, one may limit the number of equalities implied by a model by model mutation.

2 Background

A *signature* Σ is a set of function and predicate symbols. Each symbol is associated with a nonnegative integer, called the *arity*. If $arity(g) = 0$, then g is a *constant symbol*. We assume that the binary equality predicate $\simeq$ to be always present in any signature Σ. We use the standard notion of Σ-*structure* $\mathcal{M}$, that is, a support set endowed with an arity-matching interpretation of the function and predicate symbols from Σ. We use $f^{\mathcal{M}}$ ($p^{\mathcal{M}}$) to denote the interpretation of the function symbol f (predicate symbol p) in the structure $\mathcal{M}$. The truth of a Σ-formula in $\mathcal{M}$ is defined in the standard ways. A formula ϕ is satisfiable in $\mathcal{M}$ iff its *existential closure* is true in $\mathcal{M}$. In this case, we say $\mathcal{M}$ is a *model* for ϕ. A *sentence* is a first-order formula with no free variables. A (first-order) *theory* $\mathcal{T}$ over a signature Σ is a set of (deductively closed) sentences over Σ. We say two theories $\mathcal{T}_1$ and $\mathcal{T}_2$ have disjoint signatures when $\Sigma_1 \cap \Sigma_2 = \{\simeq\}$. A theory $\mathcal{T}$ is *stably infinite* if every satisfiable quantifier free formula is satisfiable in an infinite model. A theory $\mathcal{T}$ is *convex* [15] iff for all finite sets Γ of literals and for all non-empty disjunctions $\bigvee_{i\in I} u_i \simeq v_i$ of variables, $\Gamma \models_{\mathcal{T}} \bigvee_{i\in I} u_i \simeq v_i$ iff $\Gamma \models_{\mathcal{T}} u_i \simeq v_i$ for some $i \in I$. Intuitively, a theory is convex if for every satisfiable set of literals there is a model where variables not implied to be equal have a distinct interpretation. For example, linear integer arithmetic is not convex, because the set of literals $\{0 \leq x_1 \leq 1, 0 \leq x_2 \leq 1, 0 \leq x_3 \leq 1\}$ is satisfiable, no equality $x_i \simeq x_j$ for $i \neq j$ is implied, but there is no model where x_1, x_2 and

x_3 are all distinct.

2.1 Nelson-Oppen combination method

Nelson-Oppen (NO) combination method [12] provides a rather simple solution for the theory combination problem for theories that are stably infinite and have disjoint theories. More formally, let $\mathcal{T}_1$ and $\mathcal{T}_2$ be consistent, stably infinite theories over disjoint (countable) signatures. Assuming satisfiability of conjunction of literals can be decided in $\mathcal{O}(\mathcal{T}_1(n))$ and $\mathcal{O}(\mathcal{T}_2(n))$ time respectively. Then,

(i) The combined theory $\mathcal{T}$ is consistent and stably infinite.

(ii) Satisfiability of quantifier free conjunction of literals in $\mathcal{T}$ can be decided in $\mathcal{O}(2^{n^2} \times (\mathcal{T}_1(n) + \mathcal{T}_2(n)))$.

(iii) If $\mathcal{T}_1$ and $\mathcal{T}_2$ are convex, then so is $\mathcal{T}$ and satisfiability in $\mathcal{T}$ is in $\mathcal{O}(n^4 \times (\mathcal{T}_1(n) + \mathcal{T}_2(n)))$.

Let Γ be a set of literals over $\Sigma_1 \cup \Sigma_2$. Then, the non-deterministic NO combination method can be described in the following way. First, a satisfiability preserving transformation called *purification* is used to transform Γ into $\Gamma_1 \wedge \Gamma_2$, such that, the symbols from Γ_i are in $\Sigma_i^{\underline{a}}$, where $\Sigma_i^{\underline{a}} = \Sigma_i \cup \underline{a}$, and $\underline{a}$ $(= \mathcal{V}(\Gamma_1) \cap \mathcal{V}(\Gamma_2))$ denotes the set of shared variables between Γ_1 and Γ_2. Then, a partition of $\underline{a}$ into disjoint subsets is guessed and is expressed as a conjunction of literals ϕ. For example, the partition $\{x_1\}, \{x_2, x_3\}, \{x_4\}$ is represented as $x_1 \not\simeq x_2, x_1 \not\simeq x_4, x_2 \not\simeq x_4, x_2 \simeq x_3$. Then, the individual procedures are used to decide whether $\Gamma_i \wedge \phi$ is satisfiable. The combined procedure returns unsatisfiable if one of the procedures returns unsatisfiable.

For convex theories, instead of *guessing*, one can *deduce* the equalities to be shared. The key idea is to propagate $x \simeq y$ to Γ_2 whenever $\mathcal{T}_1 \cup \Gamma_1 \models x \simeq y$, and vice-versa. This process is repeated until no further equations can be propagated. Then, the individual procedures are used to decide whether Γ_i is satisfiable. Sharing equalities in this case is sufficient, because a theory $\mathcal{T}_1$ can assume that $x^{\mathcal{M}_2} \neq y^{\mathcal{M}_2}$ whenever $x \simeq y$ is not implied by $\mathcal{T}_2$ and vice versa. So, for convex theories, there is an efficient way to construct a partition of the set of shared variables.

3 Related Work

3.1 Convex theories only

For convex theories it is sufficient to propagate all implied equalities between shared variables. For instance, theories that admit canonizers solve equality

propagation by rewriting. In general, a theory $\mathcal{T}_i$ is canonizing (as coined in [18]) if it admits a function $\_\downarrow_{\_}$, such that:

$$\mathcal{T}_i \cup \Gamma_i \models s \simeq t \quad \textbf{iff} \quad s\downarrow_{\Gamma_i} = t\downarrow_{\Gamma_i} \quad .$$

Linear rational arithmetic is convex, procedures based on the Fourier-Motzkin algorithm produce all implied equalities in a straight-forward way, but the procedure may require exponential space. For procedures based on the Simplex algorithm, all implied equalities can be deduced using the approaches described in [16,7].

The main disadvantage of these approaches is poor performance (either of the solver or of the equality propagation) and the inability to deal with non-convex theories.

3.2 DTC: Delayed Theory Combination

Several SMT solvers [3] use the underlying SAT solver to guess a partition of $\mathcal{V}(\Gamma_1) \cap \mathcal{V}(\Gamma_2)$, the idea is to create a literal $u \simeq v$ for every pair of shared variables u and v. One may be concerned that guessing a partition would be exponentially more expensive than deriving it when the theories are convex. However, as shown in [6], back-jumping and lemma learning allow *simulating* the standard Nelson-Oppen combination method: equalities that are implied by a theory, once learned are not flipped.

The obvious disadvantage of this approach is that the number of additional equality literals is quadratic in the number of shared variables. There is an additional assumption that may be tool specific to MathSat, but are pervasive in the results from [6]: all literals used by the SAT solver must be present in the input to the SAT solver. At the current time of writing CVC3 [2], Verifun [10], Yices [9], and our tool Z3 all support dynamically added literals.

Our approach, presented later, appeals to an SMT solver that allows introducing literals on the fly. If this is not possible, the approach boils down to a branching heuristic on top of DTC.

3.3 Ackermannization

When combining two theories $\mathcal{T}_1$ and $\mathcal{T}_2$, where $\mathcal{T}_1$ is the theory of uninterpreted functions, one can eliminate $\mathcal{T}_1$ by creating all ground instances of Leibniz's rule [1]:

$$\bigwedge n_i \simeq m_i \;\rightarrow\; f(\overline{n}) \simeq f(\overline{m}) \tag{1}$$

Thus, we can eliminate a function symbol f from a set of formulas F by applying the procedure ackermannize:

ackermannize(f, F) :
  foreach $f(\overline{n}) \in F$ **where** $\overline{n}$ do not contain f
    create a fresh constant $a_{f(\overline{n})}$
    replace $f(\overline{n})$ by $a_{f(\overline{n})}$ in F
  foreach $a_{f(\overline{n})}, a_{f(\overline{m})}$
    add the clause $\bigwedge n_i \simeq m_i \;\rightarrow\; a_{f(\overline{n})} \simeq a_{f(\overline{m})}$ to F

A partial Ackermann's reduction heuristic is proposed and investigated in [5]. Functions are only eliminated when the number of distinct occurrences is smaller than the set of shared variables in the function arguments.

Ackermannization has the same disadvantage as DTC in that the number of additional literals is quadratic in the size of the input. It is furthermore problematic to use Ackermannization in the context of several theories and when combining SMT solvers with quantifier instantiation.

3.3.1 Dynamic Ackermannization

Congruence closure algorithms for deduction in equational theories are ubiquitous. Efficient and incremental congruence closure based procedures are described in [7,13]. However, these algorithms miss the following propagation rule:

$$f(\overline{n}) \not\simeq f(\overline{m}) \implies \bigvee n_i \not\simeq m_i \; . \tag{2}$$

This propagation rule (which is a contrapositive of (1)) has a dramatic performance benefit in some benchmarks, and Ackermann's reduction gives this rule for free. For example, the following simple formula takes $\mathcal{O}(2^N)$ time to be solved using the algorithms presented in [7,13]. In contrast the formula can be solved in polynomial time if either $\mathcal{O}(N^2)$ axioms are added up front by ackermannize or the above propagation rule is used resulting in "only" $\mathcal{O}(N)$ space and time overhead.

$$\bigwedge_{i=1}^{N} (p_i \vee x_i \simeq v_0) \;\wedge\; (\neg p_i \vee x_i \simeq v_1) \;\wedge\; (p_i \vee y_i \simeq v_0) \;\wedge\; (\neg p_i \vee y_i \simeq v_1),$$
$$f(x_N, \ldots, f(x_2, x_1) \ldots) \not\simeq f(y_N, \ldots, f(y_2, y_1) \ldots) \tag{3}$$

This performance problem reflects a limitation in the current congruence closure algorithms used in SMT solvers, and it is not related with the theory combination problem. In fact, the formula above uses only one theory. In [9], an approach, called *Dynamic Ackermannization*, is proposed to cope with this problem. There, clauses corresponding to Ackermann's reduction are added when a congruence rule participates in a conflict.

4 Model-based Theory Combination

Our approach minimizes the number of produced shared equalities. It is based on the fact that, in practice, the number of local inconsistencies is much bigger than the number of global (cross theory) inconsistencies. It works for convex and non-convex theories alike.

(i) Each theory $\mathcal{T}_i$ maintains a model $\mathcal{M}_i$ for Γ_i, or at times only for a subset of Γ_i.

(ii) From time to time, if $u^{\mathcal{M}_i} = v^{\mathcal{M}_i}$, then the theory creates the case-split $u \simeq v$, the positive case is tried first.

(iii) At the discresion of the theory solver, change a model $\mathcal{M}_i$ to $\mathcal{M}_i'$ to satisfy newly assigned literals, or to imply fewer equalities.

It is fairly straight-forward to integrate this approach in a DPLL($\mathcal{T}$) framework, that we rename DPLL($\mathcal{T}_{\mathcal{M}}$), as the search is now model-guided. Borrowing notation from [11,14], the relevant new rules are presented in Fig. 1. The full set of rules are repeated from [14] in Fig. A.1. The transition rules modify triplets of the form $\boldsymbol{\mathcal{M}}, \Gamma \parallel F$, where $\boldsymbol{\mathcal{M}}$ is a set of models for theories $\mathcal{T}_1, \ldots, \mathcal{T}_n$, Γ is a set of asserted literals, and F is a set of clauses. The rule $\mathcal{M}$-Propagate creates a fresh equality literal $(u \simeq v)^d$ when a model associated with one of the theories imply it, but the equality is not present in the context Γ. The equality literal is pushed on Γ, thus propagating the equality to all theories sharing variables u and v. The tag d on the literal indicates that the literal may be negated during backtracking. The $\mathcal{M}$-Mutate rule allows changing models during backtracking search. For instance, after applying Decide, a newly assigned literal ℓ^d may not be satisfied in the existing models. We do not need to specify when $\mathcal{M}$-Mutate is applied. In particular, theory solvers are not required to maintain models for their contexts at all times during a search. Models are only required when other case splits have been attempted. For example, when using linear programming for an integer linear programming problem, a simplex tableau may choose to delay introducing Gomory cuts to obtain an integer interpretation until other constraints have been propagated.

We use the following optimizations to minimize even further the number of necessary case splits. Let $R_{\mathcal{M}}$ be an equivalence relation on $\mathcal{V}$ such that $R_{\mathcal{M}}(u, v)$ iff $u^{\mathcal{M}} = v^{\mathcal{M}}$. Let $\mathit{classes}(R)$ be the set of equivalence classes induced by R.

(i) Opportunistic equality propagation: Equalities that can be inferred without additional expense to the theory solver are always propagated eagerly. Section 5 gives an example of opportunistic equality propagation.

$\mathcal{M}$-Propagate

$$\boldsymbol{\mathcal{M}}, \Gamma \parallel F \quad \Longrightarrow \quad \boldsymbol{\mathcal{M}}, \Gamma(u \simeq v)^d \parallel F \text{ if } \begin{cases} u, v \in \mathcal{V},\ (u \simeq v) \notin \mathcal{L} \\ u^{\mathcal{M}_i} = v^{\mathcal{M}_i} \\ \text{add } (u \simeq v) \text{ to } \mathcal{L} \end{cases}$$

$\mathcal{M}$-Mutate

$$\boldsymbol{\mathcal{M}}, \Gamma \parallel F \quad \Longrightarrow \quad \boldsymbol{\mathcal{M}}', \Gamma \parallel F \qquad \text{if } \left\{ \boldsymbol{\mathcal{M}}' \text{ is some variant of } \boldsymbol{\mathcal{M}} \right.$$

Fig. 1. Model-based propagation

(ii) Postponing model-based equality propagation: we delay applying the rule $\mathcal{M}$-Propagate until case splits on already existing literals have been performed.

(iii) Model mutators, the idea is to use a function $\delta(\mathcal{M}_k)$ that returns a more *diverse* model. More formally, $|classes(R_{\mathcal{M}_k})| \leq |classes(R_{\delta(\mathcal{M}_k)})|$

5 Simplex: An example model-producing theory solver

Following [8], a theory solver for linear arithmetic, and integer linear arithmetic, can be based on a Simplex Tableau of the form:

$$x_i \simeq \sum_{x_j \in \mathcal{N}} a_{ij} x_j \quad x_i \in \mathcal{B},$$

where $\mathcal{B}$ and $\mathcal{N}$ denote the set of basic and nonbasic variables, respectively. In addition to this tableau, the solver state stores upper and lower bounds l_i and u_i for every variable x_i and a mapping β that assigns a rational value $\beta(x_i)$ to every variable x_i. The bounds on nonbasic variables are always satisfied by β, that is, the following invariant is maintained

$$\forall x_j \in \mathcal{N}, \quad l_j \leq \beta(x_j) \leq u_j.$$

Bounds constraints for basic variables are not necessarily satisfied by β, so for instance, it may be the case that $l_i > \beta(x_i)$ for some basic variable x_i, but pivoting steps can be used to fix bounds violations, or detect an infeasible tableau. We hope it does not come as a total surprise that the current model for the simplex solver is given by β. Enumerating the equalities implied by β is simple: enumerate all the values of $\beta(x_i)$, inserting each value into a hash table. The expected time of enumerating all equalities is then $\mathcal{O}(|\mathcal{B} \cup \mathcal{N}|)$.

5.1 *Opportunistic equality propagation*

In a Simplex tableau, some equalities can be inferred using the following inexpensive rules. We say that a variable x_i is *fixed* iff $l_i = u_i$, a linear polynomial $\sum_{x_j \in \mathcal{V}} a_{ij} x_j$ is fixed iff for every $x_j \in \mathcal{V}$, x_j is fixed or $a_{ij} = 0$. Given a linear polynomial $P = \sum_{x_j \in \mathcal{V}} a_{ij} x_j$, we use $\beta(P)$ to denote $\sum_{x_j \in \mathcal{V}} a_{ij} \beta(x_j)$.

FixedEq

$$l_i \leq x_i \leq u_i, \quad l_j \leq x_j \leq u_j \quad \Longrightarrow \quad x_i \simeq x_j \quad \textbf{if} \quad l_i = u_i = l_j = u_j$$

EqRow

$$x_i \simeq x_j + P \quad \Longrightarrow \quad x_i \simeq x_j \quad \textbf{if} \quad P \text{ is fixed, and } \beta(P) = 0$$

EqOffsetRows

$$\left.\begin{array}{l} x_i \simeq x_k + P_1 \\ x_j \simeq x_k + P_2 \end{array}\right. \Longrightarrow \quad x_i \simeq x_j \quad \textbf{if} \quad \left\{\begin{array}{l} P_1 \text{ and } P_2 \text{ are fixed,} \\ \text{and } \beta(P_1) = \beta(P_2) \end{array}\right.$$

EqRows

$$\left.\begin{array}{l} x_i \simeq P + P_1 \\ x_j \simeq P + P_2 \end{array}\right. \Longrightarrow \quad x_i \simeq x_j \quad \textbf{if} \quad \left\{\begin{array}{l} P_1 \text{ and } P_2 \text{ are fixed,} \\ \text{and } \beta(P_1) = \beta(P_2) \end{array}\right.$$

The first rule can be implemented using a mapping from values to fixed variables, the second rule can be easily checked when a row is updated during a pivoting step. The rule EqOffsetRows is a simpler and less expensive version of EqRows. It can be implemented using a mapping $(x_k, v) \mapsto x_i$, where x_k and x_i are variables, and v is a value. In our implementation, the first three rules are eagerly applied, and the last one is only applied before $\mathcal{M}$-Propagate. We also aggressively remove fixed variables from the basis.

These rules can miss some implied equalities. For instance, from the constraints (4), the rules detect the implied equality $z \simeq w$, but miss the equality $x \simeq y$, because z is not a fixed variable. Fortunately, the bound propagation technique described in [8] can be used imply the bound $0 \leq w$, making w a fixed variable, and enabling the application of the rule EqRow.

$$x \simeq y + w + s, \quad z \simeq w + s, \quad 0 \leq z, \quad w \leq 0, \quad 0 \leq s \leq 0 \tag{4}$$

5.2 *Mutation using* freedom intervals

The *freedom* of a non-basic variable x_j is defined as the interval $[L_j, U_j]$, where:

$$L_j = \max\left(\left\{\beta(x_j) + \frac{l_i - \beta(x_i)}{a_{ij}} \mid a_{ij} > 0\right\} \cup \left\{\beta(x_j) + \frac{u_i - \beta(x_i)}{a_{ij}} \mid a_{ij} < 0\right\} \cup \{l_j\}\right)$$

$$U_j = \min\left(\left\{\beta(x_j) + \frac{u_i - \beta(x_i)}{a_{ij}} \mid a_{ij} > 0\right\} \cup \left\{\beta(x_j) + \frac{l_i - \beta(x_i)}{a_{ij}} \mid a_{ij} < 0\right\} \cup \{u_j\}\right)$$

Intuitively, if β satisfies all rows and bound constraints, then β will also satisfy them after executing $\mathsf{update}(x_j, v)$ for any value v in the interval $[L_j, U_j]$, where the update procedure is defined as:

$\mathsf{update}(x_j, v)$

  foreach $x_i \in \mathcal{B}$, $\beta(x_i) := \beta(x_i) + a_{ij}(v - \beta(x_j))$

  $\beta(x_j) := v$

Freedom intervals can be used to produce a more diverse β without performing any pivoting operation. A simple *greedy* heuristic seems to be quite effective: for each non-basic variable x_j, execute $\mathsf{update}(x_j, v)$, if there is a value $v \in [L_j, U_j]$ such that $|\mathit{classes}(R_\beta)| < |\mathit{classes}(R_{\beta'})|$, where β' denotes β after the update operation.

6 Experimental Evaluation

The experiments were conducted using a 32bit Pentium 4 processor running at 3.6Ghz, 2Gb of memory, and 2Mb of cache. The timeout was set to 10 minutes. We compared our approach against other SMT solvers and against different strategies within our solver Z3. We used the same benchmarks used in [5]. The benchmarks were translated to the SMT-LIB format, and are available for download in our website[3]. We also included, for $N \in [1, 99]$ the examples from (3) and the following simple family of satisfiable formulas in the comparison for $N \in [2, 99]$:

$$\varphi = \bigwedge_{i=1}^{N} f(x_i) \geq 0 \ \wedge \ x_i \geq 0 \ \wedge \ x_i \not\simeq x_{i+1} \tag{5}$$

All benchmarks but the Ackermann suite use the theories of uninterpreted functions and linear arithmetic. Tables 1 and 2 summarize the results obtained in our experiments. Each cell has the accumulated time, in seconds, used by each solver to solve a family of benchmarks. It does not include the time spent in instances where the solver produced the *unknown* result. A solver is considered to have produced the *unknown* result when it times out

[3] http://research.microsoft.com/~leonardo/SMT07

	#	MathSAT	MathSAT-dtc	Yices	Z3
EufLaArithmetic	52	1851.50 (11)	785.87 (1)	**10.45**	17.34
Hash	199	520.90	19.39	11.48	**6.54**
Wisa	256	886.36 (1)	6916.18	4.37	**2.78**
RandomCoupled	400	517.05	518.15	9516.11 (51)	**56.16**
RandomDecoupled	500	11989.60 (1)	97.07	19362.40 (51)	**41.95**
Simple (5)	98	1366.33	7053.98 (29)	2328.63 (53)	**1.00**
Ackermann (3)	99	228.49 (82)	344.00 (82)	2.99	**1.72**
Total	1604	17360.23 (95)	15734.64 (112)	31236.43 (155)	**127.49**

Table 1
Experimental results

or crashes. The number of *unknown* results is displayed using parenthesis. MathSAT-dtc denotes the MathSAT solver with the command line option `-DTC` that forces it to use Delayed Theory Combination. We used Yices version 1.0.8 in the experiments. Yices is also based on DTC, but the shared equalities are lazily generated, and it uses a filtering mechanism to avoid the generation of unnecessary shared equalities [9]. Yices and Z3 implement Dynamic Ackermannization suggested in [9]. Six different versions of our Z3 solver were used: Z3-dtc uses delayed theory combination and the additional equalities between shared variables are eagerly generated; Z3-dtc* is similar to Z3-dtc but uses the current model to implement a branching heuristic for the generated equalities; Z3-ack uses Ackermann's reduction as a pre-processing step; Z3-neq, Z3-ndack and Z3 all use Model Based Theory Combination, but Z3-neq does not use opportunistic equality propagation, and Z3-ndack does not use Dynamic Ackermannization. Notice that our implementation of Z3-dtc does not include several optimizations that may be useful for a DTC framework. It does not take advantage of theory propagation for arithmetic.

The benchmarks in the EufLaArithmetic family are trivial if the linear arithmetic solver performs some form of opportunistic equality propagation. The Simple family described above was used to demonstrate that DTC is not robust. Yices performs poorly on most satisfiable instances in the RandomCoupled and RandomDecoupled families.

Model based theory combination seems to be more robust than DTC or Ackermann's reduction. In these benchmarks, the average number of shared equalities propagated by Z3 using $\mathcal{M}$-Propagate was 2.46. The maximum was 57.

	Z3-dtc	Z3-dtc*	Z3-ack	Z3-neq	Z3-ndack	Z3
EufLaArithmetic	796.71 (11)	2830.38 (4)	1094.47 (1)	786.15	**11.56**	17.34
Hash	310.10	305.75	23.68	**5.89**	6.02	6.54
Wisa	364.71	385.06	12.31	4.89	**2.40**	2.78
RandomCoupled	8122.45 (166)	12451.82 (103)	101.24	56.45	56.65	**56.16**
RandomDecoupled	12421.30 (85)	15316.60 (71)	56.54	51.23	48.39	**41.95**
Simple (5)	7.26	7.34	33.89	**0.45**	1.00	1.00
Ackermann (3)	728.22 (77)	733.58 (77)	37.99	1.74	874.21 (77)	**1.72**
Total	22750.75 (339)	32030.53 (255)	1360.12 (1)	906.78	1000.23 (77)	**127.49**

Table 2
Experimental results (only Z3)

7 Conclusions

This paper introduced a new approach for dealing with equality propagation in the context of convex theories where equality deduction is expensive and more generally, in the context of non-convex theories. Both in theory, and as we validated experimentally, the approach solves a number of practical deficiencies with other known solutions to integrating theories.

Model-based Theory Combination requires that a decidable theory maintains a notion of a model that supports efficiently answering queries of the form $u^{\mathcal{M}} = v^{\mathcal{M}}$.

7.1 On the significance of Ackermann's reduction

The Wisa benchmark set is used in [5] to illustrate the usefulness of Ackermann's reduction in contrast with DTC. With Model-based Theory Combination, Ackermann's reduction does not help on these set of benchmarks. We therefore believe they reflect more the problems with DTC than the advantages of Ackermann's reduction. On the other hand, the synthetic Ackermann benchmarks, one can observe the utility of the reduction. Dynamic Ackermann reduction is not without an overhead: new literals are added to the search space, and the new literals cause T-Propagate to spend additional overhead of walking use-lists. Future work includes investigating whether it is a practical advantage to build in the propagation directly into a congruence closure algorithm.

7.2 Are theories amenable to Model-based combinations?

Our main example of a model producing theory solver was a classical Simplex solver. We are experimenting with adding model-based solvers to other theories, and we hope to be reporting on our findings in future work.

8 Acknowledgment

The authors would like to thank Alberto Griggio for providing the latest version of MathSAT and adding a command line option for enabling Delayed Theory Combination.

References

[1] Ackermann, W., *Solvable cases of the decision problem*, Studies in Logic and the Foundation of Mathematics (1954).

[2] Barrett, C. and C. Tinelli, *CVC3*, in: W. Damm and H. Hermanns, editors, *CAV'07, Berlin, Germany*, LNCS **4590** (2007).

[3] Bozzano, M., R. Bruttomesso, A. Cimatti, T. A. Junttila, S. Ranise, P. van Rossum and R. Sebastiani, *Efficient theory combination via boolean search.*, Inf. Comput. **204** (2006), pp. 1493–1525.

[4] Bozzano, M., R. Bruttomesso, A. Cimatti, T. A. Junttila, P. van Rossum, S. Schulz and R. Sebastiani, *The MathSAT 3 System*, in: R. Nieuwenhuis, editor, *CADE*, LNCS **3632** (2005), pp. 315–321.

[5] Bruttomesso, R., A. Cimatti, A. Franzén, A. Griggio, A. Santuari and R. Sebastiani, *To Ackermann-ize or Not to Ackermann-ize? On Efficiently Handling Uninterpreted Function Symbols in UF(*E*)*, in: *LPAR*, 2006, pp. 557–571.

[6] Bruttomesso, R., A. Cimatti, A. Franzén, A. Griggio and R. Sebastiani, *Delayed Theory Combination vs. Nelson-Oppen for Satisfiability Modulo Theories: A Comparative Analysis.*, in: *LPAR*, 2006, pp. 527–541.

[7] Detlefs, D., G. Nelson and J. B. Saxe, *Simplify: a theorem prover for program checking*, J. ACM **52** (2005), pp. 365–473.

[8] Dutertre, B. and L. de Moura, *A Fast Linear-Arithmetic Solver for DPLL(T)*, in: *CAV'06*, LNCS **4144** (2006), pp. 81–94.

[9] Dutertre, B. and L. de Moura, *The Yices SMT Solver*, http://yices.csl.sri.com/tool-paper.pdf (2006).

[10] Flanagan, C., R. Joshi and J. B. Saxe, *An explicating theorem prover for quantified formulas*, Technical Report HPL-2004-199, HP Laboratories, Palo Alto (2004).

[11] Ganzinger, H., G. Hagen, R. Nieuwenhuis, A. Oliveras and C. Tinelli, *DPLL(T): Fast decision procedures.*, in: *CAV'04*, LNCS **3144**, 2004, pp. 175–188.

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

[13] Nieuwenhuis, R. and A. Oliveras, *Fast Congruence Closure and Extensions*, Inf. Comput. **2005** (2007), pp. 557–580.

[14] Nieuwenhuis, R., A. Oliveras and C. Tinelli, *Solving SAT and SAT Modulo Theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(T).*, J. ACM **53** (2006), pp. 937–977.

[15] Oppen, D. C., *Complexity, convexity and combinations of theories.*, Theor. Comput. Sci. **12** (1980), pp. 291–302.

[16] Rueß, H. and N. Shankar, *Solving linear arithmetic constraints*, Technical Report SRI-CSL-04-01, SRI International (2004).

[17] Sheini, H. M. and K. A. Sakallah, *SMT(LU): a step toward scalability in system verification.*, in: S. Hassoun, editor, *ICCAD* (2006), pp. 844–851.

[18] Shostak, R. E., *Deciding combinations of theories.*, J. ACM **31** (1984), pp. 1–12.

A A presentation of DPLL($\mathcal{T}_{\mathcal{M}}$)

Fig. A.1 repeats from [14] the presentation of DPLL($\mathcal{T}$) as an abstract transition system. We have added $\boldsymbol{\mathcal{M}}$ to each sequent to emphasize that the current state during proof-search also carries a model. In contrast to [14], we use a set $\mathcal{L}$ of literals to choose case split candidates from. We assume $\mathcal{L}$ is a super-set of the literals occurring in the set of clauses F. Besides the fresh equality literals introduced by the rule $\mathcal{M}$-Propagate, non-convex theory implementations, such as an integer linear solver may introduce fresh literals as a side-effect of performing branch-and bound search.

UnitPropagate

$$\boldsymbol{\mathcal{M}}, \Gamma \parallel F, C \vee \ell \implies \boldsymbol{\mathcal{M}}, \Gamma\ell \parallel F, C \vee \ell \quad \textbf{if} \quad \begin{cases} \Gamma \models \neg C \\ \ell \text{ is undefined in } \Gamma \end{cases}$$

Decide

$$\boldsymbol{\mathcal{M}}, \Gamma \parallel F \implies \boldsymbol{\mathcal{M}}, \Gamma\ell^d \parallel F \quad \textbf{if} \quad \begin{cases} \ell \text{ occurs in } \mathcal{L} \\ \ell \text{ is undefined in } \Gamma \end{cases}$$

Fail

$$\boldsymbol{\mathcal{M}}, \Gamma \parallel F, C \implies \mathit{fail} \quad \textbf{if} \quad \begin{cases} \Gamma \models \neg C \\ \Gamma \text{ contains no decision variables} \end{cases}$$

Backjump

$$\boldsymbol{\mathcal{M}}, \Gamma\ell^d\Gamma' \parallel F, C \implies \boldsymbol{\mathcal{M}}, \Gamma\ell' \parallel F, C \quad \textbf{if} \quad \begin{cases} \Gamma\ell^d\Gamma' \models \neg C, \\ \text{there is a clause } C' \vee \ell' \text{ such that} \\ \quad F, C \models C' \vee \ell' \text{ and } \Gamma \models \neg C'; \\ \quad \ell' \text{ is undefined in } \Gamma; \\ \quad \ell' \text{ or } \neg\ell' \text{ occurs in } F \text{ or in } \ell^d\Gamma' \end{cases}$$

T-Propagate

$$\boldsymbol{\mathcal{M}}, \Gamma \parallel F \implies \boldsymbol{\mathcal{M}}, \Gamma\ell \parallel F \quad \textbf{if} \quad \begin{cases} \Gamma \models_{\mathcal{T}} \ell \\ \ell \text{ occurs in } \mathcal{L} \\ \ell \text{ is undefined in } \Gamma \end{cases}$$

Learn

$$\boldsymbol{\mathcal{M}}, \Gamma \parallel F \implies \boldsymbol{\mathcal{M}}, \Gamma \parallel F, C \quad \textbf{if} \quad \begin{cases} \text{all atoms of } C \text{ occur in } \mathcal{L} \\ F \models_{\mathcal{T}} C \end{cases}$$

Forget

$$\boldsymbol{\mathcal{M}}, \Gamma \parallel F, C \implies \boldsymbol{\mathcal{M}}, \Gamma \parallel F, C \quad \textbf{if} \quad \{ F \models_{\mathcal{T}} C$$

Restart

$$\boldsymbol{\mathcal{M}}, \Gamma \parallel F \implies \boldsymbol{\mathcal{M}}, \emptyset \parallel F$$

Fig. A.1. DPLL with exhaustive theory propagation