Handling Common Transitive Relations in First-Order Automated Reasoning

Koen Claessen Ann Lilliestr\"om

Chalmers University of Technology
{koen,annl}@chalmers.se

Abstract

We present a number of alternative ways of handling transitive binary relations that commonly occur in first-order problems, in particular *equivalence relations*, *total orders*, and *reflexive, transitive relations*. We show how such relations can be discovered syntactically in an input theory. We experimentally evaluate different treatments on problems from the TPTP, using resolution-based reasoning tools as well as instance-based tools. Our conclusions are that (1) it is beneficial to consider different treatments of binary relations as a user, and that (2) reasoning tools could benefit from using a preprocessor or even built-in support for certain binary relations.

1 Introduction

Most automated reasoning tools for first-order logic have some kind of built-in support for reasoning about equality. Equality is one of the most common binary relations, and there are great performance benefits from providing built-in support for equality. Together, these two advantages by far outweigh the cost of implementation.

Other common concepts for which there exists built-in support in many tools are associative, commutative operators; and real-valued, rational-valued, and integer-valued arithmetic. Again, these concepts seem to appear often enough to warrant the extra cost of implementing special support in reasoning tools.

This paper is concerned with investigating what kind of special treatment we could give to commonly appearing transitive binary relations, and what effect this treatment has in practice. Adding special treatment of transitive relations to reasoning tools has been the subject of study before, in particular by means of *chaining* [1]. The transitivity axiom

$$\forall x, y, z.\ R(x, y) \wedge R(y, z) \Rightarrow R(x, z)$$

can lead to an expensive proof exploration in resolution and superposition based theorem provers, and can generate a huge number of instances in instance-based provers and SMT-solvers. Transitive relations are also common enough to motivate special built-in support. However, as far as we know, chaining is not implemented in any of the major first-order reasoning tools (at least not in E [6], Vampire [5], Z3 [4], and CVC4 [2], which were used in this paper).

As an alternative to adding built-in support, in this paper we mainly look at (1) what a user of a reasoning tool may do herself to optimize the handling of these relations, and (2) how a preprocessing tool may be able to do this automatically. Adding built-in reasoning support in the tools themselves is not a main concern of this paper.

In: A. Editor, B. Coeditor (eds.): Proceedings of the XYZ Workshop, Location, Country, DD-MMM-YYYY, published at http://ceur-ws.org

$$
\begin{array}{ll}
\textit{reflexive} & \equiv \forall x \quad . R(x,x) \\
\textit{euclidean} & \equiv \forall x,y,z.\ R(x,y) \wedge R(x,z) \Rightarrow R(y,z) \\
\textit{antisymmetric} & \equiv \forall x,y \quad . R(x,y) \wedge R(y,x) \Rightarrow x = y \\
\textit{transitive} & \equiv \forall x,y,z.\ R(x,y) \wedge R(y,z) \Rightarrow R(x,z) \\
\textit{asymmetric} & \equiv \forall x,y \quad . \neg R(x,y) \vee \neg R(y,x) \\
\textit{total} & \equiv \forall x,y \quad . R(x,y) \vee R(y,x) \\
\textit{symmetric} & \equiv \forall x,y \quad . R(x,y) \Rightarrow R(y,x) \\
\textit{coreflexive} & \equiv \forall x,y \quad . R(x,y) \Rightarrow x = y
\end{array}
$$

Figure 1: Definitions of basic properties of binary relations

By "treatment" we mean any way of logically expressing the relation. For example, a possible treatment of a binary relation R in a theory T may simply mean axiomatizing R in T. But it may also mean transforming T into a satisfiability-equivalent theory T' where R does not even syntactically appear.

As an example, consider a theory T in which an equivalence relation R occurs. One way to deal with R is to simply axiomatize it, by means of reflexivity, symmetry, and transitivity:

$$
\begin{array}{ll}
\forall x & . R(x,x) \\
\forall x,y & . R(x,y) \Rightarrow R(y,x) \\
\forall x,y,z. & R(x,y) \wedge R(y,z) \Rightarrow R(x,z)
\end{array}
$$

Another way is to "borrow" the built-in equality treatment that exists in most theorem provers. We can do this by introducing a new symbol *rep*, and replacing all occurrences of $R(x,y)$ by the formula:

$$rep(x) = rep(y)$$

The intuition here is that *rep* is now the representative function of the relation R. No axioms are needed. As we shall see, this alternative treatment of equivalence relations is satisfiability-equivalent with the original one, and actually is beneficial in practice in certain cases.

In general, when considering alternative treatments, we strive to make use of concepts already built-in to the reasoning tool in order to express other concepts that are not built-in.

For the purpose of this paper, we have decided to focus on three different kinds of transitive relations: (1) *equivalence relations* and *partial equivalence relations*, (2) *total orders* and *strict total orders*, and (3) general *reflexive, transitive relations*. The reason we decided to concentrate on these three are because (a) they appear frequently in practice, and (b) we found well-known ways but also novel ways of dealing with these.

The target audience for this paper is thus both people who use reasoning tools and people who implement reasoning tools.

2 Common properties of binary relations

In this section, we take a look at commonly occurring properties of binary relations, which combinations of these are interesting for us to treat specially, and how we may go about discovering these.

Take a look at Fig. 1. It lists 8 basic and common properties of binary relations. Each of these properties can be expressed using one logic clause, which makes it easy to syntactically identify the presence of such a property in a given theory.

When we investigated the number of occurrences of these properties in a subset of the TPTP problem library[1] [7], we ended up with the table in Fig. 2. The table was constructed by gathering all clauses from all TPTP problems (after clausification), and keeping every clause that only contained one binary relation symbol and, possibly, equality. Each such clause was then categorized as an expression of a basic property of a binary relation symbol. We found only 163 such clauses that did not fit any of the 8 properties we chose as basic properties, but were instead instances of two new properties. Both of these were quite esoteric and did not seem to have a standard name in mathematics.

The table also contains occurrences where a *negated relation* was stated to have a certain property, and also occurrences where a *flipped relation* (a relation with its arguments swapped) was stated to have a certain property,

[1] For the statistics in this paper, we decided to only look at unsorted TPTP problems with 10.000 clauses or less.

4945	reflexive
2082	euclidean
1874	antisymmetric
1567	transitive
784	asymmetric
784	total
388	symmetric
3	coreflexive
(163	other)

Figure 2: Number of occurrences of binary relation properties in TPTP

430+18	equivalence relations
181+7	partial equivalence relations
328+7	(strict) total orders
573+20	reflexive, transitive relations (excluding the above)

Figure 3: Number of occurrences of binary relations in TPTP, divided up into Theorem/Unsatisfiable/Unknown/Open problems + Satisfiable/CounterSatisfiable problems.

and also occurrences of combined negated and flipped relations. This explains for example why the number of occurrences of *total* relations is the same as for *asymmetric* relations; if a relation is total, the negated relation is asymmetric and vice-versa.

We adopt the following notation. Given a property of binary relations *prop*, we introduce its *negated version*, which is denoted by $prop^{\neg}$. The property $prop^{\neg}$ holds for R if and only if *prop* holds for $\neg R$. Similarly, we introduce the *flipped version* of a property *prop*, which is denoted by $prop^{\leftrightarrows}$. The property $prop^{\leftrightarrows}$ holds for R if and only if *prop* holds for the flipped version of R.

Using this notation, we can for example say that *total* is equivalent with $asymmetric^{\neg}$. Sometimes the property we call *euclidean* here is called *right euclidean*; the corresponding variant *left euclidean* can be denoted $euclidean^{\leftrightarrows}$. Note that $prop^{\neg}$ is not the same as $\neg prop$! For example, a relation R can be *reflexive*, or $reflexive^{\neg}$ (which means that $\neg R$ is reflexive), or $\neg reflexive$, which means that R is not reflexive.

Using this notation on the 8 original basic properties from Fig. 1, we end up with 32 new basic properties that we can use. (However, as we have already seen, some of these are equivalent to others.)

This paper will look at 5 kinds of different binary relations, which are defined as combinations of basic properties:

$$
\begin{aligned}
&\textit{equivalence relation} && \equiv \{\textit{reflexive}, \textit{symmetric}, \textit{transitive}\}\\
&\textit{partial equivalence relation} && \equiv \{\textit{symmetric}, \textit{transitive}\}\\
&\textit{total order} && \equiv \{\textit{total}, \textit{antisymmetric}, \textit{transitive}\}\\
&\textit{strict total order} && \equiv \{\textit{antisymmetric}^{\neg}, \textit{asymmetric}, \textit{transitive}\}\\
&\textit{reflexive, transitive relation} && \equiv \{\textit{reflexive}, \textit{transitive}\}
\end{aligned}
$$

As a side note, in mathematics, strict total orders are sometimes defined using a property called *trichotomous*, which means that exactly one of $R(x, y)$, $x = y$, or $R(y, x)$ must be true. However, when you clausify this property in the presence of transitivity, you end up with $antisymmetric^{\neg}$ which says that at least one of $R(x, y)$, $x = y$, or $R(y, x)$ must be true. There seems to be no standard name in mathematics for the property $antisymmetric^{\neg}$, which is why we use this name.

In Fig. 3, we display the number of binary relations we have found in (our subset of) the TPTP for each category. The next section describes how we found these.

3 Syntactic discovery of common binary relations

If our goal is to automatically choose the right treatment of equivalence relations, total orders, etc., we must have an automatic way of identifying them in a given theory. It is easy to discover for example an equivalence relation in a theory by means of syntactic inspection. If we find the presence of the axioms *reflexive*, *symmetric*, and *transitive*, for the same relational symbol R, we know that R is an equivalence relation.

$$
\begin{array}{ll}
\mathit{total} & \Leftrightarrow \mathit{total}^{\curvearrowleft} \\
\mathit{total} & \Leftrightarrow \mathit{asymmetric}^{\neg} \\
\mathit{total} & \Leftrightarrow \mathit{asymmetric}^{\neg\curvearrowleft} \\
\mathit{reflexive} & \Leftrightarrow \mathit{reflexive}^{\curvearrowleft} \\
\mathit{reflexive}^{\neg} & \Leftrightarrow \mathit{reflexive}^{\neg\curvearrowleft} \\
\mathit{symmetric} & \Leftrightarrow \mathit{symmetric}^{\neg} \\
\mathit{symmetric} & \Leftrightarrow \mathit{symmetric}^{\curvearrowleft} \\
\mathit{symmetric} & \Leftrightarrow \mathit{symmetric}^{\neg\curvearrowleft} \\
\mathit{transitive} & \Leftrightarrow \mathit{transitive}^{\curvearrowleft} \\
\mathit{transitive}^{\neg} & \Leftrightarrow \mathit{transitive}^{\neg\curvearrowleft} \\
\mathit{coreflexive} & \Leftrightarrow \mathit{coreflexive}^{\curvearrowleft} \\
\mathit{coreflexive}^{\neg} & \Leftrightarrow \mathit{coreflexive}^{\neg\curvearrowleft} \\
\mathit{antisymmetric} & \Leftrightarrow \mathit{antisymmetric}^{\curvearrowleft} \\
\mathit{antisymmetric}^{\neg} & \Leftrightarrow \mathit{antisymmetric}^{\neg\curvearrowleft}
\end{array}
$$

Figure 4: Basic properties that are equivalent

But there is a problem. There are other ways of axiomatizing equivalence relations. For example, a much more common way to axiomatize equivalence relations in the TPTP is to state the two properties *reflexive* and *euclidean* for R.

Rather than enumerating all possible ways to axiomatize certain relations by hand, we wrote a program that computes all possible ways for any combination of basic properties to imply any other combination of basic properties. Our program generates a table that can be precomputed in a minute or so and then used to very quickly detect any alternative axiomatization of binary relations using basic properties.

Let us explain how this table was generated. We start with a list of 32 basic properties (the 8 original basic properties, plus their negated, flipped, and negated flipped versions). Firstly, we use an automated theorem prover (we used E [6]) to discover which of these are equivalent with other such properties. The result is displayed in Fig. 4. Thus, 17 basic properties can be removed from the list, because they can be expressed using other properties. The list of basic properties now has 15 elements left.

Secondly, we want to generate all implications of the form $\{prop_1, \ldots, prop_n\} \Rightarrow prop$ where the set $\{prop_1, \ldots, prop_n\}$ is minimal. We do this separately for each $prop$. The results are displayed in Fig. 5.

The procedure uses a simple constraint solver (a SAT-solver) to keep track of all implications it has tried so far, and consists of one main loop. At every loop iteration, the constraint solver guesses a set $\{prop_1, \ldots, prop_n\}$ from the set of all properties $P - \{prop\}$. The procedure then asks E whether or not $\{prop_1, \ldots, prop_n\} \Rightarrow prop$ is valid. If it is, then we look at the proof that E produces, and print the implication $\{prop_a, \ldots, prop_b\} \Rightarrow prop$, where $\{prop_a, \ldots, prop_b\}$ is the subset of properties that were used in the proof. We then also tell the constraint solver never to guess a superset of $\{prop_a, \ldots, prop_b\}$ again. If the guessed implication can not be proven, we tell the constraint solver to never guess a subset of $\{prop_1, \ldots, prop_n\}$ again. The procedure stops when no guesses that satisfy all constraints can be made anymore.

After the loop terminates, we may need to clean up the implications somewhat because some implications may subsume others.

In order to avoid generating inconsistent sets $\{prop_1, \ldots, prop_n\}$ (that would imply any other property), we also add the artificial inconsistent property *false* to the set, and generate implications for this property first. We exclude any found implication here from the implication sets of the real properties.

This procedure generates a complete list of minimal implications. It works well in practice, especially if all guesses are maximized according to their size. The vast majority of the time is spent on the implication proofs, and no significant time is spent in the SAT-solver.

To detect a binary relation R with certain properties in a given theory, we simply gather all basic properties about R that occur in the theory, and then compute which other properties they imply, using the pre-generated table. In this way, we never have to do any theorem proving in order to detect a binary relation with certain properties.

$$
\begin{array}{ll}
\mathit{false} & \Leftarrow \{\mathit{reflexive}, \mathit{reflexive}^{\neg}\} \\
\mathit{false} & \Leftarrow \{\mathit{reflexive}, \mathit{asymmetric}\} \\
\mathit{false} & \Leftarrow \{\mathit{total}, \mathit{reflexive}^{\neg}\} \\
\mathit{false} & \Leftarrow \{\mathit{total}, \mathit{asymmetric}\} \\
\mathit{reflexive} & \Leftarrow \{\mathit{total}\} \\
\mathit{euclidean} & \Leftarrow \{\mathit{symmetric}, \mathit{asymmetric}\} \\
\mathit{euclidean} & \Leftarrow \{\mathit{total}, \mathit{symmetric}\} \\
\mathit{euclidean} & \Leftarrow \{\mathit{antisymmetric}, \mathit{coreflexive}^{\neg}\} \\
\mathit{euclidean} & \Leftarrow \{\mathit{transitive}, \mathit{euclidean}^{\neg}\} \\
\mathit{euclidean} & \Leftarrow \{\mathit{antisymmetric}, \mathit{euclidean}^{\neg}\} \\
\mathit{euclidean} & \Leftarrow \{\mathit{reflexive}, \mathit{coreflexive}^{\neg}\} \\
\mathit{euclidean} & \Leftarrow \{\mathit{reflexive}, \mathit{euclidean}^{\neg}\} \\
\mathit{euclidean} & \Leftarrow \{\mathit{coreflexive}\} \\
\mathit{euclidean} & \Leftarrow \{\mathit{total}, \mathit{coreflexive}^{\neg}\} \\
\mathit{euclidean} & \Leftarrow \{\mathit{reflexive}^{\neg}, \mathit{euclidean}^{\leftharpoonup}\} \\
\mathit{euclidean} & \Leftarrow \{\mathit{reflexive}, \mathit{euclidean}^{\neg\leftharpoonup}\} \\
\mathit{euclidean} & \Leftarrow \{\mathit{asymmetric}, \mathit{euclidean}^{\leftharpoonup}\} \\
\mathit{euclidean} & \Leftarrow \{\mathit{symmetric}, \mathit{antisymmetric}\} \\
\mathit{euclidean} & \Leftarrow \{\mathit{transitive}, \mathit{reflexive}^{\neg}, \mathit{euclidean}^{\neg\leftharpoonup}\} \\
\mathit{euclidean} & \Leftarrow \{\mathit{total}, \mathit{euclidean}^{\neg}\} \\
\mathit{euclidean} & \Leftarrow \{\mathit{transitive}, \mathit{coreflexive}^{\neg}\} \\
\mathit{euclidean} & \Leftarrow \{\mathit{asymmetric}, \mathit{coreflexive}^{\neg}\} \\
\mathit{euclidean} & \Leftarrow \{\mathit{asymmetric}, \mathit{euclidean}^{\neg}\} \\
\mathit{euclidean} & \Leftarrow \{\mathit{total}, \mathit{euclidean}^{\neg\leftharpoonup}\} \\
\mathit{euclidean} & \Leftarrow \{\mathit{antisymmetric}, \mathit{reflexive}^{\neg}, \mathit{euclidean}^{\neg\leftharpoonup}\} \\
\mathit{euclidean} & \Leftarrow \{\mathit{asymmetric}, \mathit{euclidean}^{\neg\leftharpoonup}\} \\
\mathit{euclidean} & \Leftarrow \{\mathit{symmetric}, \mathit{transitive}\} \\
\mathit{euclidean} & \Leftarrow \{\mathit{euclidean}^{\neg}, \mathit{euclidean}^{\leftharpoonup}\} \\
\mathit{euclidean} & \Leftarrow \{\mathit{symmetric}, \mathit{euclidean}^{\leftharpoonup}\} \\
\mathit{euclidean} & \Leftarrow \{\mathit{reflexive}, \mathit{euclidean}^{\leftharpoonup}\} \\
\mathit{euclidean} & \Leftarrow \{\mathit{reflexive}, \mathit{symmetric}, \mathit{antisymmetric}^{\neg}\} \\
\mathit{euclidean} & \Leftarrow \{\mathit{reflexive}, \mathit{symmetric}, \mathit{transitive}^{\neg}\} \\
\mathit{euclidean} & \Leftarrow \{\mathit{total}, \mathit{euclidean}^{\leftharpoonup}\} \\
\mathit{euclidean} & \Leftarrow \{\mathit{euclidean}^{\leftharpoonup}, \mathit{coreflexive}^{\neg}\} \\
\mathit{antisymmetric} & \Leftarrow \{\mathit{coreflexive}\} \\
\mathit{antisymmetric} & \Leftarrow \{\mathit{asymmetric}\} \\
\mathit{antisymmetric} & \Leftarrow \{\mathit{transitive}, \mathit{reflexive}^{\neg}\} \\
\mathit{antisymmetric} & \Leftarrow \{\mathit{reflexive}^{\neg}, \mathit{euclidean}^{\leftharpoonup}\} \\
\mathit{antisymmetric} & \Leftarrow \{\mathit{euclidean}, \mathit{reflexive}^{\neg}\} \\
\mathit{transitive} & \Leftarrow \{\mathit{symmetric}, \mathit{asymmetric}\} \\
\mathit{transitive} & \Leftarrow \{\mathit{total}, \mathit{symmetric}\} \\
\mathit{transitive} & \Leftarrow \{\mathit{antisymmetric}, \mathit{coreflexive}^{\neg}\} \\
\mathit{transitive} & \Leftarrow \{\mathit{coreflexive}\} \\
\mathit{transitive} & \Leftarrow \{\mathit{euclidean}, \mathit{transitive}^{\neg}\} \\
\mathit{transitive} & \Leftarrow \{\mathit{symmetric}, \mathit{antisymmetric}\} \\
\mathit{transitive} & \Leftarrow \{\mathit{reflexive}, \mathit{euclidean}^{\neg}\} \\
\mathit{transitive} & \Leftarrow \{\mathit{euclidean}, \mathit{reflexive}^{\neg}\} \\
\mathit{transitive} & \Leftarrow \{\mathit{total}, \mathit{euclidean}^{\neg}\} \\
\mathit{transitive} & \Leftarrow \{\mathit{reflexive}, \mathit{coreflexive}^{\neg}\} \\
\mathit{transitive} & \Leftarrow \{\mathit{asymmetric}, \mathit{coreflexive}^{\neg}\} \\
\mathit{transitive} & \Leftarrow \{\mathit{asymmetric}, \mathit{euclidean}^{\neg}\} \\
\mathit{transitive} & \Leftarrow \{\mathit{asymmetric}, \mathit{transitive}^{\neg}\} \\
\mathit{transitive} & \Leftarrow \{\mathit{antisymmetric}, \mathit{transitive}^{\neg}\} \\
\mathit{transitive} & \Leftarrow \{\mathit{euclidean}^{\leftharpoonup}, \mathit{euclidean}^{\neg\leftharpoonup}\} \\
\mathit{transitive} & \Leftarrow \{\mathit{euclidean}, \mathit{coreflexive}^{\neg}\} \\
\mathit{transitive} & \Leftarrow \{\mathit{total}, \mathit{coreflexive}^{\neg}\} \\
\mathit{transitive} & \Leftarrow \{\mathit{euclidean}, \mathit{asymmetric}\} \\
\mathit{transitive} & \Leftarrow \{\mathit{total}, \mathit{euclidean}\} \\
\mathit{transitive} & \Leftarrow \{\mathit{reflexive}, \mathit{euclidean}^{\neg\leftharpoonup}\} \\
\mathit{transitive} & \Leftarrow \{\mathit{euclidean}^{\leftharpoonup}, \mathit{transitive}^{\neg}\} \\
\mathit{transitive} & \Leftarrow \{\mathit{euclidean}, \mathit{symmetric}\} \\
\mathit{transitive} & \Leftarrow \{\mathit{reflexive}^{\neg}, \mathit{euclidean}^{\leftharpoonup}\} \\
\mathit{transitive} & \Leftarrow \{\mathit{antisymmetric}, \mathit{euclidean}^{\neg}\} \\
\mathit{transitive} & \Leftarrow \{\mathit{total}, \mathit{euclidean}^{\neg\leftharpoonup}\} \\
\mathit{transitive} & \Leftarrow \{\mathit{euclidean}, \mathit{antisymmetric}^{\neg}\} \\
\mathit{transitive} & \Leftarrow \{\mathit{reflexive}, \mathit{euclidean}^{\leftharpoonup}\} \\
\mathit{transitive} & \Leftarrow \{\mathit{reflexive}, \mathit{symmetric}, \mathit{antisymmetric}^{\neg}\} \\
\mathit{transitive} & \Leftarrow \{\mathit{euclidean}^{\leftharpoonup}, \mathit{coreflexive}^{\neg}\} \\
\mathit{transitive} & \Leftarrow \{\mathit{reflexive}, \mathit{symmetric}, \mathit{transitive}^{\neg}\}
\end{array}
$$

$$
\begin{array}{ll}
\mathit{transitive} & \Leftarrow \{\mathit{euclidean}, \mathit{antisymmetric}\} \\
\mathit{transitive} & \Leftarrow \{\mathit{euclidean}, \mathit{euclidean}^{\neg}\} \\
\mathit{transitive} & \Leftarrow \{\mathit{total}, \mathit{euclidean}^{\leftharpoonup}\} \\
\mathit{transitive} & \Leftarrow \{\mathit{asymmetric}, \mathit{euclidean}^{\leftharpoonup}\} \\
\mathit{transitive} & \Leftarrow \{\mathit{symmetric}, \mathit{euclidean}^{\leftharpoonup}\} \\
\mathit{transitive} & \Leftarrow \{\mathit{asymmetric}, \mathit{euclidean}^{\neg\leftharpoonup}\} \\
\mathit{transitive} & \Leftarrow \{\mathit{antisymmetric}, \mathit{euclidean}^{\neg\leftharpoonup}\} \\
\mathit{transitive} & \Leftarrow \{\mathit{euclidean}, \mathit{reflexive}\} \\
\mathit{transitive} & \Leftarrow \{\mathit{euclidean}, \mathit{euclidean}^{\neg\leftharpoonup}\} \\
\mathit{transitive} & \Leftarrow \{\mathit{antisymmetric}, \mathit{euclidean}^{\leftharpoonup}\} \\
\mathit{transitive} & \Leftarrow \{\mathit{euclidean}^{\leftharpoonup}, \mathit{antisymmetric}^{\neg}\} \\
\mathit{transitive} & \Leftarrow \{\mathit{euclidean}^{\neg}, \mathit{euclidean}^{\leftharpoonup}\} \\
\mathit{total} & \Leftarrow \{\mathit{reflexive}, \mathit{coreflexive}^{\neg}\} \\
\mathit{total} & \Leftarrow \{\mathit{reflexive}, \mathit{euclidean}^{\neg\leftharpoonup}\} \\
\mathit{total} & \Leftarrow \{\mathit{reflexive}, \mathit{euclidean}^{\neg}\} \\
\mathit{total} & \Leftarrow \{\mathit{reflexive}, \mathit{antisymmetric}^{\neg}\} \\
\mathit{total} & \Leftarrow \{\mathit{reflexive}, \mathit{transitive}^{\neg}\} \\
\mathit{symmetric} & \Leftarrow \{\mathit{coreflexive}^{\neg}\} \\
\mathit{symmetric} & \Leftarrow \{\mathit{euclidean}^{\neg}, \mathit{reflexive}^{\neg}\} \\
\mathit{symmetric} & \Leftarrow \{\mathit{euclidean}, \mathit{reflexive}^{\neg}\} \\
\mathit{symmetric} & \Leftarrow \{\mathit{coreflexive}\} \\
\mathit{symmetric} & \Leftarrow \{\mathit{total}, \mathit{euclidean}^{\neg}\} \\
\mathit{symmetric} & \Leftarrow \{\mathit{total}, \mathit{euclidean}\} \\
\mathit{symmetric} & \Leftarrow \{\mathit{euclidean}, \mathit{asymmetric}\} \\
\mathit{symmetric} & \Leftarrow \{\mathit{asymmetric}, \mathit{euclidean}^{\neg}\} \\
\mathit{symmetric} & \Leftarrow \{\mathit{reflexive}, \mathit{euclidean}^{\neg\leftharpoonup}\} \\
\mathit{symmetric} & \Leftarrow \{\mathit{reflexive}, \mathit{euclidean}^{\leftharpoonup}\} \\
\mathit{symmetric} & \Leftarrow \{\mathit{reflexive}^{\neg}, \mathit{euclidean}^{\leftharpoonup}\} \\
\mathit{symmetric} & \Leftarrow \{\mathit{reflexive}^{\neg}, \mathit{euclidean}^{\neg\leftharpoonup}\} \\
\mathit{symmetric} & \Leftarrow \{\mathit{total}, \mathit{euclidean}^{\neg\leftharpoonup}\} \\
\mathit{symmetric} & \Leftarrow \{\mathit{asymmetric}, \mathit{euclidean}^{\leftharpoonup}\} \\
\mathit{symmetric} & \Leftarrow \{\mathit{euclidean}^{\neg}, \mathit{euclidean}^{\leftharpoonup}\} \\
\mathit{symmetric} & \Leftarrow \{\mathit{total}, \mathit{euclidean}^{\leftharpoonup}\} \\
\mathit{symmetric} & \Leftarrow \{\mathit{euclidean}, \mathit{euclidean}^{\neg\leftharpoonup}\} \\
\mathit{symmetric} & \Leftarrow \{\mathit{euclidean}, \mathit{euclidean}^{\leftharpoonup}\} \\
\mathit{symmetric} & \Leftarrow \{\mathit{euclidean}, \mathit{reflexive}\} \\
\mathit{symmetric} & \Leftarrow \{\mathit{asymmetric}, \mathit{euclidean}^{\neg\leftharpoonup}\} \\
\mathit{symmetric} & \Leftarrow \{\mathit{euclidean}^{\neg}, \mathit{euclidean}^{\neg\leftharpoonup}\} \\
\mathit{symmetric} & \Leftarrow \{\mathit{reflexive}, \mathit{euclidean}^{\neg}\} \\
\mathit{coreflexive} & \Leftarrow \{\mathit{symmetric}, \mathit{asymmetric}\} \\
\mathit{coreflexive} & \Leftarrow \{\mathit{antisymmetric}, \mathit{coreflexive}^{\neg}\} \\
\mathit{coreflexive} & \Leftarrow \{\mathit{transitive}, \mathit{euclidean}^{\neg}, \mathit{reflexive}^{\neg}\} \\
\mathit{coreflexive} & \Leftarrow \{\mathit{asymmetric}, \mathit{coreflexive}^{\neg}\} \\
\mathit{coreflexive} & \Leftarrow \{\mathit{euclidean}, \mathit{reflexive}^{\neg}\} \\
\mathit{coreflexive} & \Leftarrow \{\mathit{reflexive}, \mathit{antisymmetric}, \mathit{euclidean}^{\neg}\} \\
\mathit{coreflexive} & \Leftarrow \{\mathit{asymmetric}, \mathit{euclidean}^{\neg}\} \\
\mathit{coreflexive} & \Leftarrow \{\mathit{euclidean}, \mathit{asymmetric}\} \\
\mathit{coreflexive} & \Leftarrow \{\mathit{symmetric}, \mathit{antisymmetric}\} \\
\mathit{coreflexive} & \Leftarrow \{\mathit{total}, \mathit{antisymmetric}, \mathit{euclidean}^{\neg}\} \\
\mathit{coreflexive} & \Leftarrow \{\mathit{reflexive}, \mathit{antisymmetric}, \mathit{euclidean}^{\neg\leftharpoonup}\} \\
\mathit{coreflexive} & \Leftarrow \{\mathit{reflexive}, \mathit{antisymmetric}, \mathit{euclidean}^{\leftharpoonup}\} \\
\mathit{coreflexive} & \Leftarrow \{\mathit{symmetric}, \mathit{transitive}, \mathit{reflexive}^{\neg}\} \\
\mathit{coreflexive} & \Leftarrow \{\mathit{transitive}, \mathit{reflexive}^{\neg}, \mathit{coreflexive}^{\neg}\} \\
\mathit{coreflexive} & \Leftarrow \{\mathit{antisymmetric}, \mathit{euclidean}^{\neg}, \mathit{euclidean}^{\leftharpoonup}\} \\
\mathit{coreflexive} & \Leftarrow \{\mathit{total}, \mathit{antisymmetric}, \mathit{euclidean}^{\leftharpoonup}\} \\
\mathit{coreflexive} & \Leftarrow \{\mathit{total}, \mathit{euclidean}, \mathit{antisymmetric}\} \\
\mathit{coreflexive} & \Leftarrow \{\mathit{total}, \mathit{antisymmetric}, \mathit{euclidean}^{\neg\leftharpoonup}\} \\
\mathit{coreflexive} & \Leftarrow \{\mathit{antisymmetric}, \mathit{reflexive}^{\neg}, \mathit{euclidean}^{\neg\leftharpoonup}\} \\
\mathit{coreflexive} & \Leftarrow \{\mathit{reflexive}^{\neg}, \mathit{euclidean}^{\leftharpoonup}\} \\
\mathit{coreflexive} & \Leftarrow \{\mathit{antisymmetric}, \mathit{euclidean}^{\neg}, \mathit{reflexive}^{\neg}\} \\
\mathit{coreflexive} & \Leftarrow \{\mathit{asymmetric}, \mathit{euclidean}^{\leftharpoonup}\} \\
\mathit{coreflexive} & \Leftarrow \{\mathit{asymmetric}, \mathit{euclidean}^{\neg\leftharpoonup}\} \\
\mathit{coreflexive} & \Leftarrow \{\mathit{antisymmetric}, \mathit{euclidean}^{\neg}, \mathit{euclidean}^{\neg\leftharpoonup}\} \\
\mathit{coreflexive} & \Leftarrow \{\mathit{euclidean}, \mathit{antisymmetric}, \mathit{euclidean}^{\neg\leftharpoonup}\} \\
\mathit{coreflexive} & \Leftarrow \{\mathit{euclidean}, \mathit{antisymmetric}, \mathit{euclidean}^{\leftharpoonup}\} \\
\mathit{coreflexive} & \Leftarrow \{\mathit{euclidean}, \mathit{reflexive}, \mathit{antisymmetric}\} \\
\mathit{coreflexive} & \Leftarrow \{\mathit{transitive}, \mathit{reflexive}^{\neg}, \mathit{euclidean}^{\neg\leftharpoonup}\}
\end{array}
$$

Figure 5: The complete list of implications between properties

4 Handling equivalence relations

Equalification As mentioned in the introduction, an alternative way of handling equivalence relations R is to create a new symbol *rep* and replace all occurrences of R with a formula involving *rep*:

$$\begin{array}{ll} R \textit{ reflexive} & \\ R \textit{ symmetric} & \twoheadrightarrow \\ R \textit{ transitive} & \\ T\,[..R(x,y)..] & \qquad T\,[..\mathit{rep}(x) = \mathit{rep}(y)..] \end{array}$$

To explain the above notation: We have two theories, one on the left-hand side of the arrow, and one on the right-hand side of the arrow. The transformation transforms any theory that looks like the left-hand side into a theory that looks like the right-hand side. We write $T\,[..e..]$ for theories in which e occurs syntactically; in the transformation, all occurrences of e should be replaced.

We call the above transformation *equalification*. This transformation may be beneficial because the reasoning now involves built-in equality reasoning instead of reasoning about an unknown symbol using axioms.

The transformation is correct, meaning that it preserves (non-)satisfiability: ($\Rightarrow$) If we have a model of the LHS theory, then R must be interpreted as an equivalence relation. Let *rep* be the representative function of R, in other words we have $R(x,y) \Leftrightarrow \mathit{rep}(x) = \mathit{rep}(y)$. Thus we also have a model of the RHS theory. ($\Leftarrow$) If we have a model of the RHS theory, let $R(x,y) := \mathit{rep}(x) = \mathit{rep}(y)$. It is clear that R is reflexive, symmetric, and transitive, and therefore we have model of the LHS theory.

In the transformation, we also remove the axioms for reflexivity, symmetry, and transitivity, because they are not needed anymore. But what if R is axiomatized as an equivalence relation using different axioms? Then we can remove any axiom about R that is implied by reflexivity, symmetry, and transitivity. Luckily we have already computed a table of which properties imply which other ones (shown in Fig. 5).

Pequalification There are commonly occurring binary relations called *partial equivalence relations* that almost behave as equivalence relations, but not quite. In particular, they do not have to obey the axiom of reflexivity. Can we do something for these too?

It turns out that a set with a partial equivalence relation R can be partitioned into two subsets: (1) one subset on which R is an actual equivalence relation, and (2) one subset of elements which are not related to anything, not even themselves.

Thus, an alternative way of handling partial equivalence relations R is to create two new symbols, *rep* and P, and replace all occurrences of R with a formula involving *rep* and P.

$$\begin{array}{ll} R \textit{ symmetric} & \\ R \textit{ transitive} & \twoheadrightarrow \\ T\,[..R(x,y)..] & \qquad T\,[..(P(x) \wedge P(y) \wedge \mathit{rep}(x) = \mathit{rep}(y))..] \end{array}$$

Here, P is the predicate that indicates the subset on which R behaves as an equivalence relation.

We call this transformation *pequalification*. This transformation may be beneficial because the reasoning now involves built-in equality reasoning instead of reasoning about an unknown symbol using axioms. However, there is also a clear price to pay since the size of the problem grows considerably.

The transformation is correct, meaning that it preserves (non-)satisfiability: ($\Rightarrow$) If we have a model of the LHS theory, then R must be interpreted as a partial equivalence relation. Let $P(x) := R(x,x)$, in other words P is the subset on which R behaves like an equivalence relation. Let *rep* be a representative function of R on P, in other words we have $(P(x) \wedge P(y)) \Rightarrow (R(x,y) \Leftrightarrow \mathit{rep}(x) = \mathit{rep}(y))$. By the definition of P we then also have $R(x,y) \Leftrightarrow (P(x) \wedge P(y) \wedge \mathit{rep}(x) = \mathit{rep}(y))$. Thus we also have a model of the RHS theory. ($\Leftarrow$) If we have a model of the RHS theory, let $R(x,y) := P(x) \wedge P(y) \wedge \mathit{rep}(x) = \mathit{rep}(y)$. This R is symmetric and transitive, and therefore we have model of the LHS theory.

5 Handling total orders

Ordification Many reasoning tools have built-in support for arithmetic, in particular they support an order $\leqslant$ on numbers. It turns out that we can "borrow" this operator when handling general total orders. Suppose we have a total order:

$R : A \times A \to \mathit{Bool}$

We now introduce a new injective function:

$\mathit{rep} : A \to \mathbb{R}$

We then replace all occurrences of R with a formula involving rep in the following way:

R *total*		
R *antisymmetric*	$\twoheadrightarrow$	$\forall x, y.\ \mathit{rep}(x) = \mathit{rep}(y) \Rightarrow x = y$
R *transitive*		
$T\ [\,..\,R(x, y)\,..\,]$		$T\ [\,..\,\mathit{rep}(x) \leqslant \mathit{rep}(y)\,..\,]$

(Here, $\leqslant$ is the order on reals.) We call this transformation *ordification*. This transformation may be beneficial because the reasoning now involves built-in arithmetic reasoning instead of reasoning about an unknown symbol using axioms.

The above transformation is correct, meaning that it preserves (non-)satisfiability: ($\Rightarrow$) If we have a model of the LHS theory, then without loss of generality (by Löwenheim-Skolem), we can assume that the domain is countable. Also, R must be interpreted as a total order. We now construct rep recursively as a mapping from the model domain to $\mathbb{R}$, such that we have $R(x, y) \Leftrightarrow \mathit{rep}(x) \leqslant \mathit{rep}(y)$, in the following way. Let $\{\, a_0, a_1, a_2, ..\}$ be the domain of the model, and set $\mathit{rep}(a_0) := 0$. For any $n > 0$, pick a value for $\mathit{rep}(a_n)$ that is consistent with the total order R and all earlier domain elements a_i, for $0 \leqslant i < n$. This can always be done because there is always extra room for a new, unique element between any two distinct values of $\mathbb{R}$. Thus rep is injective and we also have a model of the RHS theory. ($\Leftarrow$) If we have a model of the RHS theory, let $R(x, y) := \mathit{rep}(x) \leqslant \mathit{rep}(y)$. It is clear that R is total and transitive, and also antisymmetric because rep is injective, and therefore we have model of the LHS theory.

Note on $\mathbb{Q}$ vs. $\mathbb{R}$ The proof would have worked for $\mathbb{Q}$ as well instead of $\mathbb{R}$. The transformation can therefore be used for any tool that supports $\mathbb{Q}$ or $\mathbb{R}$ or both, and should choose whichever comparison operator is cheapest if there is a choice. Using integer arithmetic would however not have been correct.

Note on strict total orders One may have expected to have a transformation specifically targeted to strict total orders, i.e. something like:

R *antisymmetric*$^{\neg}$		
R *asymmetric*	$\twoheadrightarrow$	$\forall x, y.\ \mathit{rep}(x) = \mathit{rep}(y) \Rightarrow x = y$
R *transitive*		
$T\ [\,..\,R(x, y)\,..\,]$		$T\ [\,..\,\mathit{rep}(x) < \mathit{rep}(y)\,..\,]$

However, the transformation for total orders already covers this case! Any strict total order R is also recognized as a total order $\neg R$, and ordification already transforms such theories in the correct way. The only difference is that $R(x, y)$ is replaced with $\neg(\mathit{rep}(x) \leqslant \mathit{rep}(y))$ instead of $\mathit{rep}(x) < \mathit{rep}(y)$, which is satisfiability-equivalent. (There may be a practical performance difference, which is not something we have investigated.)

Maxification Some reasoning tools do not have orders on real arithmetic built-in, but they may have other concepts that are built-in that can be used to express total orders instead. One such concept is handling of associative, commutative (AC) operators.

For such a tool, one alternative way of handling total orders R is to create a new function symbol max and replace all occurrences of R with a formula involving max:

R *total*		max *associative*
R *antisymmetric*	$\twoheadrightarrow$	max *commutative*
R *transitive*		$\forall x, y.\ \mathit{max}(x, y) = x \vee \mathit{max}(x, y) = y$
$T\ [\,..\,R(x, y)\,..\,]$		$T\ [\,..\,\mathit{max}(x, y) = y\,..\,]$

We call this transformation *maxification*. This transformation may be beneficial because the reasoning now involves built-in equality reasoning with AC unification (and one extra axiom) instead of reasoning about an unknown relational symbol (using three axioms).

The above transformation is correct, meaning that it preserves (non-)satisfiability: ($\Rightarrow$) If we have a model of the LHS theory, then R must be interpreted as a total order. Let max be the maximum function associated with this order. Clearly, it must be associative and commutative, and the third axiom also holds. Moreover, we have $R(x, y) \Leftrightarrow max(x, y) = y$. Thus we also have a model of the RHS theory. ($\Leftarrow$) If we have a model of the RHS theory, let $R(x, y) := max(x, y) = y$. Given the axioms in the RHS theory, R is total, antisymmetric, and transitive, and therefore we have model of the LHS theory.

(It may be the case that maxification can also be used to express orders that are weaker than total orders. At the time of this writing, we have not figured out how to do this.)

6 Handling reflexive, transitive relations

Transification The last transformation we present is designed as an alternative treatment for any relation that is reflexive and transitive. It does not make use of any built-in concept in the tool. Instead, it transforms theories with a transitivity axiom into theories without that transitivity axiom. Instead, transitivity is *specialized* at each *positive occurrence* of the relational symbol.

As such, an alternative way of handling reflexive, transitive relations R is to create a new symbol Q and replace all positive occurrences of R with a formula involving Q; the negative occurrences are simply replaced by $\neg Q$:

$$
\begin{array}{lcl}
R \textit{ reflexive} & & Q \textit{ reflexive} \\
R \textit{ transitive} & \rightarrow & \\
T\ [..R(x,y)\ ..\ \neg R(x,y)..] & & T\ [..(\forall r.\ Q(r,x) \Rightarrow Q(r,y))\ ..\ \neg Q(x,y)\ ..]
\end{array}
$$

We call this transformation *transification*. This transformation may be beneficial because reasoning about transitivity in a naive way can be very expensive for theorem provers, because from transitivity there are many possible conclusions to draw that trigger each other "recursively". The transformation only works on problems where every occurrence of R is either positive or negative (and not both, such as under an equivalence operator). If this is not the case, the problem has to be translated into one where this is the case. This can for example be done by means of clausification.

Note that the resulting theory does not blow-up; only clauses with a positive occurrence of R gets one extra literal per occurrence.

We replace any positive occurrence of $R(x, y)$ with an implication that says "for any r, if you could reach x from r, now you can reach y too". Thus, we have specialized the transitivity axiom for every positive occurrence of R.

Note that in the RHS theory, Q does not have to be transitive! Nonetheless, the transformation is correct, meaning that it preserves (non-)satisfiability: ($\Rightarrow$) If we have a model of the LHS theory, then R is reflexive and transitive. Now, set $Q(x, y) := R(x, y)$. Q is obviously reflexive. We have to show that $R(x, y)$ implies $\forall r.\ Q(r, x) \Rightarrow Q(r, y)$. This is indeed the case because R is transitive. Thus we also have a model of the RHS theory. ($\Leftarrow$) If we have a model of the RHS theory, then Q is reflexive. Now, set $R(x, y) := \forall r.\ Q(r, x) \Rightarrow Q(r, y)$. R is reflexive (by reflexivity of implication) and transitive (by transitivity of implication). Finally, we have to show that $\neg Q(x, y)$ implies $\neg R(x, y)$, which is the same as showing that $\forall r.\ Q(r, x) \Rightarrow Q(r, y)$ implies $Q(x, y)$, which is true because Q is reflexive. Thus we also have a model of the RHS theory.

7 Experimental results

We evaluate the effects of the different axiomatizations using two different resolution based theorem provers, E 1.9 [6] (with the *xAuto* and *tAuto* options) and Vampire 4.0 [5] (with the *casc mode* option), and two SMT-solvers, Z3 4.4.2 [4] and CVC4 1.4 [2]. The experiments were performed on a 2xQuad Core Intel Xeon E5620 processor with 24 GB physical memory, running at 2.4 GHz. We use a time limit of 5 minutes on each problem. For Vampire, no time limit is passed directly to the theorem prover but instead the process is terminated once the time limit has passed. This was done to keep solving times more stable, since Vampire uses the time limit to control its search strategies.

We started from a set of 11674 test problems from the TPTP, listed as Unsatisfiable, Theorem, Unknown or Open (leaving out the very large theories). For each problem, a new theory was generated for each applicable transformation. For most problems, no relation matching any of the given criteria was detected, and thus no new

		E			Vampire			Z3			CVC4		
equalification	(430)	422	+4	-33	428	+0	-4	362	+50	-3	370	+18	-39
pequalification	(181)	96	+0	-34	87	+5	-8	38	+9	-4	59	+1	-11
transification	(573)	324	+2	-26	274	**+32**	-10	234	+10	-46	255	+13	-42
ordification	(328)		n/a		292	**+19**	-15	238	**+51**	-13	267	**+13**	-15
maxification	(328)	273	+1	-23	292	+0	-1	238	+1	-41	267	+4	-0

Figure 6: Table showing for each theorem prover the number of test problems solved before the transformation, how many new problems are solved after the transformation, and the number of problems that could be solved before but not after the transformation. (Total number of applicable problems for each transformation in parentheses). A **+value** in boldface indicates that there were hard problems (Rating 1.0) solved with that combination of treatment and theorem prover. An underlined -value indicates that time slicing (running both methods in 50% of the time each) solves strictly more problems with that combination of treatment and theorem prover.

theories were produced for these problems. Evaluation of reasoning tools on Satisfiable and CounterSatisfiable problems is left as future work.

7.1 Equivalence relations

Equivalence relations were present in 430 of the test problems. The majority of these problems appear in the GEO and SYN categories. Interestingly, among these 430 problems, there are only 23 problems whose equivalence relations are axiomatized with transitivity axioms. The remaining 407 problems axiomatize equivalence relations with euclidean and reflexivity axioms, as discussed in section 3. The number of equivalence relations in each problem ranges from 1 to 40, where problems with many equivalence relations all come from the SYN category. There is no clear correspondence between the number of equivalence relations in a problem and the performance of the prover prior to and after the transformation.

Equalification As can be seen in Fig. 6, equalification performs very well with Z3, and somewhat well with CVC4, while it worsens the results of the resolution based provers, which already performed well on the original problems. Using a time slicing strategy, which runs the prover on the original problem for half the time and on the transformed problem for the second half, solves a strict superset of problems than the original for all of the theorem provers used in the evaluation. Fig. 7 shows in more detail the effect on solving times for the different theorem provers.

Pequalification In 181 of the test problems, relations that are transitive and symmetric, but not reflexive, were found. The majority of these problems are in the CAT and FLD categories of TPTP. All of the tested theorem provers perform worse on these problems compared to the problems with true equivalence relations. This is also the case after performing pequalification. Pequalification turns out to be particularly bad for E, which solves 34 fewer problems after the transformation. Pequalification makes Z3 perform only slightly better and Vampire and CVC4 slightly worse. When the differences to the performance is small, it is hard to know if the effect is due to the new axiomatisation, or to other reasons. For example, we have observed that simply shuffling the axioms of a theory can cause the results to deviate in a similar way.

7.2 Total orders

Total orders were found in 328 problems. The majority are in the SWV category, and the remaining in HWV, LDA and NUM. In 77 of the problems, the total order is positively axiomatized, and in the other 251 problems it is negative (and thus axiomatized as a strict total order). There is never more than one order present in any of the problems.

Ordification For each of the problems, we ran the theorem provers with built-in support for arithmetic on the problems before and after applying ordification. Vampire was run on a version in TFF format [8], and Z3 and CVC4 on a version in SMT format [3]. The original problems were also transformed into TFF and SMT in order to achieve a relevant comparison. Ordification performs well for Z3, while for Vampire and CVC4 it is good for some problems and bad for some. Fig. 8 shows how solving times are affected, and the diagrams also show great potential for time slicing, in particular for Vampire and Z3.

Hard problems solved using Ordification After Ordification, 15 problems, all from the SWV category, with

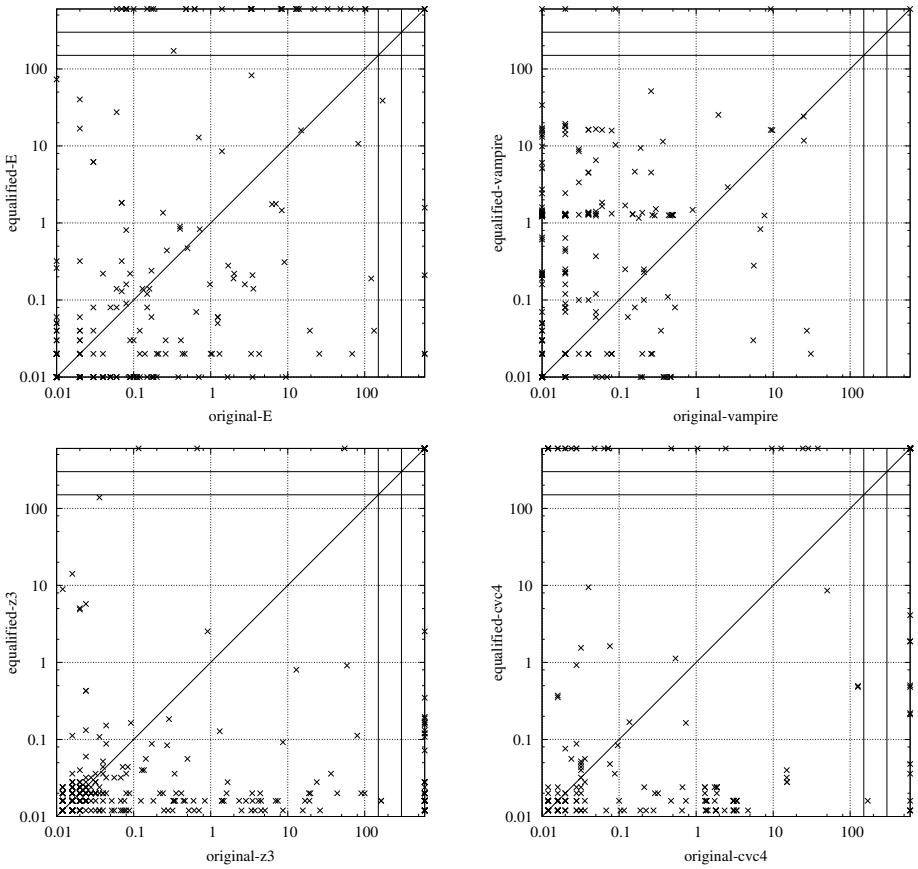

Figure 7: The time taken to solve problems, with and without equalification, using E, Vampire, Z3 and CVC4

rating 1.0 are solved. (SWV004-1, SWV035+1, SWV040+1, SWV044+1, SWV049+1, SWV079+1, SWV100+1, SWV101+1, SWV108+1, SWV110+1, SWV113+1, SWV118+1, SWV120+1, SWV124+1, SWV130+1) Vampire and Z3 each solve 14 hard problems and CVC4 solves 13 of them. Rating 1.0 means that no known current theorem prover solves the problem in reasonable time. One problem (SWV004-1) is even categorized as Unknown, which means that no prover has ever solved it. After ordification, all three provers were able to solve it in less than a second.

Maxification Maxification, the second possible treatment of total orders, turned out to be disadvantageous to E and Z3, while having very little effect on Vampire and CVC4 (see Fig. 6).

7.3 Transitive and Reflexive relations

573 test problems include relations that are transitive and reflexive, excluding equivalence relations and total orders. In all of them, transitivity occurs syntactically as an axiom. The problems come from a variety of categories, but a vast majority are in SET, SEU and SWV. Only about half of the original problems were solved by each theorem prover, which may indicate that transitivity axioms are difficult for current theorem provers to deal with. We evaluate the performance of the theorem provers before and after transification. Problems that include equivalence relations and total orders are excluded, as the corresponding methods equalification and ordification give better results and should be preferred. Vampire benefits the most from transification, and solves 32 new problems after the transformation, while 10 previously solvable problems become unsolvable within the time limit. For E, the effect of transification is almost exclusively negative. Vampire also has a significantly worse performance than E overall on the original problems that include relations that are transitive and reflexive. Both Z3 and CVC4 perform worse after the transformation, but time slicing can be a good way to improve the results.

Hard problems solved using Transification After transification, two new problems with rating 1.0 are

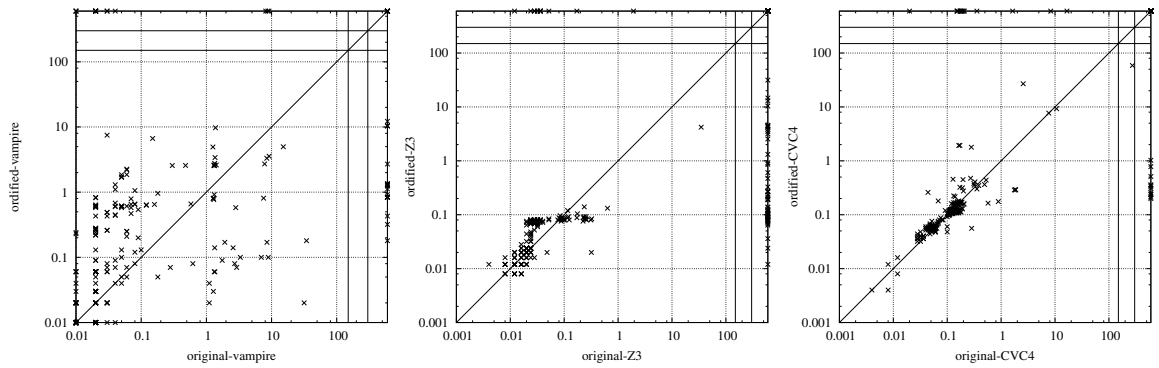

Figure 8: Effects of ordification, using Vampire, Z3 and CVC4

solved, both by Vampire (SEU322+2 and SEU372+2).

Equalification and Transification Since all equivalence relations are transitive and reflexive, the method for transification works also on equivalence relations. Comparing the two methods on the 430 problems with equivalence relations, we concluded that equalification and transification work equally bad for E, Vampire and CVC4. Both transification and equalification improve the results for Z3, but equalification does so significantly.

Ordification and Transification We compared ordification and transification on the 328 problems containing total orders. Transification seems to make these problems generally more difficult for theorem provers to solve, while ordification instead improved the results on many of the problems. Transification makes the theorem prover perform worse on these problems also for E, which cannot make use of ordification since it does not provide support for arithmetic.

8 Discussion, Conclusions, and Future Work

We have presented 5 transformations that can be applied to theories with certain transitive relations: equalification, pequalification, transification, ordification, and maxification. We have also created a method for syntactic discovery of binary relations where these transformations are applicable.

For users of reasoning tools that create their own theories, it is clear that they should consider using one of the proposed alternative treatments when writing theories. For all of our methods, there are existing theories for which some provers performed better on these theories than others. In particular, for transification and ordification, there exist 17 TPTP problems that are now solvable that weren't previously.

For implementers of reasoning tools, our conclusions are less clear. For some combinations of treatments and provers (such as transification for Vampire, and equalification for Z3), overall results are clearly better, and we would thus recommend these treatments as preprocessors for these provers. Some more combinations of treatments and provers lend themselves to a time slicing strategy that can solve strictly more problems, and could thusly be integrated in a natural way in provers that already have the time slicing machinery in place.

Related Work Chaining [1] is a family of methods that limit the use of transitivity-like axioms in proofs by only allowing certain chains of them to occur in proofs. The result is a complete proof system that avoids the derivation of unnecessary consequences of transitivity. However, chaining is not implemented in any of the reasoning tools we considered for this paper. In personal communication with some of the authors, chaining-like techniques have not been deemed important enough to be considered for implementation, and their preliminary experimental results were mostly negative.

Future Work There is a lot of room for improvements and other future work. There are many other relations that are more or less common that could benefit from an alternative treatment like the transformations described in this paper. In particular, maxification seems to be an idea that could be applied to binary relations that are weaker than total orders, which may make this treatment more effective. But there are also other, non-transitive relations that are of interest.

Ordification uses total orders as the base-transformation, and treats strict total orders as negated total orders. We would like to investigate more in which cases it may be beneficial to treat strict total orders differently from

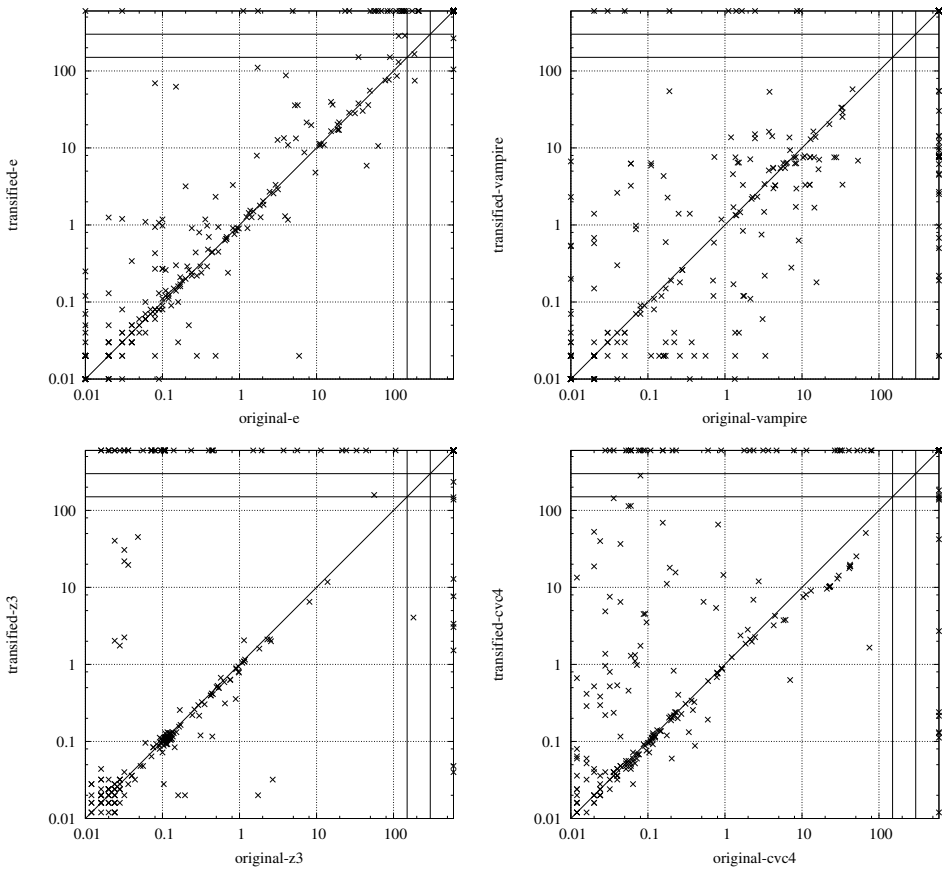

Figure 9: Effects of transification, using E, Vampire, Z3 and CVC4

total orders, and when to use $<$ on $\mathbb{R}$ instead of $\leqslant$.

We would also like to investigate the effect of our transformations on Satisfiable and CounterSatisfiable problems. What would be the effect on finite model finders? Some transformations change the number of variables per clause, which will influence the performance of finite model finders. What happens to saturation-based tools? Is it easier or harder to saturate a problem after transformation? Evaluating these questions is hard, because we did not find many satisfiable problems in the TPTP that contained relevant relations (see Fig. 3). Instead, we considered using Unsatisfiable problems, and measuring the time it takes to show the absence of models up to a certain size.

There are other kinds of relations than binary relations. For example, we can have an ternary relation that behaves as an equivalence relation in its 2nd and 3rd argument. An alternative treatment of this relation would be to introduce a binary function symbol *rep*. We do not know whether or not this occurs often, and if it is a good idea to treat higher-arity relational symbols specially in this way.

Lastly, we would like to look at how these ideas could be used inside a theorem prover; as soon as the prover discovers that a relation is an equivalence relation or a total order, one of our transformations could be applied, on the fly. The details of how to do this remain to be investigated.

References

[1] Leo Bachmair and Harald Ganzinger. Ordered chaining calculi for first-order theories of transitive relations. *Journal of the ACM (JACM)*, 45(6):1007–1049, 1998.

[2] Clark Barrett, Christopher L. Conway, Morgan Deters, Liana Hadarean, Dejan Jovanović, Tim King, Andrew Reynolds, and Cesare Tinelli. CVC4. In *Proceedings of the 23rd International Conference on Computer Aided Verification*, CAV'11, pages 171–177, Berlin, Heidelberg, 2011. Springer-Verlag.

[3] Clark Barrett, Aaron Stump, Cesare Tinelli, Sascha Boehme, David Cok, David Deharbe, Bruno Dutertre, Pascal Fontaine, Vijay Ganesh, Alberto Griggio, Jim Grundy, Paul Jackson, Albert Oliveras, Sava Krsti, Michal Moskal, Leonardo De Moura, Roberto Sebastiani, To David Cok, and Jochen Hoenicke. C.: The SMT-LIB Standard: Version 2.0. Technical report, 2010.

[4] Leonardo de Moura and Nikolaj Bjørner. Z3: An Efficient SMT Solver. In *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340. Springer, 2008.

[5] Laura Kovács and Andrei Voronkov. First-Order Theorem Proving and Vampire. In *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*, pages 1–35, 2013.

[6] Stephan Schulz. The E Theorem Prover, 2015. http://www.eprover.org/.

[7] Geoff Sutcliffe. The TPTP Problem Library, 2015. http://www.tptp.org/.

[8] Geoff Sutcliffe, Stephan Schulz, Koen Claessen, and Peter Baumgartner. The TPTP Typed First-order Form and Arithmetic. In Nikolaj Bjørner and Andrei Voronkov, editor, *The 18th International Conference on Logic for Programming Artificial Intelligence and Reasoning (LPAR-18)*, pages 406–419, Merida, Venezuela, mar 2012. Springer Verlag.