Handling Transitive Relations in First-Order Automated Reasoning

Journal of Automated Reasoning
https://doi.org/10.1007/s10817-021-09605-z

Handling Transitive Relations in First-Order Automated Reasoning

Koen Claessen[1] · **Ann Lilliestr\u00f6m**[1]

Received: 9 May 2018 / Accepted: 4 July 2021

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 *transitive relations* in general. We show how such relations can be discovered syntactically in an input theory, and how they can be expressed in alternative ways. We experimentally evaluate different such ways 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 types of binary relations.

Keywords First-order logic · Transitive relations · Automated reasoning · Transformation · Binary relations · Automated theorem proving · Transitivity

1 Introduction

This paper explores different possible ways of speeding up the handling of commonly occurring kinds of *transitive binary relations*, in the context of general first-order automated reasoning.

As an example, consider the following set of first-order axioms for a relation symbol R:

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

✉ Koen Claessen
koen@chalmers.se

Ann Lilliestr\u00f6m
annl@chalmers.se

[1] Chalmers University of Technology, Gothenburg, Sweden

A possible goal we may want to prove is:

$$\forall x.\ R(f(f(f(x))), x)$$

A quick investigation of the first three axioms reveals that R is an *equivalence relation*, which is a binary relation that occurs quite commonly in practice in first-order problems. In this paper, we suggest that an alternative way of dealing with equivalence relations is not to axiomatize them (as we did in the example), but to choose a new function symbol rep, and replace all occurrences of $R(x, y)$ with the literal $rep(x)=rep(y)$. Doing so makes the problem simpler, because the three axioms for equivalence relations (reflexivity, symmetry, and transitivity) can be removed. The resulting (complete) set of axioms is:

$$\forall x.\ rep(x) = rep(f(x))$$

The goal we need to prove becomes:

$$\forall x.\ rep(f(f(f(x)))) = rep(x)$$

As our experimental results show, this alternative way of dealing with equivalence relations presented here is beneficial for some theorem provers, because it is easier for those provers to reason about equality and an extra function symbol than about a binary predicate symbol wth additional axioms.

This paper identifies six kinds of commonly occurring binary relations (that all happen to be transitive) for which alternative treatments are proposed, that are designed to be simpler to reason about than their standard axiomatizations. The kinds of binary relations dealt with in this paper are:

- Equivalence relations, and partial equivalence relations,
- Total orders, and strict total orders,
- (general) transitive relations, with and without reflexivity.

It can be very beneficial to hardcode special ways of dealing of commonly occurring functions or relations in automated reasoning tools, as demonstrated by for example equality handling, AC handling, and arithmetic operators and relations, which have native support in many reasoning tools. However, we chose to investigate alternative ways of expressing the mentioned binary relations directly in the input problem, rather than new methods that need to be built-in to theorem provers, for several reasons: (1) it is cheaper than implementing built-in methods, simply because no extra implementation effort is required, (2) it is more flexible, because one is not tied to one particular theorem prover, and (3) it is beneficial to the user of the tool (who can choose what alternative to use) as well as the implementer of the tool (who can implement automatic transformations that choose the alternative). Such automatic transformations are also described in the paper.

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

Related Work Binary relations, and transitive relations and transitive closure in particular, have been given special treatment in automated reasoning tools in a variety of different domains. For example, Kodkod [12] and CVC4 [8] reason about relational logic for finite domains, where even transitive closure can be supported. Cristiá et al. [3] present a decision procedure for sets, relations, and partial functions that is complete for finite sets. Horrocks and Gough propose an efficient way of dealing with transitive binary relations in the context of description logics [6]. Schmidt and Hustadt present a method that translates problems in

Fig. 1 Definitions of basic properties of binary relations

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

Fig. 2 Number of occurrences of binary relation properties in TPTP

6598	reflexive
2976	transitive
2483	antisymmetric
2136	euclidean
2108	total
2108	asymmetric
1487	symmetric
80	coreflexive
(163	other)

propositional modal logic with background theories (that can also include transitivity) into first-order logic [9].

For general first-order logic, chaining [1] is a family of built-in 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. Chaining has been implemented inside one of the reasoning tools we considered for this paper (SPASS [13]). Also more specific binary relations that the ones considered in this paper have been implemented in this way [5].

In contrast, this paper presents purely preprocessing techniques for speeding up reasoning about different kinds of transitive binary relations occurring in general first-order logic. As far as we know, this is also the first paper to consider methods for dealing with (partial) equivalence relations and (strict) total orders specifically.

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. Note that the logic that we use throughout the paper is unsorted first-order logic with equality.

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[1].

When we investigated the number of occurrences of these basic properties in a subset of the TPTP problem library (v7.0.0)[2] [11], 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, no function symbols,

[1] Throughout the paper, we use the word theory to simply mean "set of formulas".

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

and, possibly, equality. Each such clause was then syntactically categorized as an expression of a basic property of a binary relation symbol. The total number of such clauses found is indicated in the table for each basic property. We found 163 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, 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 relation R, the *negated version* of R, denoted $R^{\neg}$, is defined as follows:

$$\forall x, y.\ R^{\neg}(x, y) \Leftrightarrow \neg R(x, y)$$

The *flipped version* of R, denoted $R^{\leftrightarrows}$, is defined as follows:

$$\forall x, y.\ R^{\leftrightarrows}(x, y) \Leftrightarrow R(y, x)$$

We lift these two notions to properties of relations as follows. 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 $R^{\neg}$:

$$prop^{\neg}[R] \Leftrightarrow prop[R^{\neg}]$$

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 $R^{\leftrightarrows}$:

$$prop^{\leftrightarrows}[R] \Leftrightarrow prop[R^{\leftrightarrows}]$$

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 6 kinds of different binary relations, which are defined as combinations of basic properties:

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

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 this property is clausified in the presence of transitivity, one ends up with $antisymmetric^{\neg}$ which says that at least one of $R(x, y)$, $x{=}y$, or $R(y, x)$ must be true. There

500	+48	equivalence relations
154	+2	partial equivalence relations
250	+4	(strict) total orders
806	+15	reflexive, transitive relations (excluding the above)
1128	+69	transitive relations (excluding the above)

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

Fig. 4 Basic properties that are equivalent

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

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

Our goal is to automatically choose the right treatment of equivalence relations, total orders, and general transitive relations. Thus, we must have an automatic way of identifying these relations in a given theory. It is relatively 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.

But, there are many 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 [10]) 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.

$false \Leftarrow \{reflexive, reflexive^{\neg}\}$
$false \Leftarrow \{reflexive, asymmetric\}$
$false \Leftarrow \{total, reflexive^{\neg}\}$
$false \Leftarrow \{total, asymmetric\}$
$reflexive \Leftarrow \{total\}$
$euclidean \Leftarrow \{symmetric, asymmetric\}$
$euclidean \Leftarrow \{total, symmetric\}$
$euclidean \Leftarrow \{antisymmetric, coreflexive^{\neg}\}$
$euclidean \Leftarrow \{transitive, euclidean^{\neg}\}$
$euclidean \Leftarrow \{antisymmetric, euclidean^{\neg}\}$
$euclidean \Leftarrow \{reflexive, coreflexive^{\neg}\}$
$euclidean \Leftarrow \{reflexive, euclidean^{\neg}\}$
$euclidean \Leftarrow \{coreflexive\}$
$euclidean \Leftarrow \{total, coreflexive^{\neg}\}$
$euclidean \Leftarrow \{reflexive^{\neg}, euclidean^{\curvearrowleft}\}$
$euclidean \Leftarrow \{reflexive, euclidean^{\neg\curvearrowleft}\}$
$euclidean \Leftarrow \{asymmetric, euclidean^{\curvearrowleft}\}$
$euclidean \Leftarrow \{symmetric, antisymmetric\}$
$euclidean \Leftarrow \{transitive, reflexive^{\neg}, euclidean^{\neg\curvearrowleft}\}$
$euclidean \Leftarrow \{total, euclidean^{\neg}\}$
$euclidean \Leftarrow \{transitive, coreflexive^{\neg}\}$
$euclidean \Leftarrow \{asymmetric, coreflexive^{\neg}\}$
$euclidean \Leftarrow \{asymmetric, euclidean^{\neg}\}$
$euclidean \Leftarrow \{total, euclidean^{\neg\curvearrowleft}\}$
$euclidean \Leftarrow \{antisymmetric, reflexive^{\neg}, euclidean^{\neg\curvearrowleft}\}$
$euclidean \Leftarrow \{asymmetric, euclidean^{\neg\curvearrowleft}\}$
$euclidean \Leftarrow \{symmetric, transitive\}$
$euclidean \Leftarrow \{euclidean^{\neg}, euclidean^{\curvearrowleft}\}$
$euclidean \Leftarrow \{symmetric, euclidean^{\curvearrowleft}\}$
$euclidean \Leftarrow \{reflexive, euclidean^{\curvearrowleft}\}$
$euclidean \Leftarrow \{reflexive, symmetric, antisymmetric^{\neg}\}$
$euclidean \Leftarrow \{reflexive, symmetric, transitive^{\neg}\}$
$euclidean \Leftarrow \{total, euclidean^{\curvearrowleft}\}$
$euclidean \Leftarrow \{euclidean^{\curvearrowleft}, coreflexive^{\neg}\}$
$antisymmetric \Leftarrow \{coreflexive\}$
$antisymmetric \Leftarrow \{asymmetric\}$
$antisymmetric \Leftarrow \{transitive, reflexive^{\neg}\}$
$antisymmetric \Leftarrow \{reflexive^{\neg}, euclidean^{\curvearrowleft}\}$
$antisymmetric \Leftarrow \{euclidean, reflexive^{\neg}\}$
$transitive \Leftarrow \{symmetric, asymmetric\}$
$transitive \Leftarrow \{total, symmetric\}$
$transitive \Leftarrow \{antisymmetric, coreflexive^{\neg}\}$
$transitive \Leftarrow \{coreflexive\}$
$transitive \Leftarrow \{euclidean, transitive^{\neg}\}$
$transitive \Leftarrow \{symmetric, antisymmetric\}$
$transitive \Leftarrow \{reflexive, euclidean^{\neg}\}$
$transitive \Leftarrow \{euclidean, reflexive^{\neg}\}$
$transitive \Leftarrow \{total, euclidean^{\neg}\}$
$transitive \Leftarrow \{reflexive, coreflexive^{\neg}\}$
$transitive \Leftarrow \{asymmetric, coreflexive^{\neg}\}$
$transitive \Leftarrow \{asymmetric, euclidean^{\neg}\}$
$transitive \Leftarrow \{asymmetric, transitive^{\neg}\}$
$transitive \Leftarrow \{antisymmetric, transitive^{\neg}\}$
$transitive \Leftarrow \{euclidean^{\curvearrowleft}, euclidean^{\neg\curvearrowleft}\}$
$transitive \Leftarrow \{euclidean, coreflexive^{\neg}\}$
$transitive \Leftarrow \{total, coreflexive^{\neg}\}$
$transitive \Leftarrow \{euclidean, asymmetric\}$
$transitive \Leftarrow \{total, euclidean\}$
$transitive \Leftarrow \{reflexive, euclidean^{\neg\curvearrowleft}\}$
$transitive \Leftarrow \{euclidean^{\curvearrowleft}, transitive^{\neg}\}$
$transitive \Leftarrow \{euclidean, symmetric\}$
$transitive \Leftarrow \{reflexive^{\neg}, euclidean^{\curvearrowleft}\}$
$transitive \Leftarrow \{antisymmetric, euclidean^{\neg}\}$
$transitive \Leftarrow \{total, euclidean^{\neg\curvearrowleft}\}$
$transitive \Leftarrow \{euclidean, antisymmetric^{\neg}\}$
$transitive \Leftarrow \{reflexive, euclidean^{\curvearrowleft}\}$
$transitive \Leftarrow \{reflexive, symmetric, antisymmetric^{\neg}\}$
$transitive \Leftarrow \{euclidean^{\curvearrowleft}, coreflexive^{\neg}\}$
$transitive \Leftarrow \{reflexive, symmetric, transitive^{\neg}\}$

$transitive \Leftarrow \{euclidean, antisymmetric\}$
$transitive \Leftarrow \{euclidean, euclidean^{\neg}\}$
$transitive \Leftarrow \{total, euclidean^{\curvearrowleft}\}$
$transitive \Leftarrow \{asymmetric, euclidean^{\curvearrowleft}\}$
$transitive \Leftarrow \{symmetric, euclidean^{\curvearrowleft}\}$
$transitive \Leftarrow \{asymmetric, euclidean^{\neg\curvearrowleft}\}$
$transitive \Leftarrow \{antisymmetric, euclidean^{\neg\curvearrowleft}\}$
$transitive \Leftarrow \{euclidean, reflexive\}$
$transitive \Leftarrow \{euclidean, euclidean^{\neg\curvearrowleft}\}$
$transitive \Leftarrow \{euclidean, euclidean^{\curvearrowleft}\}$
$transitive \Leftarrow \{antisymmetric, euclidean^{\curvearrowleft}\}$
$transitive \Leftarrow \{euclidean^{\curvearrowleft}, antisymmetric^{\neg}\}$
$transitive \Leftarrow \{euclidean^{\neg}, euclidean^{\curvearrowleft}\}$
$total \Leftarrow \{reflexive, coreflexive^{\neg}\}$
$total \Leftarrow \{reflexive, euclidean^{\neg\curvearrowleft}\}$
$total \Leftarrow \{reflexive, euclidean^{\neg}\}$
$total \Leftarrow \{reflexive, antisymmetric^{\neg}\}$
$total \Leftarrow \{reflexive, transitive^{\neg}\}$
$symmetric \Leftarrow \{coreflexive^{\neg}\}$
$symmetric \Leftarrow \{euclidean^{\neg}, reflexive^{\neg}\}$
$symmetric \Leftarrow \{euclidean, reflexive^{\neg}\}$
$symmetric \Leftarrow \{coreflexive\}$
$symmetric \Leftarrow \{total, euclidean^{\neg}\}$
$symmetric \Leftarrow \{total, euclidean\}$
$symmetric \Leftarrow \{euclidean, asymmetric\}$
$symmetric \Leftarrow \{asymmetric, euclidean^{\neg}\}$
$symmetric \Leftarrow \{reflexive, euclidean^{\neg\curvearrowleft}\}$
$symmetric \Leftarrow \{reflexive, euclidean^{\curvearrowleft}\}$
$symmetric \Leftarrow \{reflexive^{\neg}, euclidean^{\curvearrowleft}\}$
$symmetric \Leftarrow \{reflexive^{\neg}, euclidean^{\neg\curvearrowleft}\}$
$symmetric \Leftarrow \{total, euclidean^{\neg\curvearrowleft}\}$
$symmetric \Leftarrow \{asymmetric, euclidean^{\curvearrowleft}\}$
$symmetric \Leftarrow \{euclidean^{\neg}, euclidean^{\curvearrowleft}\}$
$symmetric \Leftarrow \{total, euclidean^{\curvearrowleft}\}$
$symmetric \Leftarrow \{euclidean, euclidean^{\neg\curvearrowleft}\}$
$symmetric \Leftarrow \{euclidean, euclidean^{\curvearrowleft}\}$
$symmetric \Leftarrow \{euclidean, reflexive\}$
$symmetric \Leftarrow \{asymmetric, euclidean^{\neg\curvearrowleft}\}$
$symmetric \Leftarrow \{euclidean^{\neg}, euclidean^{\neg\curvearrowleft}\}$
$symmetric \Leftarrow \{reflexive, euclidean^{\neg}\}$
$coreflexive \Leftarrow \{symmetric, asymmetric\}$
$coreflexive \Leftarrow \{antisymmetric, coreflexive^{\neg}\}$
$coreflexive \Leftarrow \{transitive, euclidean^{\neg}, reflexive^{\neg}\}$
$coreflexive \Leftarrow \{asymmetric, coreflexive^{\neg}\}$
$coreflexive \Leftarrow \{euclidean, reflexive^{\neg}\}$
$coreflexive \Leftarrow \{reflexive, antisymmetric, euclidean^{\neg}\}$
$coreflexive \Leftarrow \{asymmetric, euclidean^{\neg}\}$
$coreflexive \Leftarrow \{euclidean, asymmetric\}$
$coreflexive \Leftarrow \{symmetric, antisymmetric\}$
$coreflexive \Leftarrow \{total, antisymmetric, euclidean^{\neg}\}$
$coreflexive \Leftarrow \{reflexive, antisymmetric, euclidean^{\neg\curvearrowleft}\}$
$coreflexive \Leftarrow \{reflexive, antisymmetric, euclidean^{\curvearrowleft}\}$
$coreflexive \Leftarrow \{symmetric, transitive, reflexive^{\neg}\}$
$coreflexive \Leftarrow \{transitive, reflexive^{\neg}, coreflexive^{\neg}\}$
$coreflexive \Leftarrow \{antisymmetric, euclidean^{\neg}, euclidean^{\curvearrowleft}\}$
$coreflexive \Leftarrow \{total, antisymmetric, euclidean^{\curvearrowleft}\}$
$coreflexive \Leftarrow \{total, euclidean, antisymmetric\}$
$coreflexive \Leftarrow \{total, antisymmetric, euclidean^{\neg\curvearrowleft}\}$
$coreflexive \Leftarrow \{antisymmetric, reflexive^{\neg}, euclidean^{\neg\curvearrowleft}\}$
$coreflexive \Leftarrow \{reflexive^{\neg}, euclidean^{\curvearrowleft}\}$
$coreflexive \Leftarrow \{antisymmetric, euclidean^{\neg}, reflexive^{\neg}\}$
$coreflexive \Leftarrow \{asymmetric, euclidean^{\curvearrowleft}\}$
$coreflexive \Leftarrow \{asymmetric, euclidean^{\neg\curvearrowleft}\}$
$coreflexive \Leftarrow \{antisymmetric, euclidean^{\neg}, euclidean^{\neg\curvearrowleft}\}$
$coreflexive \Leftarrow \{euclidean, antisymmetric, euclidean^{\neg\curvearrowleft}\}$
$coreflexive \Leftarrow \{euclidean, antisymmetric, euclidean^{\curvearrowleft}\}$
$coreflexive \Leftarrow \{euclidean, reflexive, antisymmetric\}$
$coreflexive \Leftarrow \{transitive, reflexive^{\neg}, euclidean^{\neg\curvearrowleft}\}$

Fig. 5 The complete list of implications between properties

Secondly, we want to generate all implications of the form $\{prop_1, .., prop_n\} \Rightarrow prop$ where the set $\{prop_1, .., prop_n\}$ 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 $\{prop_1, .., prop_n\}$ from the set of all properties P–$\{prop\}$. The proce-

dure then asks E whether or not $\{prop_1, .., prop_n\} \Rightarrow prop$ is valid. If it is, then we look at the proof that E produces, and print the implication $\{prop_a, .., prop_b\} \Rightarrow prop$, where $\{prop_a, .., prop_b\}$ is the subset of properties that were used in the proof. We then also tell the constraint solver never to guess a superset of $\{prop_a, .., prop_b\}$ again. If the guessed implication can not be proven, we tell the constraint solver to never guess a subset of $\{prop_1, .., prop_n\}$ again. The procedure stops when no guesses that satisfy all constraints can be made anymore.

After the loop terminates, we remove all implications that are subsumed by others.

In order to avoid generating inconsistent sets $\{prop_1, .., prop_n\}$ (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 made by the SAT-solver 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.

Also, certain properties can be derived for a binary relation R_2 if R_2 is implied by another binary relation R_1, and R_1 has that property. This holds for reflexivity, totality and seriality. Similarly, if R_2 is antisymmetric or coreflexive, the same property can be derived for R_1. When having derived a new property of a relation in this way, we iterate the procedure of finding implied properties using the precomputed table until no new information is gained. In this way, we never perform full theorem proving, but we nonetheless detect many binary relations with the listed basic properties.

In the following three sections, we describe how to deal with equivalence relations, total orders, and general transitive relations, respectively.

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:

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

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 proposed 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 reasoning about the equivalence relation 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:

Theorem 1 (Correctness of equalification) *Two theories H and H' that respectively match the LHS and RHS of equalification, are equisatisfiable.*

Proof ($\Rightarrow$) Assume we have $m \models H$, then $m\ (R)$[3] is an equivalence relation. Let the interpretation m' interpret all existing symbols as m does. Moreover, let $m'\ (rep)$ be a representative function of the equivalence relation $m\ (R)$. This means that we have $m' \models (R(x, y) \Leftrightarrow rep(x) = rep(y))$, which means that m' is a model of H'.

($\Leftarrow$) Assume we have $m' \models H'$. Let the interpretation m interpret all existing symbols as m' does. Moreover, let $(x, y) \in m(R)$ precisely when $m'(rep)(x)=m'(rep)(y)$. The relation $m(R)$ is clearly reflexive, symmetric, and transitive, and therefore we have $m \models H$. □

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?

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.

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

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:

Theorem 2 (Correctness of pequalification) *Two theories H and H' that respectively match the LHS and RHS of pequalification, are equisatisfiable.*

Proof ($\Rightarrow$) Assume we have $m \models H$, then $m(R)$ is a partial equivalence relation. Let the interpretation m' interpret all existing symbols as m does. Moreover, let $m'(P)$ be the set of domain elements x for which we have $(x, x) \in m(R)$, i.e. the subset of the domain where R is an actual equivalence relation. Let $m'(rep)$ be a representative function of the partial equivalence relation $m(R)$. This means that we have $m' \models (R(x, y) \Leftrightarrow (P(x) \wedge P(y) \wedge rep(x) = rep(y)))$, which means that m' is a model of H'.

($\Leftarrow$) Assume we have $m' \models H'$. Let the interpretation m interpret all existing symbols as m' does. Moreover, let $(x, y) \in m(R)$ precisely when $m'(P)(x) \wedge m'(P)(y) \wedge m'(rep)(x)=m'(rep)(y)$. The relation $m\ (R)$ is clearly symmetric and transitive, and therefore we have $m \models H$. □

[3] We write $m(R)$ for the interpretation of symbol R in the model m.

Intuitively, one can see that this transformation is correct by realising that the elements on which the relation R is not reflexive cannot be related to any other elements. This is because $R(x, y)$ together with symmetry and transitivity gives us $R(x, x)$. Thus, when we encounter $R(x, y)$ in the LHS theory, we know that both x and y are in the set defined by P. (This holds also when x equals y). Since R is an equivalence relation on this set, we can use the transformation of pure equivalence relations on the subset P to get $P(x) \wedge P(y) \Rightarrow rep(x) = rep(y)$.

5 Handling Total Orders

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

$$R : A \times A \to Bool$$

We now introduce a new injective function:

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

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

$$\begin{array}{lll} R\ total & & \\ R\ antisymmetric & \rightarrowtriangle & rep\ injective \\ R\ transitive & & \\ T\ [..R(x, y)..] & & T\ [..rep(x) \leqslant rep(y)..] \end{array}$$

(Here, $\leq$ 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:

Theorem 3 (Correctness of ordification) *Two theories H and H' that respectively match the LHS and RHS of ordification, are equisatisfiable.*

Proof ($\Rightarrow$) If we have $m \models H$, then without loss of generality (by Löwenheim-Skolem), we can assume that the domain of m is countable. Also, $m(R)$ is a total order. Let the interpretation m' interpret all existing symbols as m does. We now construct $m'(rep)$ recursively as a mapping from the model domain to $\mathbb{R}$, such that we have $(x, y) \in m(R)$ precisely when $m'(rep)\ (x) \leq m'(rep)\ (y)$, in the following way. Let $\{a_0, a_1, a_2, ..\}$ be the domain of the model, and set $m'(rep)\ (a_0) := 0$. For any $n>0$, pick a value for $m'(rep)\ (a_n)$ that is consistent with the total order R and all earlier domain elements a_i, for $0 \leq i<n$. 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 $m'(rep)$ is injective and we also have a model m' of H'.

($\Leftarrow$) Assume we have $m' \models H'$. Let the interpretation m interpret all existing symbols as m' does. Moreover, let $(x, y) \in m(R)$ precisely when $m'(rep)\ (x) \leq m'(rep)\ (y)$. It is clear that $m(R)$ is total and transitive, and also antisymmetric because $m'(rep)$ is injective, and therefore $m \models H$. □

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 injectivity The transformation requires an axiom that expresses that rep is injective. There are two natural ways in which this can be expressed. Here is a direct axiom:

$$\forall x, y.\ rep(x) = rep(y) \Rightarrow x = y$$

And here is an axiom that makes use of a helper function rep^{-1} which plays the role of reps inverse:

$$\forall x.\ rep^{-1}(rep(x)) = x$$

These two are logically equivalent.

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

$$\begin{array}{ll} R\ antisymmetric^{\neg} & \\ R\ asymmetric & \rightarrowtriangle\quad rep\ injective \\ R\ transitive & \\ T\ [..R(x,y)..] & \quad\ T\ [..rep(x) < rep(y)..] \end{array}$$

However, the transformation for total orders already covers this case! Any strict total order R is also recognized as a total order $R^{\neg}$, and ordification already transforms such theories in the correct way. The only difference is that $R(x, y)$ is replaced with $\neg(rep(x) \leq rep(y))$ instead of $rep(x) < rep(y)$, which is satisfiability-equivalent. (We found no performance difference in practice between these choices.)

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:

$$\begin{array}{ll} R\ total & \quad max\ associative \\ R\ antisymmetric & \rightarrowtriangle\quad max\ commutative \\ R\ transitive & \quad \forall x, y.\ max(x, y) = x \vee max(x, y) = y \\ T\ [..R(x,y)..] & \quad\ T\ [..max(x, y) = y..] \end{array}$$

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:

Theorem 4 (Correctness of maxification) *Two theories H and H' that respectively match the LHS and RHS of maxification, are equisatisfiable.*

Proof ($\Rightarrow$) If we have $m \models H$, then $m(R)$ must be a total order. Let the interpretation m' interpret all existing symbols as m does. Let $m'(max)$ be the maximum function associated with the total order $m(R)$. The function $m'(max)$ is associative and commutative, and we have that $m'(max)(x, y) = x \vee m'(max)(x, y) = y$. Moreover, we have $m' \models R(x, y) \Leftrightarrow max(x, y) = y$. Thus we also have a model m' of H'.

($\Leftarrow$) Assume we have $m' \models H'$. Let the interpretation m interpret all existing symbols as m' does. Moreover, let $(x, y) \in m(R)$ precisely when $m'(max)(x, y) = y$. Now, $m(R)$ is total (because of $m'(max)(x, y) = x \vee m'(max)(x, y) = y$), antisymmetric (because of commutativity of $m'(max)$), and transitive (because of associativity of $m'(max)$), and therefore $m \models H$. □

6 Handling Transitive Relations in General

The treatments introduced so far all make use of built-in concepts of the reasoning tool, and they can be applied only to special cases of transitive relations. In this section we propose a more general approach, in which theories with a transitivity axiom are transformed into theories without that transitivity axiom. To this end, transitivity is specialized at each *positive occurrence* of the relational symbol. Such transformations 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 transformations presented in this subsection only work 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.

Detransification A general way of handling any transitive relation R is to create a new symbol Q and replace all positive occurrences of R with a formula involving Q (see below, positive occurrences are denoted by $+R$); the negative occurrences are simply replaced by $\neg Q$:

$$\begin{array}{l} R \textit{ transitive} \\ T\,[\,..\,{+R}(x,y)\,..\qquad \rightarrow \qquad T\,[\,..\,(Q(x,y)\,\wedge\,(\forall r.\,Q(r,x)\Rightarrow Q(r,y)))\,.. \\ \quad\;\; \neg R(x,y)\;..\,] \qquad\qquad\qquad\quad \neg Q(x,y) \qquad\qquad\qquad\qquad ..\,] \end{array}$$

We call this transformation *detransification*. It can be applied to any theory that involves a transitivity axiom. The transformation removes the transitivity, but adds for every positive occurrence of $R(x, y)$ 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:

Theorem 5 (Correctness of detransification) *Two theories H and H' that respectively match the LHS and RHS of detransification, are equisatisfiable.*

Proof ($\Rightarrow$) If we have $m \models H$, then $m\,(R)$ is transitive. Let the interpretation m' interpret all existing symbols as m does. Moreover, let $m'\,(Q)(x, y) := m\,(R)(x, y)$. We have to show that $m'\,(R)(x, y)$ implies $m'\,(Q)(x, y)$, which is trivial, and that $m'\,(R)(x, y)$ implies $\forall r.\ m'(Q)(r, x) \Rightarrow m'(Q)(r, y)$, which is indeed the case because $m'(R)$ is transitive. Thus we have $m' \models H'$.

($\Leftarrow$) Assume we have $m' \models H'$. Let the interpretation m interpret all existing symbols as m' does. Moreover, let $(x, y) \in m\,(R)$ precisely when $m'\,(Q)(x, y) \wedge \forall r.\ m'\,(Q)(r, x) \Rightarrow m'(Q)(r, y)$. We have to show that $\neg m(Q)(x, y)$ implies $\neg m(R)(x, y)$, which is the same as showing that $m\,(Q)(x, y) \wedge \forall r.\ m\,(Q)(r, x) \Rightarrow m\,(Q(r, y)$ implies $m\,(Q)(x, y)$. $m\,(R)$ is also transitive (by transitivity of implication). Thus we also have $m \models H$. □

Detransification can be seen as performing one resolution step with each positive occurrence of the relation and the transitivity axiom. A positive occurrence $R(a, b)$ of a transitive relation R, resolved with the transitivity axiom $R\,(x, y) \& R\,(y, z) \Rightarrow R\,(x, z)$ becomes $R(x, a) \Rightarrow R(x, b)$ under the substitution y := a, z := b.

Detransification with reflexivity Detransification can be simplified for transitive relations that are also reflexive. In particular, we can simplify the formula with which we replace positive occurrences of the relation symbol R:

$$\begin{array}{lcl} R\ \mathit{reflexive} & & Q\ \mathit{reflexive} \\ R\ \mathit{transitive} & \multimap & \\ T\ [..\ +R(x,y)\ .. & & T\ [..(\forall r.\ Q(r,x) \Rightarrow Q(r,y))\ .. \\ \quad\ \neg R(x,y)\ ..] & & \quad \neg Q(x,y) \qquad\qquad ..] \end{array}$$

We now *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. The part that we omit here (namely $Q(x, y)$) is implicitly implied by the fact that R is reflexive.

Similarly to the detransification transformation above, Q does not have to be transitive in the RHS theory. Nonetheless, the transformation is correct, meaning that it preserves (non-)satisfiability:

Theorem 6 (Correctness of detransification with reflexivity) *Two theories H and H' that respectively match the LHS and RHS of detransification with reflexivity, are equisatisfiable.*

Proof ($\Rightarrow$) If we have $m \models H$, then $m(R)$ is reflexive and transitive. Let the interpretation m' interpret all existing symbols as m does. Moreover, let $(x, y) \in m'(Q)$ precisely when $m(R)(x, y)$. $m'(Q)$ is obviously reflexive. We have to show that $m'(R)(x, y)$ implies $\forall r.\ m'(Q)(r, x) \Rightarrow m'(Q)(r, y)$, which is indeed the case because $m'(R)$ is transitive. Thus we have $m' \models H'$.

($\Leftarrow$) Assume we have $m' \models H'$, then $m'(Q)$ is reflexive. Let the interpretation m interpret all existing symbols as m' does. Moreover, let $(x, y) \in m(R)$ precisely when $\forall r.\ m'(Q)(r, x) \Rightarrow m'(Q)(r, y)$. $m(R)$ is reflexive (by reflexivity of implication) and transitive (by transitivity of implication). We have to show that $\neg m(Q)(x, y)$ implies $\neg m(R)(x, y)$, which is the same as showing that $\forall r.\ m(Q)(r, x) \Rightarrow m(Q)(r, y)$ implies $m(Q)(x, y)$, which is true because $m(Q)$ is reflexive. Thus we also have $m \models H$. □

7 Experimental Results

We evaluate the effects of the different axiomatizations using three different resolution based theorem provers, E 2.0 [10] (with the *xAuto* and *tAuto* options), Vampire 4.0 [7] (with the *casc mode* option), Spass 3.9 [13] (with the *Auto* option, which activates chaining in the presence of transitive predicates), and two SMT-solvers, Z3 4.5 [4] and CVC4 1.5 [2]. The experiments were performed on a PC with a 2xQuad Core Intel Xeon E5620 processor with 24 GB physical memory, running at 2.4 GHz, with Ubuntu 12.04. We use a time limit of 5 min on each problem.

We started from a set of 13410 test problems from the TPTP, listed as Unsatisfiable, Theorem or Unknown or Open (leaving out the very large theories)[4]. 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 theories were produced for these problems. In total, 2007 problems were found to include one or more transitive relations and could thus be used with at least one of the presented transformations. 130 of these problems are listed as Unknown, and an additional 172 problems have rating 1.0. No problem in the resulting set of problems is listed as Open.

The experimental results are summarized in Fig. 6.

[4] Theories with more than 10.000 clauses were considered "very large" and we chose to leave them out of these experiments, because they were impractical to deal with. We have also evaluated the transformations on

		E			Vampire			Spass		
equalification	(436)	427	+5	-38	434	+0	-7	385	+16	-46
with idempotency			+4	-56		+0	-6		+14	-43
pequalification	(617)	524	+5	-69	526	+5	-8	452	+18	-48
with idempotency			+3	-95		+6	-13		+16	-45
ordification	(326)	272	n/a	n/a	295	+1	-0	243	n/a	n/a
detransification	(2007)	1483	**+16**	-97	1542	**+39**	-21	1200	+91	-91
with reflexivity	(1359)	1025	+6	-89	1062	**+32**	-20	866	+29	-90

		Z3			CVC4		
equalification	(436)	354	+60	-7	370	+35	-20
with idempotency			+59	-11		+35	-22
pequalification	(617)	392	+66	-8	343	+45	-25
with idempotency			+61	-21		+44	-26
ordification	(326)	236	**+50**	-1	267	+23	-1
detransification	(2007)	1196	+95	-33	1375	+73	-31
with reflexivity	(1359)	870	+66	-135	999	+40	-110

Fig. 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

Overall, the results vary between each transformation and reasoning tool. For many of the transformations, we do not gain any solved problems without also losing some. A time-slicing strategy can be advantageous, were the reasoning tool is run on both the original and the transformed problem, with a suitably chosen time-limit for each. Z3 turns out to work well on ordified problems, where it can make use of its built in strategies for arithmetic. E did not benefit from any of the transformations, and a large portion of the problems became unsolvable. One may have expected better results for equalification, since introducing equality in place of each occurrence of an equivalence relation seems suitable for an equational theorem prover. However, E performs well already on the untreated problems with equivalence relations, leaving little room for improvement. Vampire has the least difference in performance before and after the transformations.

In order to make a comparison between the transformations and evaluate what transformation works best for each kind of transitive relation, we partition the test problems into different subsets (Fig. 7). These subsets are defined by the discovered properties of the transitive relation. A problem can appear in several subsets if the problem includes several transitive relations having different properties. This is the case for 156 problems. Apart from such special cases, the subsets are disjoint. Firstly, we divide the problems into two sets, one where the transitive relation is found to be total (or strictly total, as in the case of a negated total order), and one where this was not the case. We use the notation P^C to denote the subset of problems with transitive relations with no syntactic evidence of the property P.

Satisfiable/Countersatisfiable problems, but there were too few problems for the results to be significant, and the results were also mostly negative.

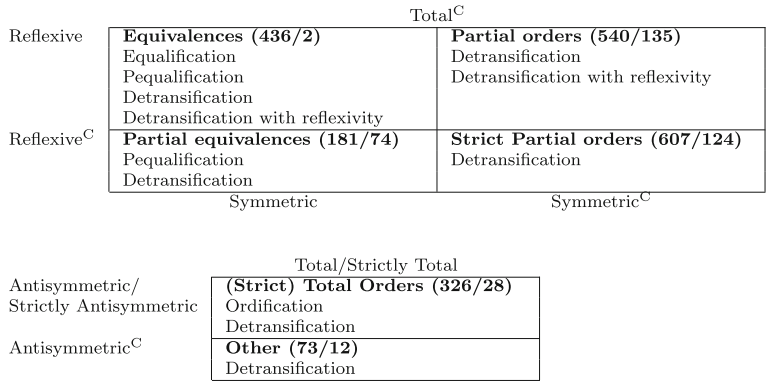

Fig. 7 Partitioning of test problems and their applicable transformations

The problems in Total^C are further divided into four groups, depending on if they contain a transitive relation that is found to be reflexive and/or symmetric. The problems containing a total relation are partitioned into two sets: problems with one or more total order (i.e. total, transitive and antisymmetric), and problems with relations that are total and transitive but lack the antisymmetry property (labelled as "other" in the diagram). Fig. 7 shows each subset with its number of problems and number of rating 1 problems, and the transformations that are applicable for that subset. For example, a problem with a transitive relation that is in Total^C, Reflexive^C and Symmetric has the applicable transformations Pequalification and Detransification, as shown in the bottom left corner of the diagram. The number of rating 1 problems in each subset can give an indication of the difficulty of dealing with different kinds of transitive relations. Problems with equivalence relations are typically less difficult than problems with partial equivalence relations, however it is hard to tell if the difficulty of a problem is related to the transitive relation or has other reasons.

7.1 Detransification

Detransification is the only transformation applicable on all 2007 test problems with transitive relations. As can be seen in Fig. 6, the benefits of detransification varies with the theorem prover and problem. The SMT-solvers profit the most from this transformation, however, big differences can be seen in the different subsets. Figure 18 presents an overview of the effects on solving times for each theorem prover in the evaluation. For all of the theorem provers, detransification lets us prove some problems that we could not previously, but some problems also become unsolvable within the time limit.

Figure 8 presents the results of detransification on each of the subsets defined in Fig. 7. Here we can get an indication of what problems detransification is useful for and what kind of problems tend to become harder after the transformation. For E, the results generally become worse after detransification, even though some new problems become solvable, including one with rating 1. For Vampire, detransification improves the results on problems with partial orders and partial equivalence relations. These subsets have a relatively low success rate before the transformations. On the other subsets, the results stay fairly stable. For Spass, partial equivalences and strict partial orders benefit the most from detransification, while the other subsets show mixed results. Both SMT-solvers, Z3 and CVC4, show improved results on the transformed equivalence relations and partial equivalence relation. The theorem provers performed well on these problems already before the transformation and thus

Equivalences (436)			
E	427	+4	-24
Vampire	434	+0	-0
Spass	385	+15	-32
CVC4	370	+31	-8
Z3	354	+54	-8

Partial Orders (540)			
E	281	+4	-15
Vampire	287	**+34**	-6
Spass	201	+15	-16
CVC4	292	**+13**	-10
Z3	215	+19	-10

Partial Equivalences (181)			
E	97	**+1**	-9
Vampire	92	+5	-0
Spass	67	+8	-0
CVC4	51	+16	-0
Z3	38	+18	-0

Strict Partial Orders (607)			
E	421	**+5**	-32
Vampire	444	+5	-11
Spass	299	+48	-10
CVC4	341	+29	-7
Z3	290	+24	-12

Total Orders (326)			
E	272	+3	-20
Vampire	296	+0	-2
Spass	243	+12	-33
CVC4	289	+1	-3
Z3	255	+0	-1

Other (73)			
E	57	+0	-7
Vampire	58	+0	-2
Spass	47	+2	-0
CVC4	55	+2	-3
Z3	51	+0	-2

Fig. 8 The effect of detransification on each subset (original number of problems solved; number of extra problems after the transformation; number of lost problems)

Equivalences (436)			
E	427	+3	-30
Vampire	434	+0	-4
Spass	385	+14	-44
CVC4	370	+32	-45
Z3	354	+57	-22

Partial Orders (540)			
E	281	+1	-20
Vampire	287	**+32**	-8
Spass	201	+8	-16
CVC4	292	+5	-55
Z3	215	+12	-41

Total Orders (326)			
E	272	+2	-34
Vampire	296	+0	-2
Spass	243	+5	-31
CVC4	289	+3	-8
Z3	255	+0	-71

Other (73)			
E	57	+0	-7
Vampire	58	+0	-6
Spass	47	+2	-0
CVC4	55	+2	-4
Z3	51	+0	-3

Fig. 9 The effect of detransification with reflexivity on each applicable subset (original number of problems solved; number of extra problems after the transformation; number of lost problems)

have less room for improvement. For partial orders and strict partial orders, the results are mixed, however more is gained than what is lost. Total orders and other transitive relations do not benefit from detransification using any of the reasoning tools in our evaluation.

7.2 Detransification with Reflexivity

The use of detransification with reflexivity is limited to transitive relations that are reflexive. It shows worse results than detransification without reflexivity on all applicable subsets for all of the tested tools. This is especially true for the SMT-solvers, for which many problems become unsolvable. The results on each applicable subset is shown in Fig. 9

7.3 Pequalification

Pequalification is applicable on any relation that is transitive and symmetric, i.e. both equivalence relations and partial equivalence relations. The results on these subsets are presented in Fig. 10, which also shows the results of the variant of pequalification with the added idempotency axiom. SMT-solvers benefit the most from pequalification, especially Z3 on the set of problems with equivalence relations. All of the tested reasoning tools perform about the same or worse given this extra axiom.

Comparing pequalification with detransification, detransification is clearly much better for partial equivalence relations, while for equivalence relations it is not as clear which trans-

Equivalences (436)			
E	427	+5/+3	-33/-59
Vampire	434	+0/+0	-2/-7
Spass	385	+16/+14	-46/-40
CVC4	370	+36/+35	-21/-22
Z3	354	+59/+56	-4/-17

Partial Equivalences (181)			
E	97	+0/+0	-36/-36
Vampire	92	+5/+6	-6/-6
Spass	67	+2/+5	-2/-2
CVC4	51	+9/+9	-4/-4
Z3	38	+9/+7	-4/-4

Fig. 10 The effect of pequalification/pequalification with idempotency on each applicable subset (original number of problems solved; number of extra problems after the transformation; number of lost problems)

Equivalences (436)			
E	427	+5/+4	-38/-56
Vampire	434	+0/+0	-7/-6
Spass	385	+16/+14	-46/-43
CVC4	370	+35/+35	-20/-22
Z3	354	+60/+59	-7/-11

Fig. 11 The effect of equalification/equalification with idempotency on the applicable subset (original number of problems solved; number of extra problems after the transformation; number of lost problems)

formation one should pick. Pequalification without idempotency seems to be a good choice on equivalence relations for SMT-solvers, especially for Z3.

7.4 Equalification

Equalification is applicable only to problems that contain equivalence relations. It performs slightly worse or about the same compared to pequalification, which is more general. Like pequalification, it shows the best results combined with the SMT solvers. Adding an idempotency axiom typically makes the results slightly worse, with E showing the most significant change. The results of equalification and equalification with idempotency are presented in Fig. 11.

7.5 Ordification

Since ordification uses arithmetic, it is only applicable with Vampire (in TFF format) and Z3 and CVC4 (in SMT format). The original problems were transformed into TFF and SMT as well, in order to achieve a fair comparison, avoiding the effects that the change of input format may have on the results. Ordification is applicable only on the set of problems containing total orders. Ordification improved the results significantly for both Z3 and CVC4, while Vampire performs about the same as before the transformation.

We can compare ordification with detransification, which is the only other transformation that is applicable on total orders. Similarly to ordification, detransification does not have any significant impact on the results for either of CVC4 or Vampire on total orders. For Z3, ordification is clearly the best choice. Note however that Z3 shows worse results than CVC4

Total Orders (326)			
Vampire	295	+1	-0
CVC4	288	+1	-1
Z3	254	+50	-1

Fig. 12 The effect of ordification on the applicable subset (original number of problems solved; number of extra problems after the transformation; number of lost problems)

Equivalences (436)	
original	427
equalification	394 (+5 - 38)
pequalification	399 (+5 -33)
detransification	407 (+4 -24)
original 270 / equalification 10	434 (+7 -0)
pequalification 10 / detransification 10	
original 75 / equalification 75	434 (+7 -6)
pequalification 75 / detransification 75	

Partial Orders (540)	
original	281
detransification	270 (+4 -15)
original 290 / detransification 10	284 (+3 -0)
original 150 / detransification 150	282 (+4 -3)

Fig. 13 Results of E by subset and strategy (number of problems solved; number of extra/lost problems in parentheses)

Fig. 14 Results of Vampire by subset and strategy (number of problems solved; number of extra/lost problems in parentheses)

Partial Equivalences (181)	
original	92
pequalification	91 (+5 -6)
detransification	97 (+5 -0)
original 290 / detransification 10	92 (+0 -0)
original 150 / detransification 150	93 (+1 -0)

Partial Orders (540)	
original	287
detransification	315 (+34 -6)
original 290 /detransification 10	301 (+14 -0)
original 150 / detransification 150	308 (+24 -3)

and Vampire prior to the transformation. After ordification, the three tools solve about the same number of problems. The results are presented in Fig. 12.

7.6 Problems with more than One Kind of Transitive Relation

156 of the problems in our evaluation contain more than one kind of transitive relation. 140 of them contain a partial equivalence relation and a strict partial order. 14 contain an equivalence relation and a partial order, and two problems contain a partial equivalence relation and a relation that is total and transitive. Almost half of these problems are hard, with rating 1 in the TPTP.

For the 140 problems with a partial equivalence and a strict partial order, we found that applying detransification to all of the transitive relations gave the best results. For the 14 problems with equivalence relation and a partial order, applying equalification on the equivalence relation and detransification on the partial order was the best, in particular for the SMT-solvers. The 2 remaining problems with multiple kinds of transitive relations are both labelled as Unknown, and were not solved before nor after any choice of transformation.

Fig. 15 Results of Spass by subset and strategy (number of problems solved; number of extra/lost problems in parentheses)

Equivalences (436)	
original	385
equalification	355 (+16 - 46)
detransification	392 (+15 -8)
original 270 / equalification 10 pequalification 10 / detransification 10	403 (+18 -0)
original 75 / equalification 75 pequalification 75 / detransification 75	398 (+18 -5)

Partial Equivalences (181)	
original	67
pequalification	61 (+0 -36)
detransification	75 (+8 -0)
original 280 / pequalification 10 /detransification 10	69 (+3 -1)
original 100 / pequalification 100 /detransification 100	75 (+8 -0)

Partial Orders (540)	
original	201
detransification	200 (+15 -16)
original 290 / detransification 10	208 (+7 -0)
original 150 / detransification 150	204 (+11 -8)

Strict Partial Orders (607)	
original	299
detransification	337 (+48 -10)
original 290 /detransification 10	322 (+24 -1)
original 150 /detransification 150	339 (+43 - 3)

Total Orders (326)	
original	243
detransification	222 (+12 -33)
original 290 / detransification 10	245 (+2 -0)
original 150 / detransification 150	251 (+12 -4)

Other (73)	
original	47
detransification	49 (+2 -0)
original 290 /detransification 10	47 (+0 -0)
original 150 /detransification 150	48 (+1 -0)

7.7 Time-Slicing

As can be seen in Figs. 18, 19, 20, 21 and 22, problems are typically solved within the first half of the 5 min time-limit or less. By splitting the time equally between the original version of the problem and the applicable transformations can thus in many cases improve the success rate. This was the case for Spass, CVC4 and Z3. For Vampire (which has its own built-in time-slicing strategies), many problems were solved towards the end of the 5 min, and time-slicing between transformations was thus less favourable. For E, whose advantages of the transformations were quite limited, the problems that did become solvable after a transformation were solved in a relatively short time. The best results were achieved by allowing 10 s on each applicable transformation, and the remaining time on the original problem.

Fig. 16 Results of Z3 by subset and strategy (number of problems solved; number of extra/lost problems in parentheses)

Equivalences (436)	
original	354
equalification	407 (+60 - 7)
pequalification	409 (+59 -4)
detransification	400 (+54 -8)
original 270 / equalification 10 pequalification 10 /detransification 10	417 (+63 -0)
original 75 / equalification 75 pequalification 75 /detransification 75	417 (+64 -1)

Partial Equivalences (181)	
original	38
pequalification	43 (+9 -4)
detransification	56 (+18 -0)
original 280 /detransification 10 pequalification 10	57 (+19 -0)
original 100 /detransification 100 pequalification 100	59 (+21 -0)

Partial Orders (540)	
original	215
detransification	224 (+19 -10)
original 290 /detransification 10	229 (+14 -0)
original 150 /detransification 150	230 (+18 -3)

Strict Partial Orders (607)	
original	290
detransification	302 (+24 -12)
original 290 /detransification 10	312 (+22 -0)
original 150 /detransification 150	311 (+23 -2)

Total Orders (326)	
original	255
detransification	254 (+0 -1)
ordification	285 (+31 -1)
original 290 /detransification 10 ordification 10	282 (+27 -0)
original 100 /detransification 100 ordification 100	285 (+30 -0)

Other (73)	
original	51
detransification	49 (+0 -2)
original 290 /detransification 10	51 (+0 -0)
original 150 /detransification 150	51 (+0 -0)

7.8 Optimal Strategies

We present for each tool an optimal strategy, that is given by identifying for each subset the transformation or combination of transformations that maximises the total number of solved problems. Since the results are based on the limited set of problems in the current TPTP library, we do not provide a universal method, but rather an idea of how parameters can be tuned to improve the results.

In Sect. 7.5, we transformed the original problem into the same format as the ordified problem (SMT format for Z3 and CVC4, and TFF format for Vampire), to focus on the effects of ordification alone. In this section we are concerned with how the results of the problems in the TPTP library can be improved. We therefore compare all results of the transformations (including ordification) with the original TPTP problem given in CNF-format. This makes a difference for Z3, which performs better on the original problems in CNF format than

Fig. 17 Results of CVC4 by subset and strategy (number of problems solved; number of extra/lost problems in parentheses)

Equivalences (436)	
original	370
equalification	385 (+35 - 20)
pequalification	385 (+36 -21)
detransification	393 (+31 -8)
original 270 / equalification 10	410 (+40 -0)
pequalification 10 /detransification 10	
original 75 / equalification 75	412 (+42 -0)
pequalification 75 /detransification 75	

Partial Equivalences (181)	
original	51
pequalification	56 (+9 -4)
detransification	67 (+16 -0)
original 280 / detransification 10	65 (+14 -0)
pequalification 10	
original 100 / detransification 100	67 (+16 -0)
pequalification 100	

Partial Orders (540)	
original	292
detransification	295 (+13 -10)
original 290 / detransification 10	302 (+10 -0)
original 150 / detransification 150	304 (+12 -0)

Strict Partial Orders (607)	
original	341
detransification	363 (+29 -7)
original 290 / detransification 10	360 (+19 -0)
original 150 / detransification 150	368 (+29 -1)

Total Orders (326)	
original	289
detransification	287 (+1 -3)
ordification	288 (+0 -1)
original 290 / detransification 10	290 (+1 -0)
ordification 10	
original 100 / detransification 100	290 (+1 -0)
ordification 100	

Other (73)	
original	55
detransification	54 (+2 -3)
original 290 /detransification 10	56 (+1 -0)
original 150 /detransification 150	56 (+2 -1)

the problems transformed into SMT. This is the reason why Z3 solves 31 new problems after ordification in Fig. 16 but 50 new problems in Fig. 12. We omit detransification with reflexivity and the idempotency version of equalification and pequalification in the diagrams below, as these transformations did not contribute to the overall results.

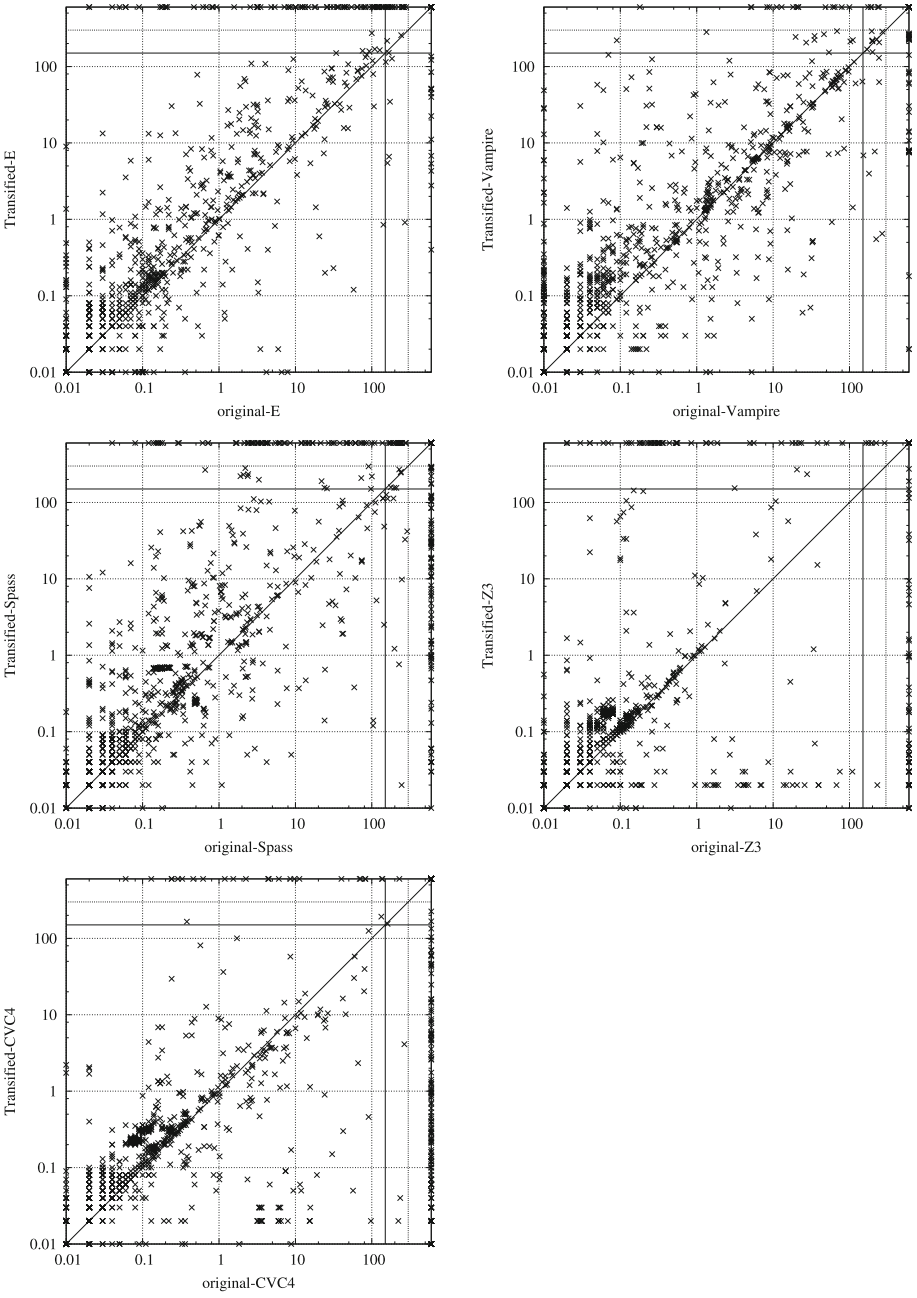

Fig. 18 Time (in s) taken to solve problems, with and without transification, using E, Vampire, Spass, Z3 and CVC4. Every mark (×) indicates a problem run before and after the transformation

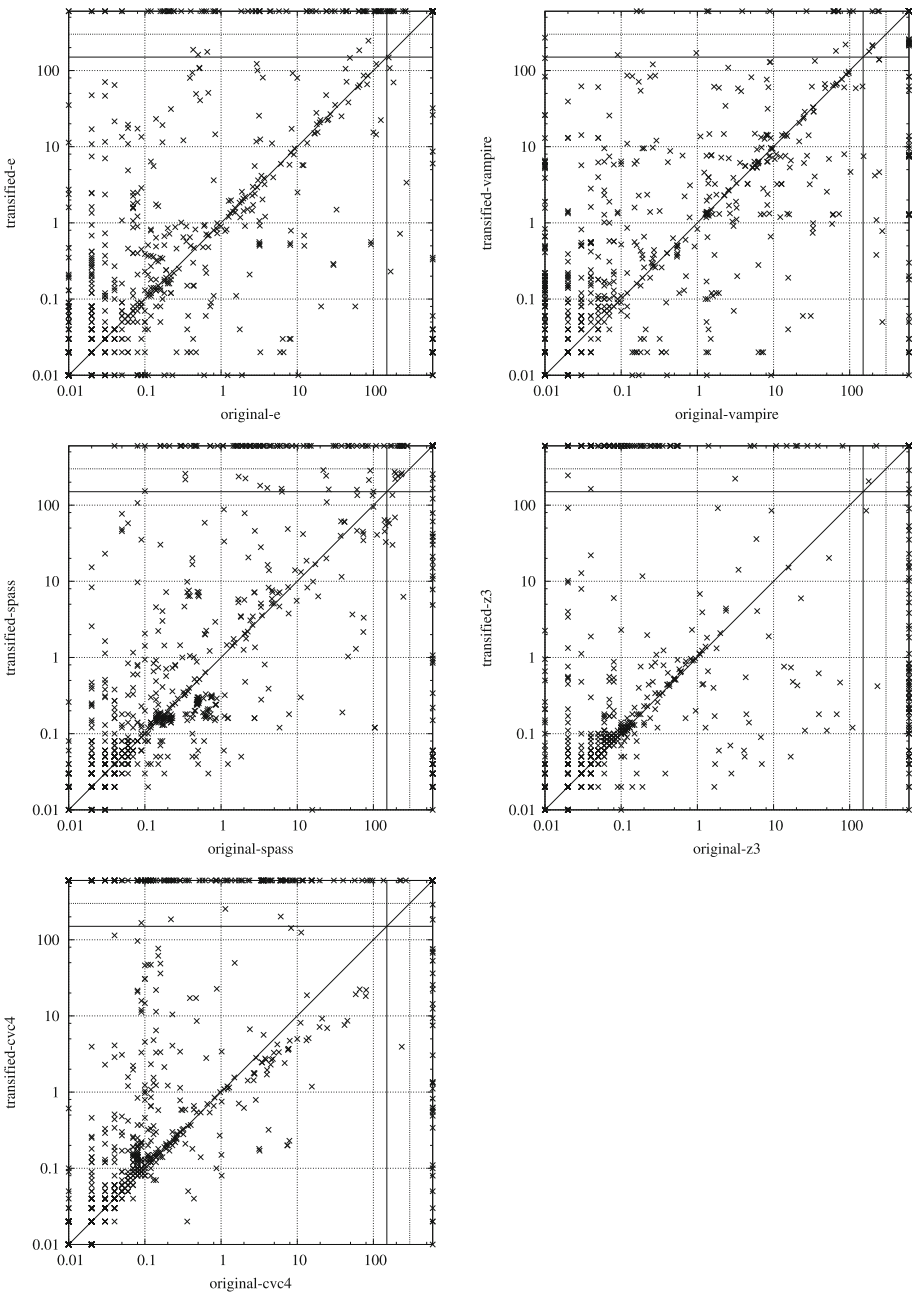

Fig. 19 Time (in s) taken to solve problems, with and without transification with reflexivity, using E, Vampire, Spass, Z3 and CVC4. Every mark (×) indicates a problem run before and after the transformation. Times in seconds

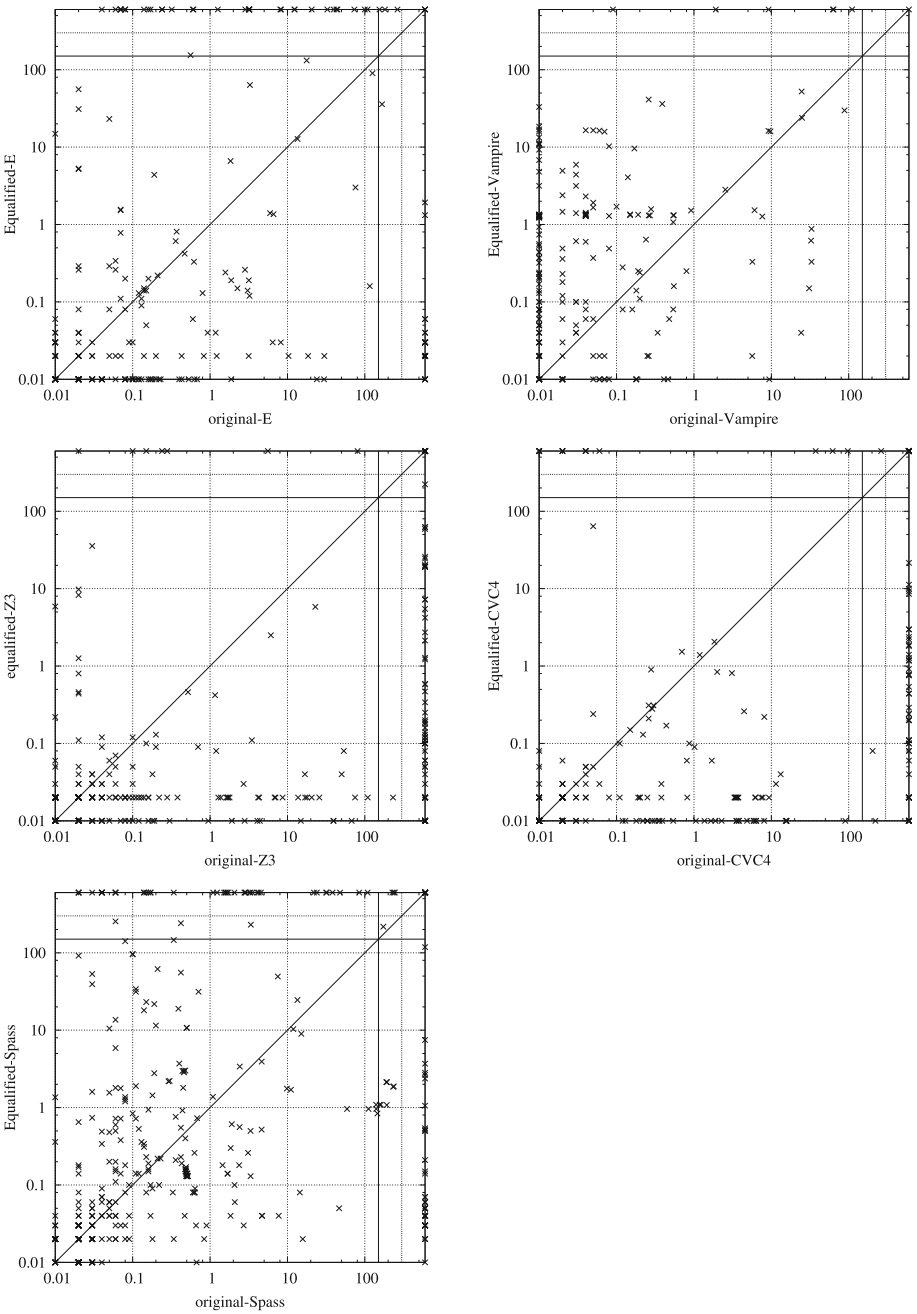

Fig. 20 Time (in s) taken to solve problems, with and without equalification, using E, Vampire, Z3 and CVC4. Every mark (×) indicates a problem run before and after the transformation. Times in seconds

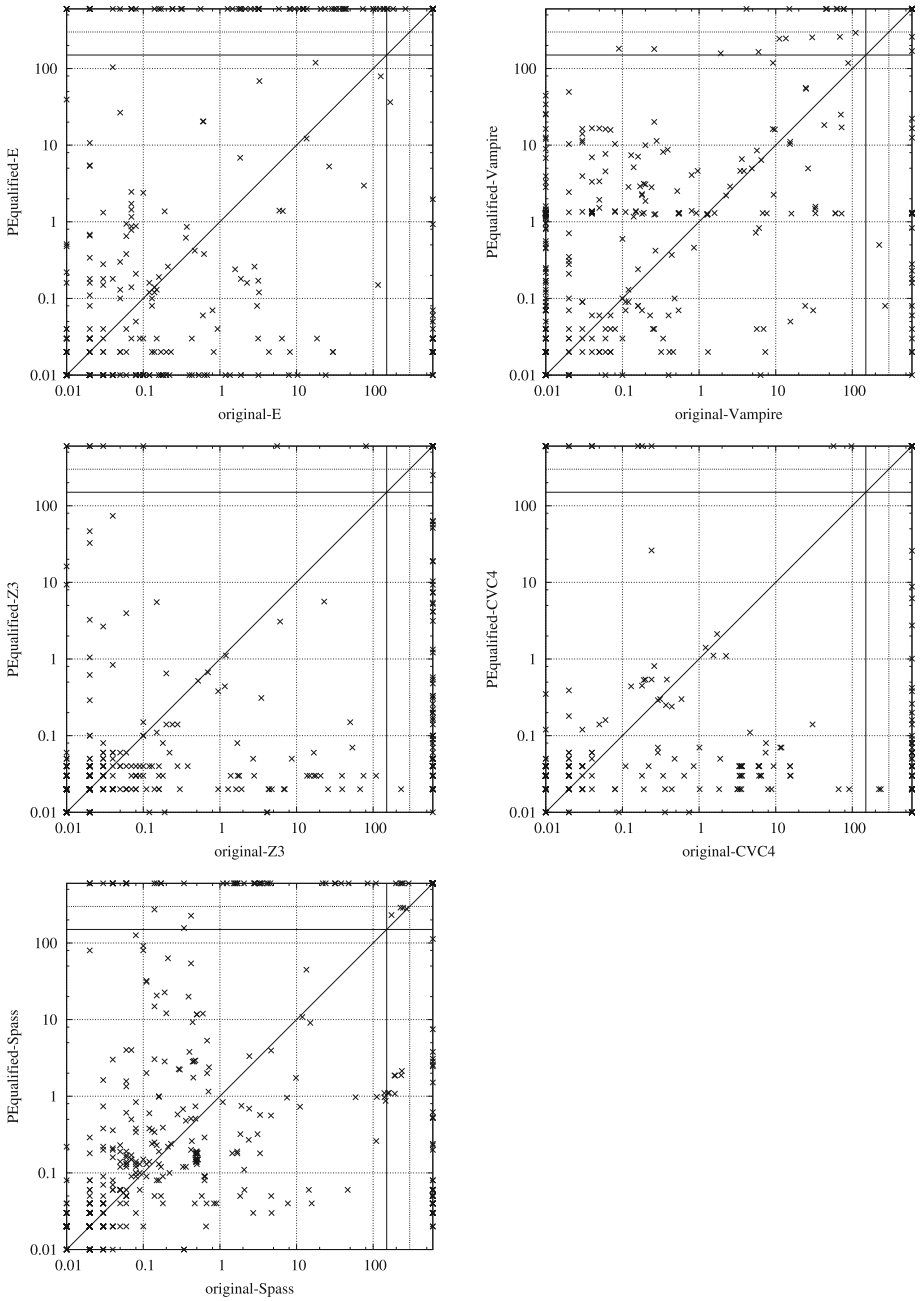

Fig. 21 Time (in s) taken to solve problems, with and without pequalification, using E, Vampire, Spass, Z3 and CVC4. Every mark (×) indicates a problem run before and after the transformation. Times in seconds

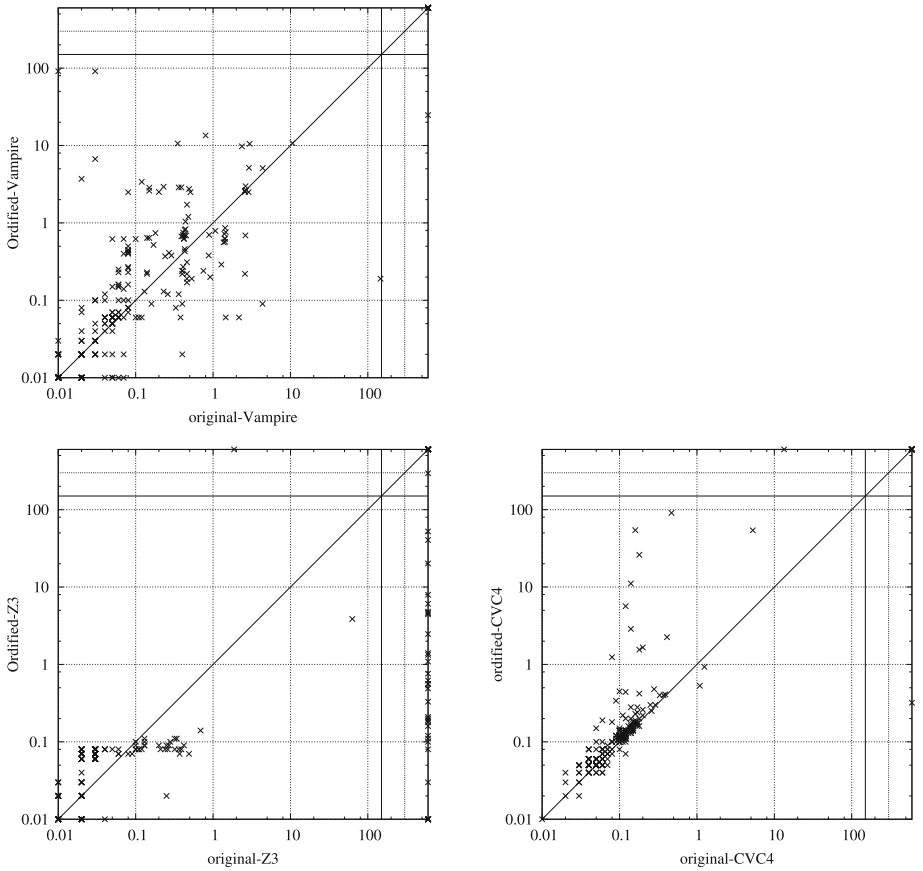

Fig. 22 Time (in s) taken to solve problems, with and without ordification, using Vampire, Z3 and CVC4. Every mark (×) indicates a problem run before and after the transformation. Times in seconds

7.8.1 E

E did not have any major benefits from any of the transformations. However, with time-slicing we can avoid a lot of the bad effects of a transformation, while still keeping the results that were improved. For E, the best results were achieved when allowing 10 s on each transformed problem and the remaining time on the original problem. This is based on a time-limit of 5 min. Fig. 13 show the subsets on which there was a transformation that solved new problems. With this strategy, we gain a total of 10 solved problems, 7 with equivalence relations and 3 with partial orders. On the remaining subsets there is no transformation that increases the success rate compared to the original, but splitting does not decrease it.

7.8.2 Vampire

For Vampire, detransification is the best choice for both partial orders and partial equivalence relations. A majority of the problems solved after detransification but not before took a long time, making time-slicing less favorable. For the other subsets, Vampire is the most successful

on the original problems. Using detransification on partial equivalences and partial orders, and no transformation otherwise, we gain 39 solved problems and lose 6.

7.8.3 Spass

For Spass, the best results for transitive and reflexive relations (i.e. equivalences and partial orders) were given by spending 10 s on each of the applicable transformations, and the remaining time on the original problem. For the remaining subsets, splitting evenly between the applicable transformations and untransformed problems solves the most problems. The 8 new solved problems with a partial equivalence also have a strict partial order, and thus are contained in the 43 new problems that were solved in the partial orders subset. In total, the optimal strategy solves 75 new problems, while it loses 15.

7.8.4 Z3

For Z3, the best strategy is to split the time evenly between the original problem and all of the applicable transformations. Spending 10 s on each applicable transformation and the remaining time on the original problem gives very similar results. In total, we solve 135 new problems compared to the original, and lose 6. 21 of the newly solved problems are in overlapping subsets (containing more than one kind of transitive relation).

7.8.5 CVC4

For CVC4, splitting evenly between the original problem and all of the applicable transformations gives the best results. 19 of the newly solved problems are overlapping; 16 problems have both a partial equivalence and a strict partial order. 3 of the problems have an equivalence relation and a partial order. The total gain of the optimal strategy is 83 problems, and the loss is 2 problems.

8 Discussion and Conclusions

We have presented 6 transformations that can be applied to theories with certain transitive relations: equalification, pequalification, ordification, maxification, detransification, and detransification with reflexivity. 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 or more 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, there exist 5 TPTP problems that are now solvable that weren't previously. These are FLD012-2 (solved by E with detransification), SEU322+2 (Solved by Vampire and Z3 with detransification), SEU270+2 (solved by CVC4 and Z3 with detransification), SEU372+2 (solved by Vampire with detransification) and LDA005-1 (solved by Z3 with ordification).

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 be integrated in a natural way in provers that already have the time slicing machinery in place.

9 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.

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.

Acknowledgements We thank Nicholas Smallbone and the anonyous referees for discussions and useful suggestions on earlier versions of this paper.

References

1. Leo Bachmair and Harald Ganzinger. Ordered chaining calculi for first-order theories of transitive relations. Journal of the ACM, 45(6), 1007–1049, 1998
2. Barrett, C., Conway, C.L., Deters, M., Hadarean, L., Jovanović, D., King, T., Reynolds, A., Tinelli, C.: CVC4. In: Proceedings of the 23rd International Conference on Computer Aided Verification (CAV'11). Springer, Berlin (2011)
3. Cristiá, M., Rossi, G.: A decision procedure for sets, binary relations and partial functions. In: Proceedings of the 28th Conference on Computer Aided Verification (CAV'16). Springer, Berlin (2016)
4. de Moura, L., Bjørner, N.: Z3: An efficient SMT solver. In: Proceedings of the 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'08). Springer (2008)
5. Hillenbrand, T., Piskac, R., Waldmann, U., Weidenbach, C.: From search to computation: Redundancy criteria and simplification at work. In: Programming Logics: Essays in Memory of Harald Ganzinger, pp. 169–193. Springer, Berlin (2013)
6. Horrocks, I., Gough, G.: Description logics with transitive roles. Descr. Logics **9**(3), 385–410 (1997)
7. Kovács, L., Voronkov, A.: First-order theorem proving and vampire. In: Proceedings of the 25th Conference on Computer Aided Verification (CAV'13). Springer, Berlin (2013)
8. Meng, B., Reynolds, A., Tinelli, C., Barrett, C.: Relational constraint solving in SMT. In: de Moura, L. (ed.) Proceedings of the 26nd Conference on Automated Deduction (CADE-26). Springer, Berlin (2017)

9. Schmidt, R.A., Hustadt, U.: The axiomatic translation principle for modal logic. ACM Trans. Comput. Logic **8**(4), 19 (2007)
10. Schulz, S.: The E theorem prover (2015). http://www.eprover.org/. Accessed 3 June 2021
11. Sutcliffe, G.: The TPTP problem library (2015). http://www.tptp.org/. Accessed 3 June 2021
12. Torlak, E., Jackson, D.: Kodkod: a relational model finder. In: Proceedings of the 13th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'07). Springer, Berlin (2007)
13. Weidenbach, C., Dimova, D., Fietzke, A., Kumar, R., Suda, M., Wischnewski, P.: Spass version 3.5. In: Proceedings of the 22nd Conference on Automated Deduction (CADE-22). Springer, Berlin (2009)