Efficient encodings of first-order Horn formulas in equational logic

Koen Claessen and Nicholas Smallbone

Chalmers University of Technology, Gothenburg, Sweden
{koen,nicsma}@chalmers.se

Abstract. We present several translations from first-order Horn formulas to equational logic. The goal of these translations is to allow equational theorem provers to efficiently reason about non-equational problems. Using these translations we were able to solve 37 problems of rating 1.0 (i.e. which had not previously been automatically solved) from the TPTP.

1 Introduction

Equational theorem provers such as Waldmeister [14] and Twee [20] are highly effective on equational problems, and often outperform first-order theorem provers. But they are also quite limited: while many problems require heavy use of equational reasoning, few problems consist purely of unit equations.

Take, for example, problem `LAT224-1` from the TPTP [21]. In many ways this problem is perfect for equational theorem provers. It is about lattice theory, and includes all the usual lattice axioms such as associativity, commutativity, idempotence and absorption. It also has the rather juicy-looking axiom

$$x \sqcap (y \sqcup (x \sqcap z)) = (x \sqcap y) \sqcup (x \sqcap (y \sqcup (z \sqcap (x \sqcup (y \sqcap z)))))$$

which any equational prover would love to reason about. Unfortunately, it has exactly one non-unit axiom,

$$x \sqcap y = \bot \wedge x \sqcup y = \top \to \bar{x} = y, \tag{1}$$

so we can not use an equational prover. No theorem prover is able to automatically prove `LAT224-1`: it has always had rating 1.0 on the TPTP.

It is possible to prove `LAT224-1` if we *encode* the non-unit axiom as a unit equation. Suppose we add a new function ifeq together with the axiom

$$\mathsf{ifeq}(x, x, y, z) = y.$$

The idea is that $\mathsf{ifeq}(x, y, z, w)$ represents the expression "if $x = y$ then z else w". We can then reformulate axiom (1) as the equation

$$\mathsf{ifeq}(x \sqcap y, \bot, \mathsf{ifeq}(x \sqcup y, \top, \bar{x}, y), y) = y$$

and now we have a unit equality problem. When we present this transformed problem to the equational prover Twee, it solves it in 5 seconds.

This encoding works for any Horn formula, it is easy to automate, and it does not alter unit clauses. Thus we can use it to cheaply add Horn clause reasoning to an equational prover without weakening its equational reasoning powers.

The idea of using ifeq to encode clauses as equations is not new: it originated as a way of encoding *full* first-order logic as equations [7, 16]. The fact that all first-order formulas can be encoded as equations is remarkable, but the cited encoding needs many axioms for ifeq, as well as congruence axioms for each function in the input problem, so it is not a practical way of proving theorems.

On the other hand, as `LAT224-1` shows, if we take advantage of the simple nature of Horn formulas, we can come up with encodings that are simple and practical for theorem proving. These encodings let an equational theorem prover reason *efficiently* about Horn formulas, and turn it into a powerful theorem prover for mostly-equational problems.

This paper introduces several such encodings, and shows that they work in theory and in practice.

Contributions We describe and prove correct four efficient encodings of Horn formulas into equational logic. The first two encodings are inspired by existing (but impractical) encodings for full first-order logic [7, 16], and the last two are our own invention. We evaluate our encodings on the TPTP and are able to solve 37 problems of rating 1.0, in other words problems that no existing prover could automatically solve.

Notation and definitions A Horn clause is a clause with at most one positive literal, for example $\neg A \vee \neg B \vee C$ or $\neg A$, where A, B and C are atomic formulas; a Horn formula is a set of Horn clauses. As in first-order logic, an atomic formula is either a predicate or an equation. A Horn clause with a positive literal is a *definite clause*; a Horn clause with no positive literal is a *goal clause*. We freely write Horn clauses as implications; for example, instead of $\neg A \vee \neg B \vee C$, we often write $A \wedge B \to C$, and we also write goal clauses as $A \wedge B \to \mathbf{false}$. When given a Horn formula, as is usual in theorem proving, the problem is to prove the conjunction of the clauses unsatisfiable.

When writing formulas, we adopt the convention that x, y, z and w are variables and s, t, u and v stand for terms.

2 Encoding equational Horn clauses as equations

In this section we present four encodings from *equational* Horn formulas to unit equations. The encodings take as input a Horn formula with no predicate symbols (other than equality), and produce a set of unit equalities plus a set of goal clauses. In fact, the goal clauses from the input formula are passed through unchanged. In Section 3 we discuss how to handle predicate symbols, and in Section 4 we discuss different ways to handle the goal clauses.

Each of the encodings in this section consists of:

- A set of axioms which are unconditionally added to the input formula.
- A rule which eliminates *one* negative literal from a clause, by replacing a clause of the form $C \wedge s = t \to u = v$ with a clause of the form $C \to u' = v'$. The rule can also add new unit clauses to the problem.

To apply the encoding, we must add the axioms specified by the encoding, and then repeatedly apply the prescribed rule until no negative literals remain, except those that are in goal clauses.

We demonstrate all the encodings on the following example clauses:

$$f(x) = f(y) \to x = y$$
$$f(a) = b \wedge f(c) = d \to a = c$$

2.1 Encoding 1: if-then-else

We start with the encoding described in the introduction. To recap, the idea is to have a function $\mathsf{ifeq}(x, y, z, w)$ which is supposed to mean "if $x = y$ then z else w", and to encode Horn clauses using if-then-else.

First we add to the input formula the axiom

$$\mathsf{ifeq}(x, x, y, z) = y$$

where ifeq must of course be a fresh symbol.

The rule we use to eliminate a negative literal is: given a clause of the form $C \wedge s = t \to u = v$, replace it with the clause

$$C \to \mathsf{ifeq}(s, t, u, v) = v.$$

Since the term v is duplicated, if v has a greater size than u, we swap u and v before applying the encoding rule, in order to reduce formula size.

Example The example formula above is encoded as the following three equations:

$$\mathsf{ifeq}(x, x, y, z) = y$$
$$\mathsf{ifeq}(f(x), f(y), x, y) = y$$
$$\mathsf{ifeq}(f(a), b, \mathsf{ifeq}(f(c), d, a, c), c) = c$$

The third clause above is derived as follows:

$$\begin{aligned} & f(a) = b \wedge f(c) = d \to a = c \\ \equiv\ & f(a) = b \to \mathsf{ifeq}(f(c), d, a, c) = c \\ \equiv\ & \mathsf{ifeq}(f(a), b, \mathsf{ifeq}(f(c), d, a, c), c) = c \end{aligned}$$

Efficiency An efficient encoding should have several characteristics:

1. It should not alter unit equations, so that the prover can deal efficiently with the equational part of the problem.
2. It should increase the size of the problem as little as possible.
3. Discharging a condition should not require lots of book-keeping inferences. In other words, from the encoded versions of $s = t$ and $s = t \to u = v$ it should be easy to deduce $u = v$.

4. It should not increase the search space by allowing needless or unproductive inferences. If possible, any valid inference in the encoded problem should correspond to a reasonable inference in the original Horn problem.

The if-then-else encoding does well on most of those fronts:

1. It does not alter unit equations.
2. The term v is duplicated during encoding, so the problem may blow up. In practice, the fact that we pick v to be smaller than u helps.
3. Discharging a condition takes at most two inferences: from $s = t$ we can deduce $\mathsf{ifeq}(s, s, u, v) = v$ and from that and $\mathsf{ifeq}(x, x, y, z) = y$ we get $u = v$.
4. Assuming that the prover uses a simplification ordering, in the equation $\mathsf{ifeq}(s, t, u, v)$, the only inferences allowed will be paramodulations into s, t, u and v. The first two are useful but the last two are not; we fix this in encoding 3.

Proof of correctness Suppose that we are encoding the formula ϕ. We start by adding the function ifeq and the axiom $\mathsf{ifeq}(x, x, y, z) = y$ to obtain the formula ϕ_0. We then eliminate negative literals one at a time to obtain a sequence of formulas $\phi_1, \ldots, \phi_n$. That is, we obtain ϕ_{i+1} from ϕ_i by replacing one clause $C \wedge s = t \to u = v$ with $C \to \mathsf{ifeq}(s, t, u, v) = v$. Our goal is to show that ϕ and ϕ_n are equisatisfiable. A note on notation: in this paper, given a model $\mathcal{M}$ and a variable assignment σ, we write $\mathcal{M}, \sigma \models \phi$ for the valuation of a formula, and $\mathcal{M}^\sigma(t)$ for the valuation of a term.

Soundness Given a model $\mathcal{M}$ of ϕ, we extend it to a model $\mathcal{M}_0$ of ϕ_0 by interpreting ifeq as follows:

$$\mathsf{ifeq}(x, y, z, w) = \begin{cases} z, \text{ if } x = y \\ w, \text{ otherwise} \end{cases}$$

This definition clearly satisfies the axiom $\mathsf{ifeq}(x, x, y, z) = z$, so we have $\mathcal{M}_0 \models \phi_0$. From the following lemma, it follows immediately by induction that $\mathcal{M}_0 \models \phi_i$ for all i, so in particular $\mathcal{M}_0 \models \phi_n$.

Lemma 1 (Single step soundness). *If $\mathcal{M}_0 \models s = t \to u = v$ then $\mathcal{M}_0 \models \mathit{ifeq}(s, t, u, v) = v$.*

Proof. Let σ be a variable assignment such that $\mathcal{M}_0, \sigma \models s = t \to u = v$. We show that $\mathcal{M}_0, \sigma \models \mathsf{ifeq}(s, t, u, v) = v$ by case analysis on the values of s, t, u and v in M_0^σ:

- If $s \neq t$,[1] then $\mathsf{ifeq}(s, t, u, v) = v$ by definition of ifeq in $\mathcal{M}_0$.
- If $s = t$ and $u = v$, then $\mathsf{ifeq}(s, t, u, v) = \mathsf{ifeq}(s, s, v, v) = v$. □

[1] Really we mean $\mathcal{M}_0^\sigma(s) \neq \mathcal{M}_0^\sigma(t)$, but we leave out the heavy notation in this proof.

Completeness Since ϕ_0 is stronger than ϕ, any model of ϕ_0 is a model of ϕ. It remains to show that if $\mathcal{M} \models \phi_n$ then $\mathcal{M} \models \phi_0$. This follows immediately by induction from the following lemma and the fact that $\mathsf{ifeq}(x, x, y, z) = y$ is an axiom of ϕ_n:

Lemma 2 (Single step completeness). *Suppose that* $\mathcal{M} \models \mathit{ifeq}(x, x, y, z) = y$. *If* $\mathcal{M} \models \mathit{ifeq}(s, t, u, v) = v$ *then* $\mathcal{M} \models s = t \to u = v$.

Proof. Given a variable assignment σ, and assuming that $\mathcal{M}, \sigma \models s = t$, we prove that $\mathcal{M}, \sigma \models u = v$. Again we drop the heavy "$\mathcal{M}, \sigma \models$" notation. From $s = t$ and $\mathsf{ifeq}(x, x, y, z) = z$ we get $\mathsf{ifeq}(s, t, u, v) = u$. Combined with the assumption $\mathsf{ifeq}(s, t, u, v) = v$ this gives $u = v$. □

2.2 Encoding 2: if-then

The if-then-else encoding is asymmetric: when encoding the clause $s = t \to u = v$, the term v is duplicated but u is not. The if-then encoding is a symmetric variant. The encoding uses a function $\mathsf{ifeq}(x, y, z)$ which is intended to mean "if $x = y$ then z else unspecified". We add the following axiom to the input formula:

$$\mathsf{ifeq}(x, x, y) = y.$$

The rule we use to eliminate a negative literal is: given a clause of the form $C \wedge s = t \to u = v$, replace it with

$$C \to \mathsf{ifeq}(s, t, u) = \mathsf{ifeq}(s, t, v).$$

Example The example clause set becomes:

$$\begin{aligned} \mathsf{ifeq}(x, x, y) &= y \\ \mathsf{ifeq}(f(x), f(y), x) &= \mathsf{ifeq}(f(x), f(y), y) \\ \mathsf{ifeq}(f(a), b, \mathsf{ifeq}(f(c), d, a)) &= \mathsf{ifeq}(f(a), b, \mathsf{ifeq}(f(c), d, c)) \end{aligned}$$

Efficiency Compared to the if-then-else encoding, the if-then encoding is likely to produce bigger equations, as the equation $\mathsf{ifeq}(s, t, u) = \mathsf{ifeq}(s, t, v)$ duplicates both s and t. It also requires more inference steps to discharge a condition, as both sides of the equation must be rewritten. However, if u and v are large terms, it may produce smaller equations than the if-then-else encoding.

Proof of correctness Almost identical to the if-then-else encoding. The only change is that ϕ_0 is now ϕ together with the axiom $\mathsf{ifeq}(x, x, y) = y$, and we construct its model differently. Given a model $\mathcal{M}$ of ϕ, we extend it to a model $\mathcal{M}_0$ of ϕ_0 by interpreting ifeq as follows, where a is an arbitrary domain element of $\mathcal{M}$:

$$\mathsf{ifeq}(x, y, z) = \begin{cases} z, \text{ if } x = y \\ a, \text{ otherwise} \end{cases}$$

2.3 Encoding 3: specialised if-then-else

The third encoding is designed to work well with Knuth-Bendix completion. The aim is to encode $s = t \to u = v$ in such a way that the resulting equations become rewrite rules in which u and v only appear on the *right-hand* side. This means that only s and t participate in critical pairs, not u and v.

The rule we use is: given a clause of the form $C \wedge s = t \to u = v$, replace it with the two clauses

$$\mathsf{fresh}_i(y, y, x_1, \ldots, x_n) = u$$
$$C \to \mathsf{fresh}_i(s, t, x_1, \ldots, x_n) = v$$

where fresh_i is a fresh function symbol and $x_1, \ldots, x_n$ is the union of the free variables of u and v.[2] We introduce a new symbol fresh_i each time the rule is applied. The idea is that $\mathsf{fresh}_i(x, y, x_1, \ldots, x_n)$ represents the expression "if $x = y$ then u else v".[3] Once the prover derives $s = t$, the two equations can be combined to yield $u = v$. Thus, fresh_i is really the function symbol ifeq from before (which is not used in this encoding), *specialised* to the implication at hand, which removes the need to have u and v as an argument to ifeq.

In our testing, the encoding works best if we always let u be the smaller term and v the bigger term of the positive literal, ordering by weight—that is, the opposite way to the if-then-else encoding. We are not sure why.

Efficiency If the two equations above are oriented left-to-right, the only inference a prover can make is to paramodulate into s and t in order to make them equal. Once s and t are made equal, the two rules can be combined to derive $u = v$. Thus the search space is reduced: the theorem prover effectively simulates a first-order prover working forward from positive unit literals.

Example Take the example clause set. The first clause is encoded as

$$\mathsf{fresh}_1(z, z, x, y) = x$$
$$\mathsf{fresh}_1(f(x), f(y), x, y) = y.$$

The second clause is first transformed into

$$\mathsf{fresh}_2(x, x) = a$$
$$f(a) = b \to \mathsf{fresh}_2(f(c), d) = c$$

and this latter clause becomes

$$\mathsf{fresh}_3(x, x) = \mathsf{fresh}_2(f(c), d)$$
$$\mathsf{fresh}_3(f(a), b) = c.$$

[2] Since we are working with clauses, free variables are universally quantified.

[3] The arguments $x_1, \ldots, x_n$ help to unambiguously identify u and v. Without them, this interpretation of fresh_i would not make sense and the encoding would be unsound.

The final result is five equations:

$$\begin{aligned}
\mathsf{fresh}_1(z, z, x, y) &= x \\
\mathsf{fresh}_1(f(x), f(y), x, y) &= y \\
\mathsf{fresh}_2(x, x) &= a \\
\mathsf{fresh}_3(x, x) &= \mathsf{fresh}_2(f(c), d) \\
\mathsf{fresh}_3(f(a), b) &= c
\end{aligned}$$

We argued above that we would like each of these equations to be oriented left-to-right, but for the fourth equation it is not clear if that will happen. This suggests that the encoding could be improved if we could give the prover an appropriate ordering for the fresh symbols. See Section 2.5 for another solution.

Proof of correctness We first introduce some notation. We write $\overrightarrow{x}$ for $(x_1, \ldots, x_n)$, the sequence of all free variables of u and v. If σ is a variable assignment, we write $\sigma(\overrightarrow{x})$ for $(\sigma(x_1), \ldots, \sigma(x_n))$. If $\mathcal{M}$ is an interpretation and $(c_1, \ldots, c_n)$ are domain elements then we write $\mathcal{M}^{\overrightarrow{c}}(u)$ or $\mathcal{M}^{\overrightarrow{c}}(v)$ for the value of u or v under the variable assignment $\{x_1 \mapsto c_1, \ldots, x_n \mapsto c_n\}$. Note that if $\sigma(\overrightarrow{x}) = \overrightarrow{c}$ then $\mathcal{M}^{\sigma}(u) = \mathcal{M}^{\overrightarrow{c}}(u)$ and $\mathcal{M}^{\sigma}(v) = \mathcal{M}^{\overrightarrow{c}}(v)$.

It is enough to show that a single application of the encoding rule is sound and complete. In other words, if ϕ is a formula which contains the clause

$$C \wedge s = t \rightarrow u = v,$$

and ϕ_{enc} is ϕ with that clause replaced by the following two clauses:

$$\begin{aligned}
\mathsf{fresh}(y, y, \overrightarrow{x}) &= u \\
C \rightarrow \mathsf{fresh}(s, t, \overrightarrow{x}) &= v,
\end{aligned}$$

then we must show that ϕ and ϕ_{enc} are equisatisfiable.

Soundness Suppose $\mathcal{M}$ is a model of ϕ. We extend $\mathcal{M}$ to a model $\mathcal{M}_{\mathsf{enc}}$ of ϕ_{enc} by interpreting fresh as follows:

$$\mathsf{fresh}(a, b, \overrightarrow{c}) = \begin{cases} \mathcal{M}^{\overrightarrow{c}}(u), \text{ if } a = b \\ \mathcal{M}^{\overrightarrow{c}}(v), \text{ otherwise} \end{cases}$$

Note that $\mathcal{M}$ and $\mathcal{M}_{\mathsf{enc}}$ agree on the truth of any formula not involving fresh, and that since fresh was freshly generated, it does not occur in s, t, u, v or C.

We need to check that both of the new clauses of ϕ_{enc} hold. First we show that $\mathcal{M}_{\mathsf{enc}} \models \mathsf{fresh}(y, y, \overrightarrow{x}) = u$: given a variable assignment σ, let $\overrightarrow{c} = \sigma(\overrightarrow{x})$; then $\mathcal{M}^{\sigma}_{\mathsf{enc}}(\mathsf{fresh}(y, y, \overrightarrow{x})) = \mathsf{fresh}(\sigma(y), \sigma(y), \overrightarrow{c}) = \mathcal{M}^{\overrightarrow{c}}_{\mathsf{enc}}(u) = \mathcal{M}^{\sigma}_{\mathsf{enc}}(u)$.

Then we assume that σ is a variable assignment such that $\mathcal{M}_{\mathsf{enc}}, \sigma \models C$, and show that $\mathcal{M}_{\mathsf{enc}}, \sigma \models \mathsf{fresh}(s, t, \overrightarrow{x}) = v$. Let $\overrightarrow{c} = \sigma(\overrightarrow{x})$. Note that since fresh does not occur in C, we must have $\mathcal{M}, \sigma \models C$. Recalling that $\mathcal{M}, \sigma \models C \wedge s = t \rightarrow u = v$, the proof proceeds by case analysis:

– If $\mathcal{M}, \sigma \not\models s = t$, then $\mathcal{M}^{\sigma}_{\mathsf{enc}}(\mathsf{fresh}(s, t, \overrightarrow{x})) = \mathsf{fresh}(\mathcal{M}^{\sigma}(s), \mathcal{M}^{\sigma}(t), \overrightarrow{c}) = \mathcal{M}^{\overrightarrow{c}}(v) = \mathcal{M}^{\sigma}_{\mathsf{enc}}(v)$, since $\mathcal{M}^{\sigma}(s) \neq \mathcal{M}^{\sigma}(t)$.
– If $\mathcal{M}, \sigma \models s = t$ and $\mathcal{M}, \sigma \models u = v$, then $\mathcal{M}^{\sigma}_{\mathsf{enc}}(\mathsf{fresh}(s, t, \overrightarrow{x})) = \mathsf{fresh}(\mathcal{M}^{\sigma}(s), \mathcal{M}^{\sigma}(t), \overrightarrow{c}) = \mathcal{M}^{\overrightarrow{c}}(u) = \mathcal{M}^{\sigma}(u) = \mathcal{M}^{\sigma}_{\mathsf{enc}}(v)$.

Completeness To go from a model of ϕ_{enc} to a model of ϕ, we show that if (i) $\mathcal{M} \models \mathsf{fresh}(y, y, \overrightarrow{x}) = u$ and (ii) $\mathcal{M}, \sigma \models C \to \mathsf{fresh}(s, t, \overrightarrow{x}) = v$, then $\mathcal{M}, \sigma \models C \wedge s = t \to u = v$.

Assume that $\mathcal{M}, \sigma \models C \wedge s = t$. From (i) and $\mathcal{M}^{\sigma}(s) = \mathcal{M}^{\sigma}(t)$ we get $\mathcal{M}^{\sigma}(\mathsf{fresh}(s, t, \overrightarrow{x})) = \mathcal{M}^{\sigma}(u)$. From (ii) and $\mathcal{M}, \sigma \models C$ we get $\mathcal{M}^{\sigma}(\mathsf{fresh}(s, t, \overrightarrow{x})) = \mathcal{M}^{\sigma}(v)$. Therefore $\mathcal{M}^{\sigma}(u) = \mathcal{M}^{\sigma}(\mathsf{fresh}(s, t, \overrightarrow{x})) = \mathcal{M}^{\sigma}(v)$ and $\mathcal{M}, \sigma \models u = v$.

2.4 Encoding 4: split if

When using Knuth-Bendix completion, the equation $\mathsf{fresh}(s, t, \overrightarrow{x}) = u$ has the disadvantage that the number of critical pairs it creates is the *product* of the number of critical pairs created using s and using t. The fourth encoding solves this problem by having two equations, one for s and one for t.

The rule we use is: replace a clause of the form $C \wedge s = t \to u = v$ with the two clauses

$$\mathsf{fresh}_i(s, x_1, \ldots, x_n) = u$$
$$C \to \mathsf{fresh}_i(t, x_1, \ldots, x_n) = v,$$

where $x_1, \ldots, x_n$ consists of the union of the free variables of s, t, u and v (not just u and v as in encoding 3) and fresh_i is a fresh function symbol. Just as in encoding 3, we introduce a new symbol fresh_i each time the rule is applied.

Example For the example clause set, the first clause is encoded as:

$$\mathsf{fresh}_1(f(x), x, y) = x$$
$$\mathsf{fresh}_1(f(y), x, y) = y$$

The second clause becomes:

$$\mathsf{fresh}_2(f(c)) = a$$
$$f(a) = b \to \mathsf{fresh}_2(d) = c$$

and the latter clause in turn is encoded as:

$$\mathsf{fresh}_3(f(a)) = \mathsf{fresh}_2(d)$$
$$\mathsf{fresh}_3(b) = c$$

Efficiency The chief gain compared to encoding 3 is the reduced number of critical pairs. One disadvantage is that we must include the free variables of all four terms, s, t, u and v, which may lead to rather large equations.

Proof of correctness We use the same setup and notation, and indeed most of the same proof, as for encoding 3. Suppose that ϕ is a formula which contains the clause

$$C \wedge s = t \to u = v,$$

and that in ϕ_{enc} that clause has been replaced by the following two:

$$\mathsf{fresh}(s, \overrightarrow{x}) = u$$
$$C \to \mathsf{fresh}(t, \overrightarrow{x}) = v.$$

We show that ϕ and ϕ_{enc} are equisatisfiable.

Soundness Suppose that $\mathcal{M}$ is a model of ϕ. We extend $\mathcal{M}$ to a model $\mathcal{M}_{\mathsf{enc}}$ of ϕ_{enc} by interpreting fresh as follows:

$$\mathsf{fresh}(a, \overrightarrow{b}) = \begin{cases} \mathcal{M}^{\overrightarrow{b}}(u), \text{ if } \mathcal{M}^{\overrightarrow{b}}(s) = a \\ \mathcal{M}^{\overrightarrow{b}}(v), \text{ otherwise} \end{cases}$$

As before, $\mathcal{M}$ and $\mathcal{M}_{\mathsf{enc}}$ agree on the truth of any formula not involving fresh, and fresh does not occur in s, t, u, v or C.

First we check that $\mathcal{M}_{\mathsf{enc}} \models \mathsf{fresh}(s, \overrightarrow{x}) = u$. Given a variable assignment σ, let $\overrightarrow{b} = \sigma(\overrightarrow{x})$. Then $\mathcal{M}^{\sigma}_{\mathsf{enc}}(\mathsf{fresh}(s, \overrightarrow{x})) = \mathsf{fresh}(\mathcal{M}^{\sigma}(s), \overrightarrow{b}) = \mathsf{fresh}(\mathcal{M}^{\overrightarrow{b}}(s), \overrightarrow{b}) = \mathcal{M}^{\overrightarrow{b}}(u) = \mathcal{M}^{\sigma}_{\mathsf{enc}}(u)$.

Then we show that if $\mathcal{M}_{\mathsf{enc}} \models C$ then $\mathcal{M}_{\mathsf{enc}} \models\!\to \mathsf{fresh}(t, \overrightarrow{x}) = v$. Take a variable assignment σ such that $\mathcal{M}_{\mathsf{enc}} \models C$, which implies that $\mathcal{M} \models C$, and let $\overrightarrow{b} = \sigma(\overrightarrow{x})$. Since $\mathcal{M}, \sigma \models C \wedge s = t \to u = v$, we can do a case split:

- If $\mathcal{M}, \sigma \not\models s = t$, then $\mathcal{M}^{\sigma}_{\mathsf{enc}}(\mathsf{fresh}(t, \overrightarrow{x})) = \mathsf{fresh}(\mathcal{M}^{\sigma}(t), \sigma(\overrightarrow{x})) = \mathsf{fresh}(\mathcal{M}^{\overrightarrow{b}}(t), \overrightarrow{b}) = \mathcal{M}^{\overrightarrow{b}}(v) = \mathcal{M}^{\sigma}_{\mathsf{enc}}(v)$, since $\mathcal{M}^{\sigma}(t) \neq \mathcal{M}^{\sigma}(s)$.
- If $\mathcal{M}, \sigma \models s = t$ and $\mathcal{M}, \sigma \models u = v$, then $\mathcal{M}^{\sigma}_{\mathsf{enc}}(\mathsf{fresh}(t, \overrightarrow{x})) = \mathsf{fresh}(\mathcal{M}^{\sigma}(t), \sigma(\overrightarrow{x})) = \mathsf{fresh}(\mathcal{M}^{\sigma}(s), \sigma(\overrightarrow{x})) = \mathsf{fresh}(\mathcal{M}^{\overrightarrow{b}}, \overrightarrow{b}) = \mathcal{M}^{\overrightarrow{b}}(v) = \mathcal{M}^{\sigma}_{\mathsf{enc}}(v)$.

Completeness To go from a model of ϕ_{enc} to a model of ϕ, we show that if (i) $\mathcal{M} \models \mathsf{fresh}(s, \overrightarrow{x}) = u$ and (ii) $\mathcal{M}, \sigma \models C \to \mathsf{fresh}(t, \overrightarrow{x}) = v$, then $\mathcal{M}, \sigma \models C \wedge s = t \to u = v$.

Assume that $\mathcal{M}, \sigma \models C \wedge s = t$. From (i) and $\mathcal{M}^{\sigma}(s) = \mathcal{M}^{\sigma}(t)$ we get $\mathcal{M}^{\sigma}(\mathsf{fresh}(t, \overrightarrow{x})) = \mathcal{M}^{\sigma}(u)$. From (ii) and $\mathcal{M}, \sigma \models C$ we get $\mathcal{M}^{\sigma}(\mathsf{fresh}(t, \overrightarrow{x})) = \mathcal{M}^{\sigma}(v)$. Therefore $\mathcal{M}^{\sigma}(u) = \mathcal{M}^{\sigma}(\mathsf{fresh}(t, \overrightarrow{x})) = \mathcal{M}^{\sigma}(v)$ and $\mathcal{M}, \sigma \models u = v$.

2.5 Tupling, an optional transformation

In encodings 3 and 4, nested implications become rather messy. For example, in encoding 4, the implication

$$a = b \wedge c = d \to e = f$$

turns into the following equations:

$$\begin{aligned}\mathsf{fresh}_1(c) &= e\\ \mathsf{fresh}_2(a) &= \mathsf{fresh}_1(d)\\ \mathsf{fresh}_2(b) &= f\end{aligned}$$

In Section 2.3, we argued that for efficiency the encoded equations should always be oriented with fresh_i on the left, but this is not possible for the middle equation, and efficiency may suffer.

If we introduce a function symbol tuple, we can transform the above implication into the following binary clause:

$$\mathsf{tuple}(a, c) = \mathsf{tuple}(b, d) \to e = f$$

This exploits the fact that the only way to prove $\mathsf{tuple}(a, c) = \mathsf{tuple}(b, d)$ is to prove $a = b$ and $c = d$, because there are no extra axioms about tuple[4].

Applying encoding 4 now gives a much cleaner translation, and the equations will be oriented correctly:

$$\begin{aligned}\mathsf{fresh}(\mathsf{tuple}(a, c)) &= e\\ \mathsf{fresh}(\mathsf{tuple}(b, d)) &= f\end{aligned}$$

In fact, we can fuse the function symbol fresh with tuple, because fresh is never used without tuple, and the result is:

$$\begin{aligned}\mathsf{fresh}(a, c) &= e\\ \mathsf{fresh}(b, d) &= f\end{aligned}$$

In general, tupling transforms the clause $t_1 = u_1 \wedge \ldots \wedge t_n = u_n \to t = u$ into the clause $\mathsf{tuple}_n(t_1, \ldots, t_n) = \mathsf{tuple}_n(u_1, \ldots, u_n) \to t = u$, where tuple_n is a function symbol of arity n.

If the input problem is sorted, the result sort of tuple should be a fresh sort, otherwise we risk unsoundness. If the input problem is unsorted, then it is safe for tuple to be unsorted too. To show this, we require two lemmas.

Lemma 3 (Safe sort erasure). *Let ϕ be a sorted formula and ϕ_{erased} be the same formula with all sorts erased. Suppose that ϕ has the property that, if it is satisfiable, it has a model where (i) all domains have the same cardinality, or (ii) all domains are infinite. Then ϕ and ϕ_{erased} are equisatisfiable.*

Proof. For (i), see Lemma 1 of [8]. For (ii), use the Löwenheim–Skolem theorem to get a model where all domains are countably infinite, then use (i). □

Lemma 4 (The cardinality of Horn formula models). *If an unsorted Horn formula has a model of cardinality at least 2, it also has an infinite model.*

[4] Also, no special support for tuples is needed in the theorem prover.

Proof. Suppose that ϕ is such a Horn formula. Since it has a model of cardinality at least 2, we know that $\phi \nvdash x = y$. We now make use of the fact that every satisfiable Horn formula has a minimal model. Take the signature of ϕ and adjoin a countably infinite set of constant symbols $c_1, c_2, \ldots$; this preserves satisfiability, and we claim that the resulting minimal model $\mathcal{M}$ is infinite.

Since none of the constants c_i occurs in ϕ, any resolution proof of $\phi \vdash c_i = c_j$ (for $i \neq j$) also proves $\phi \vdash x = y$. Since $\phi \nvdash x = y$, this implies that $\phi \nvdash c_i = c_j$, and hence in any minimal model $c_i \neq c_j$. Since $\mathcal{M}$ is minimal, none of the c_i are equal in $\mathcal{M}$, and so $\mathcal{M}$ is infinite. □

After these two lemmas, we are ready to prove the satefy of tupling for unsorted Horn formulas.

Lemma 5 (Safety of unsorted tupling for Horn formulas). *Let ϕ be an unsorted Horn formula and ϕ_{enc} be a transformed version in which we have applied (sorted) tupling. Then ϕ_{enc} and its sort erasure are equisatisfiable, i.e., the sorts can safely be erased.*

Proof. ϕ_{enc} has two sorts; let us call them ι and τ (for tuple). Since τ is a tuple sort it may always be interpreted by a tuple of domain elements of ι; in that case, if the cardinality of ι is κ, the cardinality of τ is κ^n for some n.

We are going to invoke Lemma 3. There are three cases, and in all of them the requirements of Lemma 3 hold:

- ϕ is unsatisfiable. In this case ϕ_{enc} is unsatisfiable too.
- ϕ has a model of cardinality 1. In this case, ϕ_{enc} has a model where ι has cardinality 1 and τ has cardinality $1^n = 1$.
- ϕ only has models of cardinality greater than 1. By Lemma 4, it has an infinite model, so ϕ_{enc} has a model where ι and τ are both infinite. □

3 Eliminating predicates

Equational provers do not usually support predicates. Before using the encodings of the previous section, we eliminate predicates in the standard way, by replacing them with functions: we introduce a new sort bool and a constant $\mathsf{true} : \mathsf{bool}$, and we replace each atomic formula $p(t_1, \ldots, t_n)$ with the equation $p(t_1, \ldots, t_n) = \mathsf{true}$ (p is now a function of result sort bool). We assume without proof that this well-known transformation, which we call *sorted predicate elimination*, is correct.

We would like to avoid introducing sorts if the input problem is unsorted. This suggests that we should do *unsorted* predicate elimination: the same transformation as above, but without introducing the bool sort. Unfortunately, this is not sound: the set of clauses $\{x = y, \neg p(a)\}$ is satisfiable, but $\{x = y, p(a) \neq \mathsf{true}\}$ is unsatisfiable because $x = y$ implies $p(a) = \mathsf{true}$. In fact, unsoundness occurs only when the input problem implies $x = y$, as the following lemma implies:

Lemma 6. *Let ϕ_{enc} be obtained from an unsorted formula ϕ by sorted predicate elimination. If ϕ has no model of size 1 then ϕ_{enc} and its sort erasure are equisatisfiable, i.e., the sorts can safely be erased.*

Proof (rather similar to Lemma 5). The formula ϕ_{enc} has two sorts; let us call them bool and ι. We show that either ϕ_{enc} is unsatisfiable or it has a model where bool and ι are infinite, and then invoke Lemma 3. There are two cases:

- If ϕ is unsatisfiable, then so is ϕ_{enc}.
- If ϕ is satisfiable, then by assumption it has a model of cardinality greater than 1. By Lemma 4, ϕ has an infinite model. Therefore ϕ_{enc} has a model where ι is infinite. This model can be extended to one where bool is also infinite, since the monotonicity test of [8] is satisfied. □

We can exploit Lemma 6 to eliminate predicates without introducing sorts, if the input problem is unsorted:

- Check if the input problem has a model of size 1. If so, abort the encoding and report that the formula is satisfiable.
- Otherwise, perform unsorted predicate elimination.

If the input problem is sorted, Lemma 6 does not help, so for sorted problems we always introduce the sort bool.

This leaves the problem of checking if the input formula has a model of size 1, which is easy to solve by observing that, in a model of size 1, all terms are equal and all predicates are constant-valued. To check if ϕ has a model of size 1, we replace all equality literals $t = u$ with $\mathbf{true}$, and all predicates $p(t_1, \ldots, t_n)$ with a Boolean variable p. We then check if the resulting propositional Horn formula is satisfiable, for example by doing unit propagation or using a SAT solver.

Example Given the Horn clauses $\{x = y, \neg p(a)\}$, we check if the set of clauses $\{\mathbf{true}, \neg p\}$ is satisfiable. It is, so the original problem has a model of size 1. Given the clauses $\{f(x) = f(y), p(a), \neg p(b)\}$, we check if $\{\mathbf{true}, p, \neg p\}$ is satisfiable. It is not, so we eliminate the predicates to get $\{f(x) = f(y), p(a) = \mathsf{true}, p(b) \neq \mathsf{true}\}$.

4 Encoding goal clauses

The goal clauses of a Horn formula are negated conjectures. By having several goal clauses of several literals each, a formula can have a conjecture which is an arbitrary Boolean combination (without negation) of positive literals. Most equational provers do not accept such expressive goals; some require the goal to be a single ground unit equation.

To solve this, we introduce two new constants $\mathsf{false}, \mathsf{true} : \mathsf{bool}$, where bool is the sort introduced by predicate elimination. We then replace each goal clause $C \to \mathbf{false}$ with the clause $C \to \mathsf{false} = \mathsf{true}$, and add one goal clause, $\mathsf{false} \neq \mathsf{true}$.

Some provers accept slightly more general goals, and the goal encoding can be adapted to the situation. For example, a Twee goal can be a disjunction of ground equations, so the transformation of this section is only used for non-ground conjectures, and tupling (Section 2.5) is used for ground conjectures that use conjunction. A Waldmeister goal can be a conjunction of ground equations, and so the transformation of this section must be used for disjunctions.

5 Evaluation

We evaluated our encodings on two equational theorem provers, Waldmeister [14] and Twee [20], and one first-order prover, E 2.0 [19], using the 2159 unsatisfiable Horn problems available in TPTP v7.0.0 [21]. We tried all four encodings, with tupling enabled and disabled. Each prover was allowed to run for five minutes. We ran Waldmeister with the flag `--auto` and E with the flag `--auto-schedule`. We ran Twee with a heuristic designed for Horn clauses, which we describe below.

The results are shown in Table 1, with the best result of each prover marked in bold. To provide a baseline for the comparison, we also gave the original unencoded problems to E, which solved 1972, and SPASS [22], which solved 1370.

We see that Twee solved more problems than Waldmeister, but both provers have respectable performance: as a prover for Horn problems, Waldmeister is about as powerful as SPASS, while Twee lies in between SPASS and E. Nonetheless, E is clearly better than Twee or Waldmeister at solving typical Horn problems. The results also show that the encoding has a cost: E solves about 300 fewer problems when the problems are encoded.

Prover	**Tupling?**	**Encoding**			
		1	2	3	4
Twee	No	1621	1596	1671	**1683**
	Yes	1534	1465	1493	1648
Waldmeister	No	1340	1246	1088	1151
	Yes	**1378**	1281	1176	1173
E	No	1698	**1710**	1672	1701
	Yes	1673	1676	1579	1615

Table 1. Number of problems solved using each encoding.

We also see that Twee and Waldmeister prefer entirely different encodings. Since the two provers use a similar proof procedure, the difference may lie in the heuristics used. Special heuristics for encoded Horn formulas are future work.

Rating 1 problems Together, the three provers solved the following 37 problems of rating 1.0, i.e., problems which are not currently solved by any automatic prover. All three provers solved several rating 1.0 problems; see Table 2 for details.

ALG212+1	ALG213+1	KLE077+1	KLE156+2	LAT064-1	LAT178-1
LAT180-1	LAT181-1	LAT184-1	LAT185-1	LAT186-1	LAT187-1
LAT188-1	LAT189-1	LAT190-1	LAT191-1	LAT193-1	LAT202-1
LAT203-1	LAT206-1	LAT207-1	LAT221-1	LAT224-1	LAT225-1
LAT226-1	LAT228-1	LAT229-1	LAT231-1	LAT242-1	LAT256-1
LCL147-1	LCL148-1	LCL151-1	REL020+1	REL040+3	REL040-4
REL041+1					

These problems come from several domains of the TPTP, but all consist mostly of equations, and most involve an algebraic structure having a rich equational theory. This suggests that the encodings are most effective on problems where the bulk of the reasoning is equational—as we might expect.

Prover	**Tupling?**	**Encoding**			
		1	2	3	4
Twee	No	**29**	19	24	18
	Yes	22	23	5	17
Waldmeister	No	9	10	5	5
	Yes	8	7	**15**	13
E	No	3	4	1	3
	Yes	3	5	1	**7**

Table 2. Number of rating-1 problems solved using each encoding.

A heuristic for Twee Twee employed a special heuristic designed to let it eagerly discharge preconditions of equations. Twee decides which critical pair to join next by picking the one with the lowest *score*, which is computed based on, among other things, the size of the critical pair. The heuristic is that, if we are scoring a term like $\mathsf{ifeq}(s,t,u,v) = v$, and s and t happen to be unifiable, we perform the unification but then count s and t as having zero size.

We use this heuristic in encodings 1, 2 and 3. In encoding 4, there is no way to apply it, since s and t are never present in the same term. With the heuristic enabled, Twee solves more rating 1 problems—but slightly fewer problems overall.

Satisfiable problems We also evaluated our encodings on the 312 satisfiable, Horn, non-unit equality problems in the TPTP. The results were that, regardless of the prover or the encoding used, we always solved 210–220 problems. The only exception was if-then without tupling, which solved about 190 problems depending on the prover; the problems lost were a large collection of rating-0 problems from `NLP`, such as `NLP002-1.p`. As a baseline, E with no encoding solved 195 problems, so we gained about 20 problems.

On closer inspection, it turned out that most of the problems gained had no goal clauses. Any such problem has a model of size 1, and our translation immediately reports it as satisfiable without even running the prover (see Section 3). This explains why the choice of encoding had almost no effect.

Twee was able to complete a few problems that E could not (such as `GRP112-1.p`, of rating 0.62), but all the problems that we inspected could be solved by Paradox [9] in under a second. Thus our translations are not very helpful for showing satisfiability. (We did not evaluate them on Paradox, but as the translations do not alter the size of any model we do not expect any effect.)

6 Discussion and related work

The proof search of a superposition-based prover working on a Horn problem, and a completion-based prover working on an encoded Horn problem, is quite similar. Superposition [4, 18] incorporates almost all the deduction rules of unfailing completion [3], only having a slightly stricter condition for backwards simplification. Thanks to literal selection, a superposition prover working on a Horn problem can work by forward reasoning from positive unit literals alone, much like a completion prover will do when using the "split if" encoding.

One important difference is that equational provers use more powerful redundancy tests. For example, ordered completion [2] allows a critical pair to be discarded if all of its ground instances can be joined (see e.g. [15]). A superposition prover would typically implement only a small subset of this feature, e.g., for handling commutative functions. Connectedness testing [1] is another powerful technique. Equational logic has a well-developed theory of proof orderings [2, 5, 6] which can be used to prove the correctness of many redundancy criteria.

There have been many and varied attempts to apply equational reasoning to first-order and Horn logic. As mentioned in the introduction, our first two encodings are inspired by work in universal algebra on encoding first-order logic as equations [7, 16]. Completion has been generalised to Horn clauses [10, 13, 3, 17]. Other term rewriting approaches to first-order logic include one using Boolean rings [11] and one using Gröbner bases [12].

7 Conclusion and future work

We have demonstrated that by encoding Horn formulas as equations, we can transform an equational theorem prover into a practical prover for Horn formulas. The resulting prover is strong on problems that combine difficult equational reasoning with some Horn clause reasoning. The encodings have a number of possible applications, including reasoning about functional programs, and reasoning about abstract algebra.

The encodings we presented do have some overhead. To eliminate this overhead, we plan to investigate hybrid approaches, where Horn clauses are encoded but the prover's strategy is specially tailored for reasoning about them—for example, by building certain equations into the prover or discarding unnecessary inferences. We hope that equational encodings of first-order logic [7, 16] can perhaps be made practical using such a hybrid approach.

Acknowledgements This work was supported by the Swedish Research Council (VR) grant 2016-06204, *Systematic testing of cyber-physical systems (SyTeC)*.

References

1. Bachmair, L., Dershowitz, N.: Critical pair criteria for completion. J. Symb. Comput. 6(1), 1–18 (Aug 1988), http://dx.doi.org/10.1016/S0747-7171(88)80018-X

2. Bachmair, L., Dershowitz, N.: Equational inference, canonical proofs, and proof orderings. J. ACM 41(2), 236–276 (Mar 1994)
3. Bachmair, L., Dershowitz, N., Plaisted, D.A.: Completion without failure. In: Rewriting Techniques, pp. 1–30. Elsevier (1989)
4. Bachmair, L., Ganzinger, H.: Rewrite-based equational theorem proving with selection and simplification. J. Log. Comput. 4, 217–247 (1994)
5. Bonacina, M.P., Dershowitz, N.: Abstract canonical inference. ACM Trans. Comput. Logic 8(1) (Jan 2007), http://doi.acm.org/10.1145/1182613.1182619
6. Bonacina, M.P., Hsiang, J.: Towards a foundation of completion procedures as semidecision procedures. Theoretical Computer Science 146(1-2), 199–242 (1995)
7. Burris, S.: Discriminator varieties and symbolic computation. J. Symb. Comput. 13(2), 175–207 (Feb 1992), http://dx.doi.org/10.1016/S0747-7171(08)80089-2
8. Claessen, K., Lilliesträm, A., Smallbone, N.: Sort it out with monotonicity. In: Bjørner, N., Sofronie-Stokkermans, V. (eds.) Automated Deduction – CADE-23. Lecture Notes in Computer Science, vol. 6803, pp. 207–221. Springer (2011)
9. Claessen, K., Sörensson, N.: New techniques that improve MACE-style finite model finding. In: Proceedings of the CADE-19 Workshop: Model Computation-Principles, Algorithms, Applications. pp. 11–27. Citeseer (2003)
10. Dershowitz, N.: A maximal-literal unit strategy for horn clauses. In: Kaplan, S., Okada, M. (eds.) Conditional and Typed Rewriting Systems. Lectures Notes in Computer Science, vol. 516, pp. 14–25. Springer (1991)
11. Hsiang, J.: Rewrite method for theorem proving in first order theory with equality. Journal of Symbolic Computation 3, 133–151 (02 1987)
12. Kapur, D., Narendran, P.: An equational approach to theorem proving in first-order predicate calculus. SIGSOFT Softw. Eng. Notes 10(4), 63–66 (Aug 1985)
13. Kounalis, E., Rusinowitch, M.: On word problems in horn theories. In: Lusk, E., Overbeek, R. (eds.) 9th International Conference on Automated Deduction. Lecture Notes in Computer Science, vol. 310, pp. 527–537. Springer (1988)
14. Löchner, B., Hillenbrand, T.: A phytography of WALDMEISTER. AI Commun. 15(2,3), 127–133 (Aug 2002)
15. Martin, U., Nipkow, T.: Ordered rewriting and confluence. In: Stickel, M.E. (ed.) 10th International Conference on Automated Deduction. Lecture Notes in Computer Science, vol. 449, pp. 366–380. Springer (1990)
16. Mckenzie, R.: On spectra, and the negative solution of the decision problem for identities having a finite nontrivial model. J. Symbolic Logic 40(2), 186–196 (06 1975), https://projecteuclid.org:443/euclid.jsl/1183739380
17. Nieuwenhuis, R., Orejas, F.: Clausal rewriting. In: Kaplan, S., Okada, M. (eds.) Conditional and Typed Rewriting Systems. Lecture Notes in Computer Science, vol. 516, pp. 246–258. Springer (1991)
18. Rusinowitch, M.: Theorem-proving with resolution and superposition. Journal of Symbolic Computation 11(1-2), 21–49 (1991)
19. Schulz, S.: System Description: E 1.8. In: McMillan, K., Middeldorp, A., Voronkov, A. (eds.) Proc. of the 19th LPAR, Stellenbosch. Lecture Notes in Computer Science, vol. 8312. Springer (2013)
20. Smallbone, N.: Twee, an equational theorem prover. http://nick8325.github.io/twee/ (2017)
21. Sutcliffe, G.: The TPTP Problem Library and Associated Infrastructure. From CNF to TH0, TPTP v6.4.0. Journal of Automated Reasoning 59(4), 483–502 (2017)
22. Weidenbach, C., Dimova, D., Fietzke, A., Kumar, R., Suda, M., Wischnewski, P.: Spass version 3.5. In: International Conference on Automated Deduction. Lecture Notes in Computer Science, vol. 5663, pp. 140–145. Springer (2009)