Exploiting sorts in expansion-based proof procedures

John Harrison

Intel Corporation, JF1-13
2111 NE 25th Avenue
Hillsboro OR 97124
johnh@ichips.intel.com

Abstract. It has long been recognized that formulating a problem in a many-sorted logic can make automated reasoning more efficient, essentially by dealing implicitly via typing with what would otherwise need explicit inference. We point out here that the advantages of a many-sorted formulation for automated reasoning can go further. Certain problems that appear to fall outside one of the known decidable subsets may not do so when considered with the additional refinement that a many-sorted formulation offers. We motivate this observation by considering theories such as metric spaces, and give some practical applications in verification.

1 Introduction

It's often natural in logical formalization to use more than one *type* or *sort*, to reflect intuitive distinctions between classes of objects. (We'll use 'type' and 'sort' more or less interchangeably in what follows.) For example, when formalizing geometry in traditional fashion [15] it's common to consider points (say P) and lines (L) as two quite different categories, and we can reflect this by formalization in a logic with more than one sort of object, *many-sorted* logic [19]. The property that any two distinct points determine a unique line might be stated as follows, where the binary predicate On (asserting that a point lies on a line) requires its first argument to be of type P and its second to be of type L.

$$\forall x : P,\ y : P.\ \neg(x = y) \Longrightarrow \exists! l : L.\ \mathrm{On}(x, l) \wedge \mathrm{On}(y, l)$$

In most of the logic and automated reasoning literature, many-sorted first-order logic is mentioned at best in passing. This is understandable, since rigorous definitions of model, validity etc. become more technically involved with multiple sorts. And as the books often note, the apparatus of sorts is in principle not necessary because they can be replaced by relativization of quantifiers using unary predicates, e.g.

$$\forall x,\ y.\ P(x) \wedge P(y) \wedge \neg(x = y) \Longrightarrow \exists! l.\ L(l) \wedge \mathrm{On}(x, l) \wedge \mathrm{On}(y, l)$$

It has long been recognized, though, that a many-sorted formulation often allows more efficient proof automation; both [9] and [31] illustrate this point for Schubert's classic steamroller problem. Various steps that would need to be justified by an explicit inference rule (say, a resolution step) in a one-sorted formulation with relativized quantifiers can just happen implicitly as part of the type distinction.

However, once a many-sorted *formulation* is arrived at, what is the contribution of the types to the subsequent process of proof search? Most traditional first-order proof algorithms such as resolution are based on unification. Properties of unification in many-sorted logic are highly dependent on the nature of the sorts; for example whether there are polymorphic types or whether the types are arranged in a hierarchy [32]. But for the simple and common case where there is a fixed finite set of sorts, all distinct, little seems to be lost for most proof procedures by ignoring or erasing the sorts. Conventional first-order unification applied to well-typed terms will always result in well-typed instantiations, so the algorithm will never waste time exploring ill-typed intermediate formulas, even if the typing is implicit.

The main thesis of this paper is that for proof search algorithms based on explicit *expansion*, rather than on unification, there *is* often much to be gained from considering the sorts during proof search. But why should we be interested in expansion-based proof procedures, when the most successful proof procedures are based on unification anyway? Well, the pre-eminence of unification-based methods has recently been challenged by the development of some other methods such as ordered semantic hyper-linking [23]. And on general grounds, it is worth trying to exploit the enormous strides made recently in the efficiency of checkers for SAT (propositional satisfiability) and SMT (satisfiability modulo theories). It's very attractive to organize a prover based on an initial expansion step followed by one of these 'big guns'. A successful example of this philosophy is the aggressively "eager" expansion of problems into SAT performed by UCLID [8].

The combination of quantifier reasoning and reasoning in theories like arithmetic is often important in applications, yet the existing approaches tend to be heuristic and often incomplete [5]. We observe that for expansion-based proof procedures, the finer distinctions offered by many-sorted logic can make a significant difference. Indeed, by giving us a more refined notion of quantifier prefix, we can see at once the decidability of some problems that might naively seem to be outside the known decidable fragments.

2 The AE fragment

We start by reminding the reader about the decidability of validity for the 'AE fragment' of first-order logic. A formula is in the AE fragment when it contains no constants or function symbols, and is in (or can obviously be put into) prenex normal form with the following quantifier structure:

$$\forall x_1, \ldots, x_n.\, \exists y_1, \ldots, y_m.\, P[x_1, \ldots, x_n, y_1, \ldots, y_m]$$

with $P[x_1, \ldots, x_n, y_1, \ldots, y_m]$ quantifier-free, for some $m \geq 0$ and $n \geq 0$. We can allow individual constants if we regard them as additional universally quantified variables in the list $x_1, \ldots, x_n$, but we need to account for them in the number n. The decidability result holds for pure first-order logic [4] and for first-order logic with equality [25]; in fact it is easy to deduce the latter from the former because we can adjoin equivalence and congruence axioms for equality without disturbing the AE form. There are two natural ways of showing the decidability of the AE fragment. In each case we actually consider the dual problem of satisfiability of the following formula ϕ, where Q is $\neg P$.

$$\exists x_1, \ldots, x_n.\, \forall y_1, \ldots, y_m.\, Q[x_1, \ldots, x_n, y_1, \ldots, y_m]$$

The first approach is based on the following observation: ϕ has a model iff it has a model with domain size n.[1] The right-to-left direction is trivial. For the other direction, suppose we have any model M of ϕ with domain D. Then since ϕ holds there are elements $a_1, \ldots, a_n \in D$ such that for any $b_1, \ldots, b_m \in D$ we have $Q_M[a_1, \ldots, a_n, b_1, \ldots, b_m]$, where Q_M represents the interpretation of the formula Q in the model. We claim that M', the restriction of M to the domain $D' = \{a_1, \ldots, a_n\}$, is also a model of ϕ. Indeed, we still have the elements $a_1, \ldots, a_n \in D'$ as required, and since $Q_M[a_1, \ldots, a_n, b_1, \ldots, b_m]$ then holds for all $b_1, \ldots, b_m \in D$, it holds *a fortiori* for all $b_1, \ldots, b_m \in D'$. Note that this reasoning relies on the quantifier prefix, and fails with function symbols, since we have no reason to assume D' is closed under their interpretations f_M.

The other approach, more in the spirit of automated theorem proving methods, is to use Skolemization. By the basic theorem that Skolemization preserves satisfiability, ϕ is satisfiable iff the formula $\forall y_1, \ldots, y_m.\, Q[c_1, \ldots, c_n, y_1, \ldots, y_m]$ is satisfiable, where $c_1, \ldots, c_n$ are new constants. But by the Skolem-Gödel-Herbrand theorem, this is equivalent to the (propositional) satisfiability of the n^m-fold conjunction

$$\bigwedge_{t_1, \ldots, t_n} Q[c_1, \ldots, c_n, t_1, \ldots, t_m]$$

where each t_i ranges over the Herbrand base $\{c_1, \ldots, c_n\}$. Again this fails if we have function symbols, because the Herbrand base is infinite, containing all terms built up using the functions, say 0, $0 + 0$, $0 + (0 + 0)$ etc. for a binary function symbol '+'. And the wrong kind of quantifier alternation at the beginning would give rise to new functions after Skolemization.

The decidability of the AE fragment can also be used to decide monadic first-order logic by prenexing in the right way, and some not entirely trivial problems can be stated in it, such as the following 'nonobvious' puzzle due to Łoś:

[1] Unless we want to allow empty models, we should assume $n \geq 1$; we can always add a vacuous quantifier if we wish; similarly we customarily add one constant if the Herbrand base is empty.

$$
\begin{array}{l}
(\forall x\ y\ z.\ P(x,y) \wedge P(y,z) \Longrightarrow P(x,z)) \wedge \\
(\forall x\ y\ z.\ Q(x,y) \wedge Q(y,z) \Longrightarrow Q(x,z)) \wedge \\
(\forall x\ y.\ P(x,y) \Longrightarrow P(y,x)) \wedge \\
(\forall x\ y.\ P(x,y) \vee Q(x,y)) \\
\Longrightarrow (\forall x\ y.\ P(x,y)) \vee (\forall x\ y.\ Q(x,y))
\end{array}
$$

which is in fact valid. However, the AE fragment is not generally thought to be very useful in practice, for example in direct verification applications or in translating other logical problems of interest.

3 The benefits of many-sortedness

The decidability of the AE fragment generalizes straightforwardly to the many-sorted case. We will not give the technical details, but in terms of Skolemization, the core result is exactly the same: we can test the conjunction of all substitution instances of the formula over the Herbrand base. The only difference is that the Herbrand base is generated with type distinctions and so we can exclude certain ill-typed terms from consideration and give a smaller instantiation. In particular (in terms of the earlier pattern) when considering instantiations of the body Q, we only need to consider instantiating each y_i to the subset of elements of $\{c_1, \ldots, c_n\}$ of the same type as y_i.

This observation has already been exploited by Jereslow [16] to give more efficient decision procedures for the AE subset. Our interest here will be in cases where the type distinctions allow us to go *outside* the obvious AE subset, for example in adding function symbols, while still maintaining a finite Herbrand base because of type distinctions. This possibility is implicit in work on parametrized system verification by Pnueli et al [24], and our goal here is to present the approach in its logical generality. We will in fact consider parametrized system verification as one of the examples. But we also show how it can be used as a general way of performing complete quantifier instantiation in problems with a certain logical structure.

There are many suites of decision procedures for validity in quantifier-free combinations of theories, based either on combination methods due to Nelson and Oppen [21], or those due to Shostak [27], which suitably corrected [26] can be regarded as special-case optimizations of the Nelson-Oppen method [2]. However, it is easy to see that in a one-sorted formulation, any serious extension to a richer quantifier structure leads to undecidability. (For example, in Presburger arithmetic with one uninterpreted function symbol, we can define multiplication.) Thus, while some decision procedure suites are able to perform quantifier instantiation, the methods remain heuristic and with somewhat ill-defined ranges of applicability; cf. for example section 5 of [11] or section 2.2 of [18]. Bjørner and Stickel [5] approach the problem from the other end, and look at incorporating arithmetical reasoning into traditional first-order proof methods, but again the methods are heuristic and incomplete.

Therefore it seems there is considerable value in identifying cases, as we do here, where on a more refined (type-aware) view, we can extend these decision procedures to handle richer quantifier structure. This could not only lead to new approaches to quantifier instantiation in combined decision procedure suites [3, 11, 18], but even explain the success or failure of existing heuristics, and allow us to deduce invalidity from their failure.

A final observation is that even if we are faced with a problem in untyped first-order logic, say in the TPTP library [29] we can easily *invent* types for ourselves, which may prove to be beneficial in the sense we have described. Essentially, we can just use the Hindley-Milner 'most general type' algorithm [20] often used for type assignment in functional languages. We have not experimented with how much is to be gained by this method, but it would be interesting to try some examples. When naturally many-sorted problems are formulated in one-sorted logic, one sometimes adds "ill-typed" axioms such as 'every x is either a point or a line'. One would need to carefully delete or modify such axioms to recover the many-sorted formulation.

4 Metric spaces

As a motivating example, consider a theory of metric spaces. A metric is a binary function $d : S \times S \to \mathbb{R}$ from some set S into the reals satisfying the following properties (one can find slightly simpler equivalents) whose conjunction we write M:

$$\begin{array}{l}\forall x\ y : S.\ d(x,y) \geq 0\\ \forall x\ y : S.\ d(x,y) = 0 \Leftrightarrow x = y\\ \forall x\ y : S.\ d(x,y) = d(y,x)\\ \forall x\ y\ z : S.\ d(x,z) \leq d(x,y) + d(y,z)\end{array}$$

Intuitively, we think of $d(x, y)$ as a notion of distance between x and y, and the classic example is the usual Euclidean metric $d(x, y) = \|x - y\|$ on $S = \mathbb{R}^N$. But there are many other possible metrics, including the near-degenerate case of the discrete metric

$$d(x,y) = \begin{cases} 0 \text{ if } x = y \\ 1 \text{ if } x \neq y \end{cases}$$

So consider a 2-sorted first-order theory with types S and $\mathbb{R}$ and a metric function $d : S \times S \to \mathbb{R}$ as well as the usual constants (0, 1, ...), functions ('+', '×', ...) and relations ('≤', '=') on $\mathbb{R}$. We assume all the real operations are interpreted in the standard way, but that S and the function d are allowed an arbitrary interpretation. Now, a formula holds in all metric spaces iff it follows from the metric axiom M above with the usual interpretation of the real-number operations, or equivalently, follows logically from $R \cup \{M\}$ where R is the set of all first-order properties that hold in $\mathbb{R}$. (We could equally well use a more economical axiomatization of the same theory for R).

So let us consider whether certain formulas hold in all metric spaces. One way of generating interesting questions is to define in metrical terms the two primitive notions of 'betweenness' and 'congruence' from Tarski's system of geometry [30]:

$$\begin{aligned} B(x,y,z) &=_{def} d(x,y) + d(y,z) = d(x,z) \\ (u,v) \equiv (x,y) &=_{def} d(u,v) = d(x,y) \end{aligned}$$

and see which properties of them follow from the metric axioms alone. For example, $B(a,b,a) \Longrightarrow a = b$ (Axiom 6, the identity axiom for betweenness) becomes:

$$\forall a\ b.\ d(a,b) + d(b,a) = d(a,a) \Longrightarrow a = b$$

It is pretty easy to prove that this does follow: we have $d(a,a) = 0$, and $d(a,b) + d(b,a) = 0$ implies, since $d(a,b) \geq 0$ and $d(b,a) \geq 0$, that $d(a,b) = d(b,a) = 0$. (Alternatively we could use symmetry to argue $2 \cdot d(a,b) = 0$ and so $d(a,b) = 0$.) And that means that $a = b$ as required. But now let us consider how an automated algorithm could approach this problem and a whole class of problems with the same logical structure in a uniform way.

$$R \cup \{M\} \vdash \forall a\ b.\ d(a,b) + d(b,a) = d(a,a) \Longrightarrow a = b$$

or in other words

$$R \vdash M \Longrightarrow \forall a\ b.\ d(a,b) + d(b,a) = d(a,a) \Longrightarrow a = b$$

which we can write

$$\begin{aligned} R \vdash\ & (\forall x\ y : S.\ d(x,y) \geq 0) \wedge \\ & (\forall x\ y : S.\ d(x,y) = 0 \Leftrightarrow x = y) \wedge \\ & (\forall x\ y : S.\ d(x,y) = d(y,x)) \wedge \\ & (\forall x\ y\ z : S.\ d(x,z) \leq d(x,y) + d(y,z)) \\ & \Longrightarrow \forall a\ b.\ d(a,b) + d(b,a) = d(a,a) \Longrightarrow a = b \end{aligned}$$

It is not hard to see that we can if we wish put the conclusion into prenex AE form; the ranges of the quantifier blocks are disjoint so we can pull them out in the order $\forall a\ b.\ \exists x\ y\ z.\ \cdots$ (since the antecedent is effectively negative, those become existential quantifiers). We won't bother with that, but consider directly how to obtain a contradiction from the Skolemized clauses, where a and b are Skolem constants, and the real-number axioms are also Skolemized to give R^*:

$$\begin{aligned} R^* \cup \{\ & (\forall x\ y.\ d(x,y) \geq 0), \\ & (\forall x\ y.\ d(x,y) = 0 \Leftrightarrow x = y), \\ & (\forall x\ y.\ d(x,y) = d(y,x)), \\ & (\forall x\ y\ z.\ d(x,z) \leq d(x,y) + d(y,z)), \\ & d(a,b) + d(b,a) = d(a,a), \\ & \neg(a = b) \end{aligned} \quad \}$$

By compactness and the usual Herbrand-type argument, this is contradictory iff some conjunction of ground instances of these formulas is contradictory. And now sorts come to our rescue. The *only* terms in the Herbrand base of type S are the constants a and b; by contrast, on a naive unsorted view, we would have to consider $d(a, d(a, a))$ and infinitely many other terms. On the other hand, R^* trivially implies all its instances, so it's equivalent to obtaining a contradiction from R^* together with all ways of instantiating universally quantified variables of type S to the two constants a and b:

$$
\begin{aligned}
R^* \cup \{\, & d(a,a) \geq 0, && \} \\
& d(a,a) = 0 \Leftrightarrow a = a, \\
& d(a,a) = d(a,a), \\
& d(a,a) \leq d(a,a) + d(a,a), \\
& d(a,b) \leq d(a,a) + d(a,b), \\
& d(a,b) \geq 0, \\
& d(a,b) = 0 \Leftrightarrow a = b, \\
& d(a,b) = d(b,a), \\
& d(a,a) \leq d(a,b) + d(b,a), \\
& d(a,b) \leq d(a,b) + d(b,b), \\
& d(b,a) \geq 0, \\
& d(b,a) = 0 \Leftrightarrow b = a, \\
& d(b,a) = d(a,b), \\
& d(b,a) \leq d(b,a) + d(a,a), \\
& d(b,b) \leq d(b,a) + d(a,b), \\
& d(b,b) \geq 0, \\
& d(b,b) = 0 \Leftrightarrow b = b, \\
& d(b,b) = d(b,b), \\
& d(b,a) \leq d(b,b) + d(b,a), \\
& d(b,b) \leq d(b,b) + d(b,b), \\
& d(a,b) + d(b,a) = d(a,a), \\
& \neg(a = b)
\end{aligned}
$$

this is equivalent to deriving a contradiction from the enumerated set of formulas over $\mathbb{R}$. While it looks a bit unwieldy on the printed page, this is tiny compared with the expansions routinely performed in UCLID [8], and it should be an easy problem for this or any of the major combined decision procedure suites. While this was a straightforward example, the crucial points are:

- The method is complete, so we can actually deduce that something does *not* hold in all metric spaces from its failure.
- The method is applicable to decide any formula with an AE quantifier structure over the type S and to other axioms with analogous logical form.

For an example of the first, we can confirm using the same method that $B(a, b, d) \wedge B(b, c, d) \Longrightarrow B(a, b, c)$ (Tarski's Axiom 15, inner transitivity of betweenness) does *not* hold in all metric spaces. For an example of the second, we

can call on a presumably apocryphal story[2] about a student whose thesis topic was the theory of 'antimetric' spaces, which arise from reversing the inequality in the triangle law, and considering the following set of axioms (we don't need symmetry for the story to work):

$$\begin{array}{l}(\forall x\ y.\ d(x,y) = 0 \Leftrightarrow x = y) \wedge \\ (\forall x\ y.\ 0 \leq d(x,y)) \wedge \\ (\forall x\ y\ z.\ d(x,y) + d(y,z) \leq d(x,z))\end{array}$$

Our student developed an impressive theory of these structures, including many beautiful theorems. However, just before his PhD examination, his thesis was criticized on the grounds that it contained too much abstract development and not enough motivation in terms of practical examples. Accordingly, the student worked hard to find some concrete examples of antimetric spaces, only to discover to his horror that all antimetric spaces have at most one element and hence that his entire thesis was mathematically trivial. He could have shown this automatically using our decision method, which gives the following contradictory expansion:

$$\begin{array}{l}(d(a,a) = 0 \Leftrightarrow a = a) \wedge (0 \leq d(a,a)) \wedge \\ (d(a,a) + d(a,a) \leq d(a,a)) \wedge (d(a,a) + d(a,b) \leq d(a,b)) \wedge \\ (d(a,b) = 0 \Leftrightarrow a = b) \wedge (0 \leq d(a,b)) \wedge \\ (d(a,b) + d(b,a) \leq d(a,a)) \wedge (d(a,b) + d(b,b) \leq d(a,b)) \wedge \\ (d(b,a) = 0 \Leftrightarrow b = a) \wedge (0 \leq d(b,a)) \wedge \\ (d(b,a) + d(a,a) \leq d(b,a)) \wedge (d(b,a) + d(a,b) \leq d(b,b)) \wedge \\ (d(b,b) = 0 \Leftrightarrow b = b) \wedge (0 \leq d(b,b)) \wedge \\ (d(b,b) + d(b,a) \leq d(b,a)) \wedge (d(b,b) + d(b,b) \leq d(b,b)) \wedge \\ \neg(a = b)\end{array}$$

Again, any good combined decision procedure suite should find it easy to prove that this is contradictory. Indeed, replacing $a = a$ and $b = b$ by $\top$ and $a = b$ and $b = a$ by $\bot$ and considering terms $d(x,y)$ as new variables, it is directly contradictory over $\mathbb{R}$.

5 Parametrized system verification

We now turn to the application where earlier work [24] anticipates our present observations, parametrized system verification. The typical problem in this area is that we have a system with a number N of equivalent replicated components, so the state space involves some Cartesian product

$$\Sigma = \Sigma_0 \times \overbrace{\Sigma_1 \times \cdots \times \Sigma_1}^{N \text{ times}}$$

[2] I heard it from Per Bjesse, who got it from John Hughes, who heard it when an undergraduate in Cambridge.

and the transition relation is symmetric between the replicated components, so if π is any[3] permutation of $\{1, \ldots, N\}$ we have

$$(\sigma_0, \sigma_1, \ldots, \sigma_n) \longrightarrow (\sigma'_0, \sigma'_1, \ldots, \sigma'_n)$$

if and only if

$$(\sigma_0, \sigma_{\pi(1)}, \ldots, \sigma_{\pi(n)}) \longrightarrow (\sigma'_0, \sigma'_{\pi(1)}, \ldots, \sigma'_{\pi(n)})$$

In fact, we will usually consider the case where the transition relation is simply a disjunction of the component transition relations, which automatically obeys strong symmetry. As usual with verification of transition systems, our main interest is in proving 'safety' properties of the form 'starting in a state satisfying p, we cannot reach a state satisfying q'. If we assume that Σ_0 and Σ_1 are finite, then in principle such questions are decidable for any *particular* value of N, say $N = 3$, and can be decided by ordinary model checking. However, this might in practice not be feasible when N is large enough. Besides, the system may be such that the particular value N doesn't (or shouldn't) matter and we would like to have a general proof for *all* N.

A classic example of such a parametrized system is a cache coherence protocol for a multiprocessor system, where each of the N processors (or 'cacheing agents') has its own private cache; we will use a simple MESI protocol [14] as an example. For simplicity, we assume that all cache lines are handled independently, so we can consider just the state connected with a given cache line ($\approx$ memory address). We model the state space for each cacheing agent w.r.t. that line by a 4-element enumerated type `State` with these members (hence the name 'MESI')

- `Modified` — Only this agent has a valid copy; contents have been written more than once by it.
- `Exclusive` — Only this agent has a valid copy; contents have been written exactly once by it.
- `Shared` — This agent shares a copy with at least one other agent, and all agree.
- `Invalid` — This agent's copy is not up to date with the latest write.

At this level of abstraction, the state space of the entire system can be modelled as $\texttt{State}^N$, i.e. as an N-element array `Cache` of states, taking the indices to come from an indexing type `Node`. And we can model the protocol's permissible transitions by logical formulas relating the initial and final arrays `Cache` and `Cache'`. The overall transition relation is the disjunction of various possibilities:

- A read hit on some cache i has no effect; it can only happen if the contents are valid:

  $\exists$`i:Node. ¬(Cache(i) = Invalid)` $\wedge$ `Cache' = Cache`

[3] We sometimes want to consider restrictions to subgroups of permutations, e.g. circular symmetry when we consider components on a ring network, but here we'll only consider the case of full symmetry.

– A read miss on cache i: we get a shared copy and any other copies are changed to 'shared':

```
∃i:Node. Cache(i) = Invalid ∧
         Cache'(i) = Shared ∧
         ∀j:Node. ¬(j = i)
                  ⟹ Cache'(j) = if Cache(j) IN {Modified,Exclusive}
                                then Shared
                                else Cache(j)
```

– A write miss on cache i: cache i gets an exclusive copy and all other copies are invalidated.

```
∃i:Node. Cache(i) = Invalid ∧
         Cache'(i) = Exclusive ∧
         ∀j:Node. ¬(j = i) ⟹ Cache'(j) = Invalid
```

– A write hit on cache i. If the original line was exclusive, change it to modified. Otherwise change it to exclusive an invalidate all other caches.

```
∃i:Node.
    ¬(Cache(i) = Invalid) ∧
    (if Cache(i) = Exclusive
     then Cache'(i) = Modified ∧
          ∀j. ¬(j = i) ⟹ Cache'(j) = Cache(j)
     else if Cache(i) = Modified then Cache' = Cache
     else Cache'(i) = Exclusive ∧
          ∀j. ¬(j = i) ⟹ Cache'(j) = Invalid)
```

Let's assume that initially all caches are in the invalid state. We would like to show that in all states reachable in arbitrarily many steps of the transition relation, coherence holds, meaning that if any cache has an exclusive or modified copy, all other caches are in the invalid state:

```
∀i:Node. Cache(i) IN {Modified, Exclusive}
         ⟹ ∀j. ¬(j = i) ⟹ Cache(j) = Invalid
```

One approach is to exploit symmetry to show that the system is simulated by an abstracted one that just counts how many caches are in each state, the so-called 'counter abstraction':

```
trans (m,e,s,i) (m',e',s',i') ⇔
  m + e + s >= 1 ∧ m' = m ∧ e' = e ∧ s' = s ∧ i' = i ∨
  i >= 1 ∧ i' = i - 1 ∧ e' = 0 ∧ m' = 0 ∧ s' = s + e + m + 1 ∨
  m >= 1 ∧ m' = m ∧ e' = e ∧ s' = s ∧ i' = i ∨
  e >= 1 ∧ e' = e - 1 ∧ m' = m + 1 ∧ s' = s ∧ i' = i ∨
  s >= 1 ∧ s' = 0 ∧ e' = 1 ∧ m' = 0 ∧ i' = (i + m + e + s) - 1 ∨
  i >= 1 ∧ e' = 1 ∧ s' = 0 ∧ m' = 0 ∧ i' = (i + e + m + s) - 1
```

Now the coherence property of the original system follows from an analogous property for the abstracted one: $m + e \geq 1 \Longrightarrow m + e + s = 1$. The latter can be established using tools for reachability over subsets of Euclidean space [10]. Actually, in this case the situation is simpler: coherence is already an inductive invariant, so we can just reason by induction using the arithmetical fact:

```
|- (m + e >= 1 ⟹ m + e + s = 1) ∧
   trans (m,e,s,i) (m',e',s',i')
   ⟹ m' + e' >= 1 ⟹ m' + e' + s' = 1
```

We will now show that using our approach one can establish exactly the same inductive invariance property for the original system *without* making the counter abstraction. We can split the problem into four distinct cases for the four disjuncts of the transition relation; for example the 'write miss on cache i' case needs us to establish:

```
(∀i. Cache(i) IN {Modified, Exclusive}
     ⟹ ∀j. ¬(j = i) ⟹ Cache(j) = Invalid) ∧
(∃i. Cache(i) = Invalid ∧
     Cache'(i) = Exclusive ∧
     ∀j:Node. ¬(j = i) ⟹ Cache'(j) = Invalid)
⟹ (∀i. Cache'(i) IN {Modified, Exclusive}
         ⟹ ∀j. ¬(j = i) ⟹ Cache'(j) = Invalid)
```

This is, in fact, pretty trivial. However, note how we can solve it *systematically*. We want to obtain a contradiction from the negated Skolemized form, where `n`, `p` and `q` are Skolem constants:

```
(∀i j. (Cache(i) = Modified ∨ Cache(i) = Exclusive) ∧ ¬(j = i)
       ⟹ Cache(j) = Invalid) ∧
(Cache(n) = Invalid ∧ Cache'(n) = Exclusive ∧
(∀j. ¬(j = n) ⟹ Cache'(j) = Invalid)) ∧
(Cache(p) = Modified ∨ Cache(p) = Exclusive) ∧ ¬(q = p) ∧
¬(Cache(q) = Invalid)
```

As usual, it's equivalent to getting a contradiction from the set of all ground instances. And here, once again, the sorts come to our rescue. In a naive unsorted first-order formulation, we would consider all potential instantiations of these variables with terms of the form:

```
n, Cache(n), Cache(Cache(n)), Cache(Cache(Cache(n))), ...
```

But `Cache` has type `Node` → `State`, so in the sorted version, only two of these terms make sense and only one has type `Node`. So the only possible instantiations for the universal quantifiers are the three Skolem constants. Thus we just need to get a contradiction from

```
((Cache(n) = Modified ∨ Cache(n) = Exclusive) ∧ ¬(n = n)
 ⟹ Cache(n) = Invalid) ∧
((Cache(n) = Modified ∨ Cache(n) = Exclusive) ∧ ¬(p = n)
 ⟹ Cache(p) = Invalid) ∧
((Cache(n) = Modified ∨ Cache(n) = Exclusive) ∧ ¬(q = n)
 ⟹ Cache(q) = Invalid) ∧
((Cache(p) = Modified ∨ Cache(p) = Exclusive) ∧ ¬(n = p)
 ⟹ Cache(n) = Invalid) ∧
((Cache(p) = Modified ∨ Cache(p) = Exclusive) ∧ ¬(p = p)
 ⟹ Cache(p) = Invalid) ∧
((Cache(p) = Modified ∨ Cache(p) = Exclusive) ∧ ¬(q = p)
 ⟹ Cache(q) = Invalid) ∧
((Cache(q) = Modified ∨ Cache(q) = Exclusive) ∧ ¬(n = q)
 ⟹ Cache(n) = Invalid) ∧
((Cache(q) = Modified ∨ Cache(q) = Exclusive) ∧ ¬(p = q)
 ⟹ Cache(p) = Invalid) ∧
((Cache(q) = Modified ∨ Cache(q) = Exclusive) ∧ ¬(q = q)
 ⟹ Cache(q) = Invalid) ∧
(Cache(n) = Invalid ∧ Cache'(n) = Exclusive ∧
(¬(n = n) ⟹ Cache'(n) = Invalid)) ∧
(¬(p = n) ⟹ Cache'(p) = Invalid)) ∧
(¬(q = n) ⟹ Cache'(q) = Invalid)) ∧
(Cache(p) = Modified ∨ Cache(p) = Exclusive) ∧ ¬(q = p) ∧
¬(Cache(q) = Invalid)
```

This is within the capabilities of standard free-variable decision procedure. We could even use pure logical reasoning if we're prepared to add axioms asserting that each object of type `State` is one of the four indicated values, and that all these are distinct. However, an appealing point is that we can include some arithmetic or other theory reasoning as well, and our expansion technique still works well. Observe that all we required of the logical form is:

- The transition relation and invariant only involve quantifiers over nodes.
- The invariant has no quantifier alternations (purely universal or purely existential).
- The transition relation has the form $\exists \cdots \exists \forall \cdots \forall$.

Such invariants, like cache coherence, are common in practice. And the transition relation arising from a system with many duplicated components very naturally has the form indicated. An outer existential quantifier arises because the overall transition relation is a nondeterministic choice between all the replicated components. The universal quantifiers arise because we need to specify array updates by quantification over indices; in the case of an unchanged array we might write $\forall i.\, a[i] = b[i]$ for the higher-order statement $a = b$.

The MESI protocol verification here is trivial, just for illustration. In more interesting protocol verification examples, the various "transactions" are decomposed into multiple communication primitives and so can end up interleaving

in complicated ways. Possibly the simplest standard example that is realistic in this sense is German's protocol. Here, coherence itself is not an inductive property, and one needs to find a stronger inductive invariant. In [24] an invariant is constructed automatically by generalizing the reachable states of a finite configuration, and then proven automatically techniques close to those we are describing here. More 'human-oriented' ways of synthesizing inductive invariants are discussed in [22], and we emphasize that *any* inductive invariants for German that have no quantifier alternations can be proven automatically by our techniques, however they are generated.

However, the method fails if the system involves arrays that contain other node indices. For at that point, we once again find ourselves in a situation with an infinite Herbrand universe. This essentially explains why the method in [24] is not immediately applicable to some other protocols that do have that feature, e.g. the Stanford FLASH protocol.

6 Conclusions

It has long been observed that certain metatheorems are stronger in their many-sorted formulations, in particular the interpolation property [12, 13, 17]. This paper notes a similar improvement for automated reasoning. We have pointed out how the use of sorts, generally neglected in automated theorem proving, can be quite important, rendering certain problems — including some practically interesting ones — decidable. Although the idea is implicit in [24], we think it is beneficial to separate it out from the particular application and present it in full logical generality. It would be interesting to see if there are any significant ways the result can be extended, e.g. finding analogs of these results for typed versions of other decidable classes [6]. And it would be interesting to consider subtler arguments than those based on sorts about why in certain cases instantiation with finite subsets of the Herbrand base are sufficient for completeness. Quantifier instantiation heuristics (of the kind already used in UCLID and Simplify) could be enhanced with this deeper understanding.

Another interesting question is the relationship with the theory of arrays. The earliest and best-known decision procedure for the extensional theory of arrays [28] explicitly formulates the theory in 2-sorted terms with a type I for indices and another type V of values, but these need not be distinct. As usual, the decision procedure is restricted to quantifier-free formulas, and as the authors point out 'single-sorted first-order theories with function symbols and equality may be translated into this theory ... even quite restricted quantified fragments ... are undecidable'. It would be interesting to see to what extent enforcing a strict separation of the types I and V allows one to introduce some limited form of quantification over indices into the theory. We would also like to see if we can understand some of the decidability and undecidability results given in [7] in terms of type distinctions.

Acknowledgements

I would like to thank Jesse Bingham, Clark Barrett, Ching Tsun Chou, Sava Krstić and all the regular members of the Protocol Study Group for valuable discussions.

NOTES TO MYSELF

*** John Matthews pointed out to me that all this may have been anticipated by Pascal Fontaine, who has very similar (and maybe better) stuff in his thesis. He's one of the "Harvey" people.

**** Actually, I presented his TACAS paper at the Protocol Study Group, and indeed it does largely seem to subsume all I have here. So I can forget about publication now.

References

1. *Proceedings of the Sixteenth Annual IEEE Symposium on Logic in Computer Science*. IEEE Computer Society Press, 2001.
2. C. Barrett. *Checking Validity of Quantifier-Free Formulas in Combinations of First-Order Theories*. PhD thesis, Stanford University Computer Science Department, 2002.
3. C. Barrett and S. Berezin. CVC Lite: A new implementation of the cooperating validity checker. In R. Alur and D. A. Peled, editors, *Computer Aided Verification, 16th International Conference, CAV 2004*, volume 3114 of *Lecture Notes in Computer Science*, pages 515–518, Boston, MA, 2004. Springer-Verlag.
4. P. Bernays and M. Schönfinkel. Zum Entscheidungsproblem der mathematischen Logik. *Mathematische Annalen*, 99:401–419, 1928.
5. N. S. Bjørner, M. E. Stickel, and T. Uribe. A practical integration of first-order reasoning and decision procedures. In W. McCune, editor, *Automated Deduction — CADE-14*, volume 1249 of *Lecture Notes in Computer Science*, pages 101–115, Townsville, Australia, 1997. Springer-Verlag.
6. E. Börger, E. Grädel, and Y. Gurevich. *The classical Decision Problem*. Springer-Verlag, 2001.
7. A. R. Bradley, Z. Manna, and H. B. Sipma. What's decidable about arrays? In E. A. Emerson and K. S. Namjoshi, editors, *Verification, Model Checking, and Abstract Interpretation, 7th International Conference, VMCAI 2006*, volume 3855 of *Lecture Notes in Computer Science*, pages 427–442, Charleston, SC, 2006. Springer-Verlag.
8. R. E. Bryant, S. K. Lahiri, and S. A. Seshia. Modeling and verifying systems using a logic of counter arithmetic with lambda expressions and uninterpreted functions. In E. Brinksma and K. G. Larsen, editors, *Computer Aided Verification, 14th International Conference, CAV 2002*, volume 2404 of *Lecture Notes in Computer Science*, pages 79–92, Copenhagen, Denmark, 2002. Springer-Verlag.
9. A. G. Cohn. On the solution of Schubert's steamroller in many sorted logic. In *Proceedings of the 9th International Joint Conference on Artificial Intelligence*, pages 1169–1174, 1985.

10. G. Delzanno. Automatic verification of parameterized cache coherence protocols. In E. A. Emerson and A. P. Sistla, editors, *Proceedings of the 12th International Conference on Computer Aided Verification, CAV 2000*, volume 1855 of *Lecture Notes in Computer Science*, pages 53–68. Springer-Verlag, 2000.
11. D. Detlefs, G. Nelson, and J. B. Saxe. Simplify: A theorem prover for program checking. Technical Report HPL-2003-148, HP Labs, 2003.
12. S. Feferman. Lectures on proof theory. In M. H. Löb, editor, *Proceedings of the Summer School in Logic*, volume 70 of *Lecture Notes in Mathematics*, Leeds, 1968. Springer-Verlag.
13. S. Feferman. Applications of many-sorted interpolation theorems. In L. Henkin, editor, *Tarski Symposium: Proceedings of an International Symposium to Honor Alfred Tarski*, volume XXV of *Proceedings of Symposia in Pure Mathematics*, pages 205–223, Berkeley CA, 1974. American Mathematical Society.
14. J. Handy. *The Cache Memory Book*. Morgan Kaufmann, 1997.
15. D. Hilbert. *Grundlagen der Geometrie*. Teubner, 1899. English translation 'Foundations of Geometry' published in 1902 by Open Court, Chicago.
16. R. G. Jereslow. Computation-oriented reductions of predicate to propositional logic. *Decision Support Systems*, 4:183–197, 1988.
17. G. Kreisel and J.-L. Krivine. *Elements of mathematical logic: model theory*. Studies in Logic and the Foundations of Mathematics. North-Holland, revised second edition, 1971. First edition 1967. Translation of the French 'Eléments de logique mathématique, théorie des modeles' published by Dunod, Paris in 1964.
18. S. K. Lahiri, S. A., and R. E. Bryant. Modeling and verification of out-of-order microprocessors in UCLID. In M. Aagaard and J. O'Leary, editors, *Formal Methods in Computer-Aided Design: Fourth International Conference FMCAD 2002*, volume 2517 of *Lecture Notes in Computer Science*, pages 142–159. Springer-Verlag, 2002.
19. K. Meinke and J. V. Tucker, editors. *Many-sorted Logic and its Applications*. John Wiley and Sons, 1993.
20. R. Milner. A theory of type polymorphism in programming. *Journal of Computer and Systems Sciences*, 17:348–375, 1978.
21. G. Nelson and D. C. Oppen. Simplification by cooperating decision procedures. *ACM Transactions on Programming Languages and Systems*, 1:245–257, 1979.
22. S. Pandav, K. Slind, and G. Gopalakrishnan. Counterexample guided invariant discovery for parameterized cache coherence verification. In D. Borrione and W. J. Paul, editors, *Correct Hardware Design and Verification Methods, 13th IFIP WG 10.5 Advanced Research Working Conference, CHARME 2005*, volume 3725 of *Lecture Notes in Computer Science*, pages 317–331, Saarbrücken, Germany, 2005. Springer-Verlag.
23. D. A. Plaisted. Ordered semantic hyper-linking. *Journal of Automated Reasoning*, 25:167–217, 2000.
24. A. Pnueli, S. Ruah, and L. Zuck. Automatic deductive verification with invisible invariants. In T. Margaria and W. Yi, editors, *Proceedings of TACAS01: Tools and Algorithms for the Construction and Analysis of Systems*, volume 2031 of *Lecture Notes in Computer Science*. Springer-Verlag, 2001.
25. F. P. Ramsey. On a problem of formal logic. *Proceedings of the London Mathematical Society (2)*, 30:361–376, 1930.
26. H. Reuß and N. Shankar. Deconstructing Shostak. In *Proceedings of the Sixteenth Annual IEEE Symposium on Logic in Computer Science* [1], pages 19–28.
27. R. Shostak. Deciding combinations of theories. *Journal of the ACM*, 31:1–12, 1984.

28. A. Stump, D. L. Dill, C. W. Barrett, and J. Levitt. A decision procedure for an extensional theory of arrays. In *Proceedings of the Sixteenth Annual IEEE Symposium on Logic in Computer Science* [1], pages 29–37.
29. C. B. Suttner and G. Sutcliffe. The TPTP problem library. Technical Report AR-95-03, Institut für Infomatik, TU München, Germany, 1995. Also available as TR 95/6 from Dept. Computer Science, James Cook University, Australia, and on the Web.
30. A. Tarski and S. Givant. Tarski's system of geometry. *Bulletin of Symbolic Logic*, 5:175–214, 1999.
31. C. Walther. A mechanical solution of Schubert's steamroller by many-sorted resolution. *Artificial Intelligence*, 26:217–224, 1985.
32. C. Walther. Many-sorted unification. *Journal of the ACM*, 35:1–17, 1988.