Proving invariants using many-sorted logic

John Harrison

Intel Corporation

WG 2.3 meeting, Bruges

March 16, 2006

Plan for the talk

Central problem in verification: reachability or non-reachability w.r.t. a transition relation.

Can be done by arriving at an inductive invariant (using ingenuity).

We would, like to *verify* inductiveness automatically (no ingenuity).

If the invariant and/or transition relation includes quantifiers (‘for all x . . . ’) even this is not in general a decidable problem.

By exploiting types, some apparently problematic cases are in fact seen to be decidable.

Transition relations, reachable states

We'll use sets and predicates interchangeably, so $\sigma \in S$ and $S(\sigma)$ mean the same thing.

We have a set S of "states" and a binary "transition relation" $R \subseteq S \times S$.

Fix a set $S_0 \subseteq S$ of "initial states". A state is reachable if we can get to it from a state in S_0 by any number of R-steps:

$$S^*(\sigma') = \exists \sigma.\, S_0(\sigma) \wedge R^*(\sigma, \sigma')$$

We write S^* for the set of reachable states.

Invariant and inductive invariant

A property P is an *invariant* if it holds in all reachable states, i.e. $S^* \subseteq P$. The set of reachable states is the smallest (strongest) invariant.

A property P is *inductive* if it holds in all initial states ($S_0 \subseteq P$) and is preserved by transitions ($\forall \sigma\ \sigma'.\ P(\sigma) \wedge R(\sigma, \sigma') \Rightarrow P(\sigma')$).

Every inductive property is also an invariant, so we call it an “inductive invariant”.

Not every invariant is inductive. But the set of reachable states certainly is.

How to solve it

We can imagine that S_0, R and P are all represented by logical formulas.

They may be specified directly that way (e.g. TLA), or we may derive them by embedding some language like Murphi or SMV.

We want to show that P is an invariant. How might we do it?

- Enumeration — if the state space is finite and small enough, can model-check.
- Abstraction — cleverly replace the transition system by a simpler one and infer results from that.
- Induction — we'll concntrate on this.

Finding an inductive invariant

If we're very lucky, the property P may be inductive. In real problems it usually isn't.

The solution is familiar from 'strengthening the inductive hypothesis'. We seek an I such that

- I is an inductive invariant
- $\forall \sigma.\ I(\sigma) \Rightarrow P(\sigma)$

This may be quite difficult and need considerable skill and ingenuity.

From the verification coalface

Three bugs are crawling on the coordinate plane. They move one at a time, and each bug will only crawl in a direction parallel to the line joining the other two.

The bugs start out at (0,0), (3,0), and (0,3).

(a) Is it possible that after some time the first bug will end up back where it started, while the other two bugs switch places?

(b) Can the bugs end up at (1,2), (2,5), and (-2,3)?

Finding an inductive invariant

The answer is *no* in both cases. How could we prove it?

The property of not being at a particular point is certainly *not* inductive.

So we need to find some inductive invariant that implies the property we want.

Solution to the bug puzzle

The oriented area of the triangle formed by the bugs does not change.

If the bugs are at (a_x, a_y), (b_x, b_y) and (c_x, c_y), the oriented area is:

$$((b_x - a_x) \cdot (c_y - a_y) - (c_x - a_x) \cdot (b_y - a_y))/2$$

In the initial configuration this is $9/2$, and in the two final configurations it's $-9/2$ and ± 5.

So all we need to prove is that ‘the oriented area is $9/2$’ is inductive, and we're done.

Proof obligation

Final proof obligation is just algebra:

$$
\begin{aligned}
&(\exists a.\, a'_x = a_x + a \cdot (c_x - b_x) \wedge a'_y = a_y + a \cdot (c_y - b_y) \wedge \\
&\qquad b'_x = b_x \wedge b'_y = b_y \wedge c'_x = c_x \wedge c'_y = c_y) \vee \\
&(\exists b.\, b'_x = b_x + b \cdot (a_x - c_x) \wedge b'_y = b_y + b \cdot (a_y - c_y) \wedge \\
&\qquad a'_x = a_x \wedge a'_y = a_y \wedge c'_x = c_x \wedge c'_y = c_y) \vee \\
&(\exists c.\, a'_x = a_x \wedge a'_y = a_y \wedge b'_x = b_x \wedge b'_y = b_y \wedge \\
&\qquad c'_x = c_x + c \cdot (b_x - a_x) \wedge c'_y = c_y + c \cdot (b_y - a_y)) \wedge \\
&((b_x - a_x) \cdot (c_y - a_y) - (c_x - a_x) \cdot (b_y - a_y))/2 = 9/2 \\
&\Rightarrow ((b'_x - a'_x) \cdot (c'_y - a'_y) - (c'_x - a'_x) \cdot (b'_y - a'_y))/2 = 9/2
\end{aligned}
$$

After prenex normal form, purely universal. Easily solved automatically; HOL Light's `REAL_RING` rule proves it in $0.57s$.

When is inductiveness hard?

This was a characteristic inductive correctness proof:

- Arriving at the inductive invariant required human insight
- The proof *of* inductiveness was then automatic.

We will not consider the problem of automating inductive invariant generation. Instead we ask:

- What if the inductiveness proof itself is hard? In particular, what if the transition relation and/or invariant contains quantifiers?

Parametrized systems

An interesting target for verification is *parametrized systems*.

Such a system typically contains some number N of equivalent replicated components, so the state space involves some Cartesian product

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

and the transition relation is symmetric between the replicated components.

Sometimes we have subtler symmetry, but we'll just consider full symmetry.

Parametrized verification

Even if Σ_1 is finite, we can only use straightforward model checking when N is a specific number.

In practice, only small N may be feasible.

Yet the system is often expected/supposed to work for *arbitrary* N.

So we would like a proof that is general, with N treated as an arbitrary parameter.

Multiprocessors with private cache

A classic example of a parametrized system is a multiprocessor where each processor has its own cache.

We have N cacheing agents with state space Σ_1 each, and maybe some special ‘home node’ with state space Σ_0.

We can consider Σ_1 as finite with two radical but not unreasonable simplifications:

- Assume all cache lines are independent (no resource allocation conflicts)
- Ignore actual data and consider only state of cache line (dirty, clean, whatever)

Coherence

The permitted transitions are constrained by a protocol designed to ensure that all caches have a coherent view of memory.

On some simplifying assumptions, we can express this adequately just using the cache states.

In classic MESI protocols, each cache can be in four states: `Modified`, `Exclusive`, `Shared` and `Invalid`.

Coherence means:

$$\forall \texttt{i}.\ \texttt{Cache(i) IN } \{\texttt{Modified, Exclusive}\} \\ \Rightarrow \forall \texttt{j}.\ \neg(\texttt{j = i}) \Rightarrow \texttt{Cache(j) = Invalid}$$

Form of inductiveness claim

For a very simple and abstract protocol description, coherence may already be inductive. Look at logical form of inductiveness claim $I(\sigma) \wedge R(\sigma, \sigma') \Rightarrow I(\sigma')$.

The inductive invariant I is universally quantified, and occurs in both antecedent and consequent.

The transition relation has outer existential quantifiers $\exists i.\ \cdots$ because we have a symmetric choice between all components.

Inside, we may also have universal quantifiers if we choose to express array updates `a(i) := Something` as relations between functions:

$$a'(i) = \texttt{Something} \wedge \forall j.\ \neg(j = i) \Rightarrow a'(j) = a(j)$$

Our quantifier prefix

So our inductiveness claim may look like

$$(\forall i, j, \ldots.\ \cdots) \land (\exists i.\ \forall j.\ \cdots) \Rightarrow (\forall i, j, \ldots.\ \cdots)$$

If we put this into prenex normal form in the right way, the quantifier prefix is of the form:

$$\forall \cdots \forall \exists \cdots \exists$$

Suppose we don't need any arithmetic.

We can add assumptions for exhaustiveness and exclusiveness of the 4-element type of cache states without disturbing the logical form.

Is this problem decidable?

The AE fragment

A classic decidability result for first order logic due to Bernays, Schönfinkel and Ramsey.

A first-order formula is in AE form if it contains no function symbols and has, or can obviously be transformed into, the following prenex form:

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

with $P[x_1, \ldots, x_n, y_1, \ldots, y_m]$ quantifier-free. Dually, EA form is

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

Logical validity for AE formulas / satisfiability for EA formulas is decidable.

Finite model proof

An AE formula $\exists x_1, \ldots, x_n.\ \forall y_1, \ldots, y_m.\ P[x_1, \ldots, x_n, y_1, \ldots, y_m]$ has a model iff it has a model with domain size n.

One way is trivial. And if it has a model with domain D, there are $a_1, \ldots, a_n \in D$ such that for any $b_1, \ldots, b_m \in D$ we have $P_M[a_1, \ldots, a_n, b_1, \ldots, b_m]$.

Let $D' = \{a_1, \ldots, a_n\}$. Then *a fortiori* for any $b_1, \ldots, b_m \in D$ we have $P_M[a_1, \ldots, a_n, b_1, \ldots, b_m]$. So the formula holds in D'.

Note that this fails if the formula involves function symbols: we don't know that D' is closed under the action of their interpretation.

Skolem-Gödel-Herbrand proof

By Skolemization, the formula is satisfiable iff this is:

$$\forall y_1, \ldots, y_m.\ P[c_1, \ldots, c_n, y_1, \ldots, y_m]$$

By the Skolem-Gödel-Herbrand theorem this is unsatisfiable iff the set of all ground instances

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

with t_i ranging over all ground terms.

But the only ground terms are the constants c_i, so this is a finite conjunction, and we can decide it propositionally.

Again, this fails if we have a function symbol, because then we need to consider the infinite set of instantiations to c, $f(c)$, $f(f(c))$, . . .

Not quite what we need

Our inductive invariance claim does have an AE quantifier prefix.

And it doesn't need any background theory like arithmetic.

Unfortunately it *does* include functions! We have the function `Cache` representing the array of caches . . .

Many-sorted logic

Traditional first-order logic only has one kind of object, and models have a single domain.

However it's sometimes more natural to consider many-sorted logic where type distinctions are made and there are separate domains for each type.

For example in formalizing geometry we may have separate sorts P for points and L for lines:

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

Why are types ignored?

Very little of the literature on automated reasoning considers many-sorted logic.

Most first-order proof procedures do not expand over the set of ground terms, but identify instantiations by unifications.

Well-typed problems lead to well-typed unifiers, even if the types are ignored during proof search.

However, when we *do* consider expansion over the ground terms, things are very different . . .

Many-sorted Skolem-Gödel-Herbrand

In many sorted-logic, the obvious analog of the Skolem-Gödel-Herbrand theorem holds.

However, the construction of ground terms is constrained by type: we only consider well-typed combinations.

In particular, since $\texttt{Cache}$ has type $\texttt{Node} \rightarrow \texttt{State}$, terms like $\texttt{Cache}(\texttt{Cache}(i))$ are ill-typed.

So there is *still* only a finite set of ground terms!

Practical implications

Our inductiveness problem *is* decidable. The decision method: a relatively modest finite expansion then hit it with a free-variable SMT solver.

Works for relatively complex transition relations and invariants, provided their logical form is right.

We can even add theories such as arithmetic. Non-trivial, because it's generally thought that combining either quantifiers or uninterpreted functions with linear arithmetic leads to undecidability.

The main problem is that we cannot have arrays of node indices, since then we once again have an infinite set of ground terms. (So we can handle the German protocol, but not FLASH.)

Logical generality

The present observations are implicit in work by Pnueli et al. on “invisible invariants”.

However, it’s tied up there with a particular form of invariant synthesis. We want to present it in full logical generality.

There are other situations where we can get a decision method thanks to the more refined view sorts give us.

We’ll give one more such example.

Metric spaces

A metric is a binary function $d : S \times S \to \mathbb{R}$ such that:

$$\forall x\ y : S.\ d(x,y) \geq 0$$

$$\forall x\ y : S.\ d(x,y) = 0 \Leftrightarrow x = y$$

$$\forall x\ y : S.\ d(x,y) = d(y,x)$$

$$\forall x\ y\ z : S.\ d(x,z) \leq d(x,y) + d(y,z)$$

This is naturally formulated as a 2-sorted first-order theory with the background theory of arithmetic. Again, terms like $d(d(x,y), d(x,y))$ are ill-typed.

So we can decide whether an AE formula holds in all metric spaces based on a finite expansion then using a normal SMT solver.

Antimetric spaces

Works with any axioms with the same logical form and the same type structure, such as *antimetric spaces* where we change the axioms to:

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

Again, we can decide whether an AE formula holds in all antimetric spaces.

An expansion proof

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

The sorry truth

$$
\begin{aligned}
&(\forall x\ y.\ d(x,y) = 0 \Leftrightarrow x = y)\wedge \\
&(\forall x\ y.\ 0 \le d(x,y))\wedge \\
&(\forall x\ y\ z.\ d(x,y) + d(y,z) \le d(x,z)) \\
&\Rightarrow \forall x\ y.\ x = y
\end{aligned}
$$

Conclusions

In practical applications, we are increasingly using expansion-based techniques, rather than unification-based ones. (Look at UCLID!)

By maintaining type distinctions, we often get much smaller and more efficient expansions (Jeroslow).

The special case of invariants for parametrized system is interesting, and implicit in existing work (Pnueli . . .)

More generally, there are many benefits to considering sorts properly, yet most SMT suites do not.

`www.cl.cam.ac.uk/users/jrh/papers/manysorted.pdf`

`www.cl.cam.ac.uk/users/jrh/papers/holhol.pdf`