IC3PO Finite Convergence Check

Ian Dardik

February 3, 2022

1 Introduction

IC3PO proposed an automatic method for performing "finite convergence checks" [?]. In this document I will describe the problem, the proposed algorithm, as well as issues that I discovered. Throughout this document I will adopt the conventions as used in [?], but I have chosen different variable names for convenience and readability.

2 Preliminaries

Let P be either a property or a transition system. Then the *template* of P is $P(S_1, ..., S_n)$ where $n \in \mathbb{N}$ and each S_i is a sort. A finite instance of P is denoted as $P(|S_1|, ..., |S_n|)$, where each $|S_i| \in \mathbb{N}$ denotes the size of sort S_i. For example, if we have a property $Init$ parameterized by two sorts S_1 and S_2, then $Init(S_1, S_2)$ is the template of $Init$ while $Init(2, 3)$ is a finite instance.

3 Problem Statement

Given a transition system $T = (Init, Trans)$ with template $T(S_1, ..., S_n)$ and a property P with template $(S_1, ..., S_n)$, we want to determine whether $P(S_1, ..., S_n)$ is an inductive invariant for $T(S_1, ..., S_n)$. $P(S_1, ..., S_n)$ is said to be an inductive invariant for $T(S_1, ..., S_n)$ iff $P(|S_1|, ..., |S_n|)$ is an inductive invariant for $T(|S_1|, ..., |S_n|)$ for all $\{|S_1|, ..., |S_n|\} \in \mathbb{N}^n$.

4 Proposed Solution

Given the problem statement outlined in the previous section, [?] proposes the following algorithm: Manually identify a vector $V \in \mathbb{N}^n$ which assigns *basesizes* to each of the n sorts. V must be chosen such that $T(v_1, ..., v_n)$ exhibits non-trivial behavior; "non-trivial behavior" is glossed over in the IC3PO paper but is intuitively sort sizes large enough to see some complex behaviors in the protocol T. Then P is an inductive invariant if the following checks hold for $1 \leq i \leq n$:

1. $Init(v_1, .., v_i + 1, .., v_n) \rightarrow P(v_1, .., v_i + 1, .., v_n)$
2. $P(v_1, .., v_i + 1, .., v_n) \wedge Trans(v_1, .., v_i + 1, .., v_n) \rightarrow P'(v_1, .., v_i + 1, .., v_n)$

5 Counterexample

There is a simple counterexample to this proposed solution. We begin by defining the property R and then proceed to describe the counterexample.

5.1 The *R* Property

Suppose that $T = (Init, Trans)$ is a transition system parameterized by a single sort S, and has at least one reachable state outside of $Init$. Further suppose that Q is an inductive invariant for $T(S)$. Then we define $R(m) = Q \wedge (|S| = m \rightarrow Init)$. Because T has at least one reachable state outside of $Init$, it is clear that $Init$ is neither inductive nor invariant. It follows that there is no value of $m \in \mathbb{N}$ such that $R(m)$ is an inductive invariant for $T(S)$.

5.2 *R*(1) Counterexample

The proposed solution in section 4 will incorrectly decide that $R(1)$ is an inductive invariant. To see why, consider an arbitrary basesize vector $V = \{v\}$ where $v \in \mathbb{N}$. Then IC3PO will check the following for finite convergence:

1. $Init(v+1) \rightarrow (R(1))(v+1)$
2. $(R(1))(v+1) \wedge Trans(v+1) \rightarrow (R(1))'(v+1)$

Notice that if $v > 0$, then

$$\begin{aligned} & (R(1))(v+1) \\ \iff & Q \wedge (v+1 = 1 \rightarrow Init) \\ \iff & Q \wedge (false \rightarrow Init) \\ \iff & Q \end{aligned}$$

Thus, for any basesize larger than 0, the finite convergence check reduces to checking whether Q is an inductive invariant. Because Q is an inductive invariant, the algorithm will incorrectly decide that $R(1)$ is an inductive invariant.

5.3 *R*(1) Counterexample in IC3PO

The $R(1)$ counterexample is easily realized in IC3PO with the following Ivy program:

```
#lang ivy1.7

type num

relation val(X:num)

after init {
  val(X) := false;
}

action t(x:num) = {
  val(x) := true;
}

export t

invariant [Q] val(X) | ~val(X)
invariant [R(1)] (forall X:num,Y. X = Y) -> (forall Z. ~val(Z))
```

In this Ivy program there is a single sort num, and Q is separated from $R(1)$ for convenience. Notice that

$$\begin{aligned} & Q \\ \iff & \forall X : \mathrm{val(X)} \vee \neg\mathrm{val(X)} \\ \iff & true \end{aligned}$$

which is clearly an inductive invariant, and

$$\begin{aligned} & R(1) \\ \iff & (\forall X, Y : X = Y) \rightarrow (\forall Z, \neg\mathrm{val(Z)}) \\ \iff & (|num| = 1) \rightarrow Init \end{aligned}$$

Running IC3PO on this program with $|num| = 0$ yields a property violation as expected. Running with $|num| = 1$ yields a property violation too; this is expected because IC3PO finds the property violation while it tries to synthesize an invariant (not during the finite convergence check). For any values of $|num| > 1$ however, IC3PO incorrectly posits that the conjunction of Q and $R(1)$ is an inductive invariant. Since this example is in EPR, I was able to run ivy_check (which does not use finite interpretations of the sorts) to produce a counterexample and verify that $Q \wedge R(1)$ is not an inductive invariant.

5.4 R(m) Counterexamples

This counterexample generalizes to higher values of m, though I have not tested it on IC3PO yet (because I don't know how to test for cardinality of a sort in Ivy). Imagine the following Ivy program:

```
#lang ivy1.7

type num

relation val(X:num)

after init {
  val(X) := false;
}

action t(x:num) = {
  val(x) := true;
}

export t

invariant [Q] val(X) | ~val(X)
invariant [R(100)] (|num|=100) -> (forall Z. ~val(Z))
```

In this case, the conjunction of the invariants is still clearly not an inductive invariant. However, users of IC3PO will most likely choose basesizes of small integers such as 2 or 3; checking for convergence using the method in 4 will incorrectly decide that the conjunction of the invariants *is* an inductive invariant. In this case, the user would need to choose a basesize of 99 or larger for the method to work properly.

6 Discussion

Since m can be arbitrarily chosen to create $R(m)$, *no* finite number of checks on finite instances are sufficient to prove that a a property is an inductive invariant. However, if we restrict our checks to only properties of some special form, then a finite number of checks may be possible.

7 Conclusion

A counterexample to IC3PO's finite convergence detection algorithm is presented. It is possible that restricting the universe of allowed inductive invariants could make the method (or a similar one) work, and this would be a very interesting research direction.