IC3PO Finite Convergence Check

Ian Dardik

February 2, 2022

1 Introduction

IC3PO proposed an automatic method for performing "finite convergence checks" [1]. 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 [1].

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 $|S_i|$ denotes the size of sort S_i in the finite instance. For example, if we have a property $Init$ parameterized by two sorts, 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 $P(S_1, ..., S_n)$, we want to determine whether $P(S_1, ..., S_n)$ is an inductive invariant for $T(S_1, ..., S_n)$.

4 Proposed Solution

Given the problem statement outlined in section problem-statement, [1] proposes the following algorithm: Identify a vector V of sort *base sizes*, where $V = \{v_1, ..., v_n\}$. 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 check holds 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 with a single sort S and at least one reachable state outside of $Init$. Suppose that Q is an inductive invariant for $T(S)$, and let $R(m) = Q \wedge (|S| = m) \rightarrow Init$. Because T has at least one reachable state outside of $Init$, it is clear that $Init$ is neither inductive nor invariant, and hence $R(m)$ cannot be inductive invariant for all $m \in \{1, 2, ...\}$.

5.2 *R(1)* Counterexample

The proposed solution in section sol will incorrectly decide that $R(1)$ is an inductive invariant. To see why, consider an arbitrary base size vector $V = \{v\}$ where $v \in \{1, 2, ...\}$. 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 $(R(1))(v + 1) \iff Q \wedge (v + 1 = 1) \rightarrow Init \iff Q$, where the second biconditional follows because $v + 1 \neq 1$ for all values of v. Thus, both checks will pass because Q 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 [safety] val(X) | ~val(X)
invariant [bad] (forall X:num,Y. X = Y) -> (forall Z. ~val(Z))
```

In this Ivy program, Q (the safety property) is simply *true* which is an inductive invariant. $R(1)$ is realized as the invariant “bad”, which states that if there’s exactly one constant in the sort “num”, then all values of “val” are $false$ (which is identical to $Init$).

Running IC3PO on this program with $|S| = 0$ or $|S| = 1$ yields a property violation, while $|S| = 2$ or higher suggests that $safety$ and *bad* are together an inductive invariant. Running ivy_check (which does not use finite interpretations of the sorts) shows a counterexample on this program.

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 [safety] 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 base sizes of small integers such as 2 or 3; checking for convergence using the method in sol will incorrectly decide that the conjunction of the invariants *is* an inductive invariant. In this case, the user would need to choose a base size of 99 or larger for the method to work properly.

6 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 work, and this would be a very interesting research direction.