Project Abstract: Formally verifying Dining Philosophers as Feedback Control

Siddharth Bhat

May 4, 2020

1 Introduction

I formally verify the first three sections of the paper 'Generalized Dining Philosophers as Feedback Control'. Here, I describe the setting, definitions, theorem statement, and a sketch of the proof. I then move to the experience report, where I describe the interactions that occurred when attempting to formally verify this in Coq. Finally, I end with a discussion of the feedback given, as well as some thoughts on Tabuada composition as a substrate for system design.

2 Theorem Statement In Coq

2.1 System

We first begin by defining a system.

Definition 1. A **System** is a 4-tuple of (1) a ground set X of system **states**, (2) a set U of **actions**, (3) a subset $X_0 \subseteq X$ of **initial states**, (4) a relation $\rightarrow \subseteq X \times U \times X$ of **transitions**. Formally:

$$\text{System} \equiv (X : \textbf{Set}, U : \textbf{Set}, X_0 \subseteq X, \rightarrow \subseteq X \times U \times X).$$

To encode this definition within Coq, note that we need to encode sets and subsets. A set S ie enoded by a type `T_S`, whose values `v_s : T_S` corresponds to members $s \in S$. To encode subsets, we exploit the fact that a subset $X_0 \subseteq X$ is equivalent to a membership function $\texttt{member}(X_0) : X \rightarrow \texttt{Bool}$:

$$X_0 \subseteq X \iff \texttt{member}(X_0) : X \rightarrow \texttt{Bool};\ \texttt{member}(X_0)(x) \equiv \begin{cases} \texttt{true} & x \in X_0 \\ \texttt{false} & x \notin X_0 \end{cases}$$

So, we encode a system in Coq as:

```
Record system (X: Set) (U: Set) :=
    mksystem { isx0: X -> Prop; trans: X -> U -> X -> Prop }.
```

2.2 Traces

We now have the definition of a system. We need to define what a legal trace of the given system is. Intuitively, a trace is a sequence of states $xs : \mathbb{N} \to X$ and actions $us : \mathbb{N} \to U$, such that (1) the first state is an initial state: $xs(0) \in X_0$; (2) successive states are related by the transition relation:

$$\forall n \in \mathbb{N}, xs(n) \xrightarrow{us(n)} xs(n+1)$$

.

We encode this as an inductive proposition in coq called `ValidTrace s xs us n`, which witnesses the fact that (xs, us) forms a valid trace for the system s for n steps:

```
(* ValidTrace s xs us n: trace suggested by xs, us is valid for n steps *)
Inductive ValidTrace {X U: Set}
    (s: system X U) (xs: nat -> X) (us: nat -> U): nat -> Prop :=
| Start: forall  (VALID: (isx0 X U s) (xs 0)) , ValidTrace s xs us 0
| Cons: forall (n: nat)
    (TILLN: ValidTrace s xs us n) (* Trace valid till n *)
    (ATN: trans X U s (xs n) (us n) (xs (S n))), (* Transition at n *)
    ValidTrace s xs us (S n).
```

2.3 Tabuada Interconnect

Given two systems S, T, we define a new system, their **Tabuada Interconnect**, styled as $S \times_{\mathcal{I}} T$. This interconnect is governed by $\mathcal{I}$. Formally:

1 $S \equiv (X : \mathbf{Set}, U_X : \mathbf{Set}, X_0 \subseteq X, \underset{X}{\to} \subseteq X \times U_X \times X)$.

2 $T \equiv (Y : \mathbf{Set}, U_Y : \mathbf{Set}, Y_0 \subseteq Y, \underset{Y}{\to} \subseteq Y \times U_Y \times Y)$.

3 **Interconnect:** $\mathcal{I} \subseteq (X \times Y) \times (U_X \times U_Y)$

4 **Composition:**

$$S \times_{\mathcal{I}} T \equiv (Z \equiv X \times Y, U_Z \equiv U_X \times U_Y, X_0 \times Y_0, \underset{Z,\mathcal{I}}{\to} \subseteq Z \times U_Z \times Z).$$

$$(x, y) \xrightarrow[Z,\mathcal{I}]{u_x, u_y} (x', y') \iff x \xrightarrow{u_x} x' \wedge y \xrightarrow{u_y} y' \wedge (x, y, u_x, u_y) \in \mathcal{I}.$$

In Coq, we define a new system as `tabuada sx sy connect : system`.

```
(* 2.2: system composition *)
(* tabuada connection new system *)
Definition tabuada {X Y UX UY: Set}
  (sx: system X UX) (sy: system Y UY)
  (connect: X*Y->UX*UY->Prop): system (X*Y) (UX*UY) :=
  mksystem (X*Y) (UX*UY) (tabuada_start (isx0 X UX sx) (isx0 Y UY sy))
           (tabuada_trans connect (trans X UX sx) (trans Y UY sy)).
```

Where tabuada_start identifies the initial states of sx $\times_{\texttt{connect}}$ sy, and tabuada_trans identified the transition:

```
(* initial state for tabuada composition *)
Definition tabuada_start {X Y: Type}
  (isx0: X -> Prop) (isy0: Y -> Prop) (x: X * Y): Prop :=
  isx0 (fst  x) /\ isy0 (snd x).

(* transition fn for tabuada composition *)
Definition tabuada_trans {X Y: Type} {UX UY: Type}
    (connect: X*Y->UX*UY->Prop)
    (transx: X -> UX -> X -> Prop)
    (transy: Y -> UY -> Y -> Prop)
    (s: X*Y) (u: UX*UY) (s': X*Y): Prop :=
  transx (fst s) (fst u) (fst s') /\
  transy (snd s) (snd u) (snd s') /\
  (connect s u).

Lemma system38_starvation_free:
  forall (n: nat) (ss: nat -> the * cmd)
   (ts: nat -> cmd * maybe choice * the)
   (TRACE_SSSN: ValidTrace system38 ss ts (S(S(S n))))
   (BOTTOM_EVEN:
      forall (i: nat) (IEVEN: even i = true),
      snd (fst (ts i)) = nothing choice)
   (NOT_BOTTOM_ODD:
      forall (i: nat) (IODD: odd i = true),
      snd(fst (ts i)) <> nothing choice)
   (HUNGRY: fst (ss n) = h),
  exists (m: nat),  m > n /\  fst (ss m) = e.
Proof.
    (* 30 lines of proof,
       200 lines of supporting lemmas ommitted *)
Qed.

Lemma system_38_phil_not_hungry_then_next_philo_choice:
  forall (n: nat) (ss: nat -> the * cmd) (c: choice)
   (ts: nat -> cmd * maybe choice * the)
   (TRACE_SSSN: ValidTrace system38 ss ts (S(S(S n))))
   (NOTHUNGRY: fst (ss (S n)) <> h)
   (CHOICE: snd (fst (ts (S n))) = just choice c)
   (NEVEN: even n = true)
   (BOTTOM_EVEN:
     forall (i: nat) (IEVEN: even i = true),
     snd (fst (ts i)) = nothing choice),
    fst (ss (S(S(S n)))) = trans32fn (fst (ss (S n))) c.
Proof.
```

```
    (* 20 lines of proof,
       200 lines of supporting lemmas ommitted *)
Qed.
```

3 Proof Statistics

- 667 lines of coq code.
- All tables upto section 4 verified by computation.
- All theorems upto section 4 formally verified.

4 Thoughts on tabuada composition

4.1 The Definition

Traditionally, the composition $Z \equiv X \times_{\mathcal{I}} Y$ is defined as:

1 **Interconnect:** $\mathcal{I} \subseteq (X \times Y) \times (U_X \times V_Y)$

2 **Composition:**

$$S \times_{\mathcal{I}} T \equiv (Z \equiv X \times Y, U_Z \equiv U_X \times V_Y, X_0 \times Y_0, \underset{Z,\mathcal{I}}{\rightarrow} \subseteq Z \times U_Z \times Z).$$

$$(x, y) \xrightarrow[Z,\mathcal{I}]{u_x, u_y} (x', y') \iff x \xrightarrow{u_x} x' \wedge y \xrightarrow{u_y} y' \wedge (x, y, u_x, v_y) \in \mathcal{I}.$$

Note that the definition of $\mathcal{I}$ does not discriminate between the states $X \times Y$ and the actions $U_X \times U_Y$. However, if we consider legal traces of this system:

$$(x_0, y_0) \xrightarrow{(u_0,v_0)} (x_1, y_1) \xrightarrow{(u_1,v_1)} (x_2, y_2) \xrightarrow{(u_2,v_2)} \dots$$

We note that since we *start with the state* (x_0, y_0), we are forced to pick $(u_0, v_0) \in \{(u, v) : (x_0, y_0, u, v) \in \mathcal{I}\}$. Similarly, once we have determined a (x_1, y_1), we are forced to pick $(u_1, v_1) \in \{(u, v) : (x_1, y_1, u, v) \in \mathcal{I}\}$. So, there is an inherent *causality* in the way in which the tabuada composition is *applied*. Really, the type of $\mathcal{I}$ ought to be:

$$\mathcal{I} : X \times Y \to 2^{U_X \times V_Y} \quad \text{(actions determined by state)}$$
$$(x_n, y_n) \xrightarrow{(u_n,v_n)} (x_{n+1}, y_{n+1}) : (u_n, v_n) \in \mathcal{I}(x_n, y_n)$$

4.2 Control Using Tabuada Composition

There appears

5 Feedback: Starting With Tabuada?

The first bit of feedback was:

> Seems that the problem statement has been made unnecessarily complicated. The starting point could have been a coupled system of equations that /assumes/ alternation. That way, the entire low level Tabuada composition that invokes 'bottom' as a possible value could be skipped. One has to carefully choose the abstractions to coax Coq into accepting them.

As written in the abstract of the project proposal:

> We propose to formally verify the proof development as sketched in the paper on ArXiV: Generalised Dining Philosophers as Feedback Control, within the Coq proof assistant. We will implement the proof, as laid out in sections 1, 2, and 3 (till *The One Dining Philosopher Problem*) of the document above.

I followed the paper as it built the theory. My understanding was that the key contribution of the paper was the modular reasoning offered by means of composing systems using tabuada composition; a *coupled* system of equations is (by definition) **coupled** — ergo, monolithic. If we wished to begin from the coupled set of equations, I feel the problem statement ought to have been framed differently.

On the other hand, I could have setup a *new system* which was driven by a coupled set of equations, that I showed was equivalent to the original system as defined by tabuada composition. I did not do this, since the lemmas that I have proved (for driving the proof) are essentially equivalent to having this "new system". It is unclear to me what the upshot of defining this would be.Roving the equivalence between the original system and the new system is equivalent to writing lemmas that "drive" the equational reasoning.

6 Feedback: Encode Relations As Functions?

The second bit of feedback was:

> If relations are awkward to use in Coq, why is the transition relation simply not defined as a function of the type `State -> set State`? I was unable to discern a clear reason.

It's unclear this is a win. Specifically, `set` is not a *free theory* — equality of sets is not freely generated from its constructors. Compare:

```
Inductive  list (A: Type): Type :=
 nil : list A | cons : A -> list A -> list A.
```

```
Lemma list_eq_equational: forall
    (X: Type)
    (x x': X) (xs xs': list X)
    (EQ: cons X x xs = cons X x' xs'),
    x = x' /\ xs = xs'.
Proof.
  intros. inversion EQ. (* equational *)
  auto.
Qed.
```

On the other hand, this theorem is patently false:

```
Lemma set_eq_equational: forall
    (X: Type)
    (x x': X) (xs xs': set X)
    (EQ: add X x xs = union X x' xs'),
    x = x' /\ xs = xs'.
Proof. Abort.
```

For example, we know that:

$$\texttt{add}\ 1\ \{1\} = \texttt{add}\ 1\ \emptyset; \qquad \{1\} \neq \emptyset$$

Hence, we have no way to equationally reason about sets. We have to carry proofs, and this burden of proof-carrying feels equivalent to writing lemmas that perform computation. However, there maybe "hidden wins" to doing it this way that I don't see; only trying it out can actually tell.