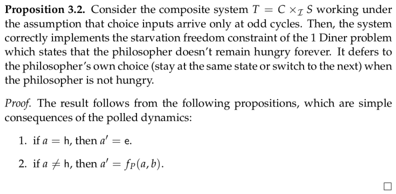
Sample title

Siddharth Bhat

IIIT Hyderabad

April 29, 2020

The story of a proof: from paper to kernel



Proposition 3.2. Consider the composite system $T = C \times_{\mathcal{I}} S$ working under the assumption that choice inputs arrive only at odd cycles. Then, the system correctly implements the starvation freedom constraint of the 1 Diner problem which states that the philosopher doesn't remain hungry forever. It defers to the philosopher's own choice (stay at the same state or switch to the next) when the philosopher is not hungry.

Proof. The result follows from the following propositions, which are simple consequences of the polled dynamics:

1. if $a = \mathsf{h}$, then $a' = \mathsf{e}$.
2. if $a \neq \mathsf{h}$, then $a' = f_P(a, b)$.

$\square$

The story of a proof: from paper to kernel

Proposition 3.2. Consider the composite system $T = C \times_{\mathcal{I}} S$ working under the assumption that choice inputs arrive only at odd cycles. Then, the system correctly implements the starvation freedom constraint of the 1 Diner problem which states that the philosopher doesn't remain hungry forever. It defers to the philosopher's own choice (stay at the same state or switch to the next) when the philosopher is not hungry.

Proof. The result follows from the following propositions, which are simple consequences of the polled dynamics:

1. if $a = \mathsf{h}$, then $a' = \mathsf{e}$.
2. if $a \neq \mathsf{h}$, then $a' = f_P(a, b)$.

□

```
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.
```

Statistics

- 667 lines of coq code.

Statistics

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

The definitions: Systems

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

The definitions: Systems

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

Recall: $X_0 \subseteq X \iff \texttt{member}(X_0) : X \rightarrow \texttt{Bool};\ \texttt{member}(X_0)(x) \equiv x \underset{?}{\in} X_0$

The definitions: Systems

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

Recall: $X_0 \subseteq X \iff \texttt{member}(X_0) : X \rightarrow \texttt{Bool};\ \texttt{member}(X_0)(x) \equiv x \underset{?}{\in} X_0$

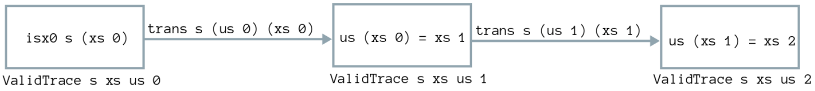
```
(* 2.1: system specifcation *)
Record system (X: Set) (U: Set) :=
    mksystem { isx0: X -> Prop;
               trans: X -> U -> X -> Prop }.
```

Running a System: Valid Traces

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

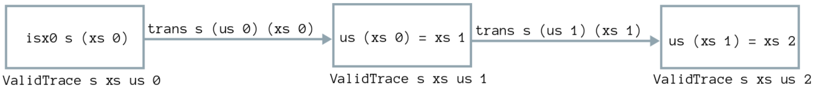
Running a System: Valid Traces

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



Running a System: Valid Traces

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



```
(* 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) (ATN: trans X U s (xs n) (us n) (xs (S n))),
ValidTrace s xs us (S n).
```

Composing Systems: Tabuada Connection

The sets:

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

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

Composing Systems: Tabuada Connection

The sets:

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

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

The interconnect:

$$\mathcal{I} \subseteq (X \times Y) \times (U_X \times U_Y)$$

Composing Systems: Tabuada Connection

The sets:

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

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

The interconnect:

$$\mathcal{I} \subseteq (X \times Y) \times (U_X \times U_Y)$$

The composition:

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

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

Composing Systems: Tabuada Connection

The sets:

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

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

The interconnect:

$$\mathcal{I} \subseteq (X \times Y) \times (U_X \times U_Y)$$

The composition:

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

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

```
(* 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)).

(* 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).
```

One diner system

One diner system

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

One diner system

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

Definition system38 : system := tabuada phil37 controller34 connect38.
```

One diner system

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

Definition system38 : system := tabuada phil37 controller34 connect38.

Definition phil37 := mksystem the (cmd * maybe choice) isthinking  (fun s u s' => trans37fn s u = s').
Definition controller34 := mksystem cmd the ispass (fun s u s' => trans34fn s u = s').
```

One diner system

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

Definition system38 : system := tabuada phil37 controller34 connect38.

Definition phil37 := mksystem the (cmd * maybe choice) isthinking  (fun s u s' => trans37fn s u = s').
Definition controller34 := mksystem cmd the ispass (fun s u s' => trans34fn s u = s').

(* Philosopher spec *)
Definition isthinking (s: the): Prop := s = t.

Inductive cmd := cmd_pass | cmd_bang0 | cmd_bang1.

Definition trans37fnDEPR (s: the) (u: cmd * maybe choice): the :=
  match u with
  | (_, nothing _) => s (* this looks fishy! I don't trust this :(. Indeed, moving this down
                           to be below cmd_bang0, cmd_bang1 breaks things *)
  | (cmd_bang0, _) => s | (cmd_bang1, _) => next s | (cmd_pass, just _ ch) => trans32fn s ch
  end.
```

One diner system

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

Definition system38 : system := tabuada phil37 controller34 connect38.

Definition phil37 := mksystem the (cmd * maybe choice) isthinking  (fun s u s' => trans37fn s u = s').
Definition controller34 := mksystem cmd the ispass (fun s u s' => trans34fn s u = s').

(* Philosopher spec *)
Definition isthinking (s: the): Prop := s = t.

Inductive cmd := cmd_pass | cmd_bang0 | cmd_bang1.

Definition trans37fnDEPR (s: the) (u: cmd * maybe choice): the :=
  match u with
  | (_, nothing _) => s (* this looks fishy! I don't trust this :(. Indeed, moving this down
                          to be below cmd_bang0, cmd_bang1 breaks things *)
  | (cmd_bang0, _) => s | (cmd_bang1, _) => next s | (cmd_pass, just _ ch) => trans32fn s ch
  end.

Inductive choice := choice_0 | choice_1.
Definition trans32fn (s: the) (c: choice): the :=
  match c with
  | choice_0 => s | choice_1 => next s
  end.
```

One diner system

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

Definition system38 : system := tabuada phil37 controller34 connect38.

Definition phil37 := mksystem the (cmd * maybe choice) isthinking  (fun s u s' => trans37fn s u = s').
Definition controller34 := mksystem cmd the ispass (fun s u s' => trans34fn s u = s').

(* Philosopher spec *)
Definition isthinking (s: the): Prop := s = t.

Inductive cmd := cmd_pass | cmd_bang0 | cmd_bang1.

Definition trans37fnDEPR (s: the) (u: cmd * maybe choice): the :=
  match u with
  | (_, nothing _) => s (* this looks fishy! I don't trust this :(. Indeed, moving this down
                           to be below cmd_bang0, cmd_bang1 breaks things *)
  | (cmd_bang0, _) => s | (cmd_bang1, _) => next s | (cmd_pass, just _ ch) => trans32fn s ch
  end.

Inductive choice := choice_0 | choice_1.
Definition trans32fn (s: the) (c: choice): the :=
  match c with
  | choice_0 => s | choice_1 => next s
  end.

(* Controller spec *)
Definition ispass (c: cmd):Prop := c = cmd_pass.
Definition trans34fn (s: cmd) (u: the): cmd :=
  match u with
  | h => cmd_bang1 | e => cmd_pass | t => cmd_pass
  end.
```

One diner system

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

Definition system38 : system := tabuada phil37 controller34 connect38.

Definition phil37 := mksystem the (cmd * maybe choice) isthinking  (fun s u s' => trans37fn s u = s').
Definition controller34 := mksystem cmd the ispass (fun s u s' => trans34fn s u = s').

(* Philosopher spec *)
Definition isthinking (s: the): Prop := s = t.

Inductive cmd := cmd_pass | cmd_bang0 | cmd_bang1.

Definition trans37fnDEPR (s: the) (u: cmd * maybe choice): the :=
  match u with
  | (_, nothing _) => s (* this looks fishy! I don't trust this :(. Indeed, moving this down
                          to be below cmd_bang0, cmd_bang1 breaks things *)
  | (cmd_bang0, _) => s | (cmd_bang1, _) => next s | (cmd_pass, just _ ch) => trans32fn s ch
  end.

Inductive choice := choice_0 | choice_1.
Definition trans32fn (s: the) (c: choice): the :=
  match c with
  | choice_0 => s | choice_1 => next s
  end.

(* Controller spec *)
Definition ispass (c: cmd):Prop := c = cmd_pass.
Definition trans34fn (s: cmd) (u: the): cmd :=
  match u with
  | h => cmd_bang1 | e => cmd_pass | t => cmd_pass
  end.

(* Connection Spec *)
Definition connect38 (xy: the * cmd)
                     (ux_uy: cmd * (maybe choice) * the): Prop :=
  (fst xy) = (snd ux_uy) /\ (snd xy = fst (fst ux_uy)).
```

Pain point: Non determinism

- Coq has computational ability.
- Functions are computational; Relations are not.
- Lose a lot of proof automation.

Pain point: Non determinism

- Coq has computational ability.
- Functions are computational; Relations are not.
- Lose a lot of proof automation.

```
Inductive X: Set := X1 | X2 | X3.  Inductive Y: Set := Y1 | Y2 | Y3.
```

Pain point: Non determinism

- Coq has computational ability.
- Functions are computational; Relations are not.
- Lose a lot of proof automation.

```
Inductive X: Set := X1 | X2 | X3.  Inductive Y: Set := Y1 | Y2 | Y3.

Definition x2y_fn (x: X): Y :=
  match x with | X1 => Y1 | X2 => Y2 | X3 => Y3 end.
(* Works; computational *)
Lemma x1_to_y1_fn: x2y_fn X1 = Y1. Proof. reflexivity. Qed.
```

Pain point: Non determinism

- Coq has computational ability.
- Functions are computational; Relations are not.
- Lose a lot of proof automation.

```
Inductive X: Set := X1 | X2 | X3.  Inductive Y: Set := Y1 | Y2 | Y3.

Definition x2y_fn (x: X): Y :=
  match x with | X1 => Y1 | X2 => Y2 | X3 => Y3 end.
(* Works; computational *)
Lemma x1_to_y1_fn: x2y_fn X1 = Y1. Proof. reflexivity. Qed.

Definition x2y_rel (x: X) (y: Y): Prop :=
  (x = X1 /\ y = Y1) \/ (x = X2 /\ y = Y2) \/  (x = X3 /\ y = Y3).
Lemma x1_to_y2_rel: x2y_rel X1 Y1.
Proof.
  try reflexivity. (* Does not work! Not computational *)
  unfold x2y_rel. left. auto.
Qed.
```

Pain point: Modularity

```
Lemma system38_s_the_to_t_the:
  forall (n: nat) (ss: nat -> the * cmd) (ts: nat -> cmd * maybe choice * the)
         (TRACE: ValidTrace system38 ss ts (S n)), snd (ts n) = fst (ss n).
Proof.
  intros.
  inversion TRACE as [TRACE1 | npred TRACE1 AT1]. subst. (* 1 *)
  inversion AT1 as [AT11 [AT12 AT13]]. (* 2 *)
  set (s1 := ss 1) in *.
  destruct s1 as [s1_the s1_cmd].
  set (t1 := ts 1) in *.
  destruct t1 as [[t1_cmd t1_mchoice] t1_the]. simpl in *.
  inversion AT13; simpl in *. (* 3;  useful info *)
  auto.
Qed.
```

Pain point: Modularity

```
Lemma system38_s_the_to_t_the:
  forall (n: nat) (ss: nat -> the * cmd) (ts: nat -> cmd * maybe choice * the)
         (TRACE: ValidTrace system38 ss ts (S n)), snd (ts n) = fst (ss n).
Proof.
  intros.
  inversion TRACE as [TRACE1 | npred TRACE1 AT1]. subst. (* 1 *)
  inversion AT1 as [AT11 [AT12 AT13]]. (* 2 *)
  set (s1 := ss 1) in *.
  destruct s1 as [s1_the s1_cmd].
  set (t1 := ts 1) in *.
  destruct t1 as [[t1_cmd t1_mchoice] t1_the]. simpl in *.
  inversion AT13; simpl in *. (* 3;  useful info *)
  auto.
Qed.
```

```
n : nat
ss : nat -> the * cmd
ts : nat -> cmd * maybe choice * the
TRACE : ValidTrace system38 ss ts (S n)
TRACE1 : ValidTrace system38 ss ts n
AT1 : tabuada_trans connect38
        (fun (s : the) (u : cmd * maybe choice) (s' : the) => trans37fn s u = s')
        (fun (s : cmd) (u : the) (s' : cmd) => trans34fn s u = s') (ss n)
        (ts n) (ss (S n))
AT11 : trans37fn (fst (ss n)) (fst (ts n)) = fst (ss (S n))
AT12 : trans34fn (snd (ss n)) (snd (ts n)) = snd (ss (S n))
AT13 : connect38 (ss n) (ts n)
s1_the : the
s1_cmd, t1_cmd : cmd
t1_mchoice : maybe choice
t1_the : the
H : fst (ss n) = snd (ts n)
H0 : snd (ss n) = fst (fst (ts n))
============================
snd (ts n) = fst (ss n)
```

Lemmas to drive reasoning

```
(* Helper: reason about even/odd *)
Lemma even_n_odd_Sn: forall (n: nat),
    (even n = true) <-> (odd (S n) = true).
(* Helper: reason about even/odd *)
Lemma odd_n_even_Sn: forall (n: nat),
    (odd n = true) <-> (even (S n) = true).
(* Rewrite ts in terms of ss *)
Lemma s_cmd_to_t_cmd:
  forall (n: nat) (ss: nat -> the * cmd)
    (ts: nat -> cmd * maybe choice * the)
    (TRACE: ValidTrace system38 ss ts (S n)),
    fst (fst (ts n)) = snd(ss n).
(* Rewrite ss in terms of ts *)
Lemma s_the_to_t_the:
  forall (n: nat) (ss: nat -> the * cmd)
    (ts: nat -> cmd * maybe choice * the)
    (TRACE: ValidTrace system38 ss ts (S n)),
    snd (ts n) = fst (ss n).
(* ‘h‘ leads to ‘!1‘ command next. *)
Lemma phil_hungry_then_next_controller_bang1:
  forall (n: nat) (ss: nat -> the * cmd)
    (ts: nat -> cmd * maybe choice * the)
    (TRACE_SN: ValidTrace system38 ss ts (S n))
    (HUNGRY: fst (ss n) = h),
    snd (ss (S n)) = cmd_bang1.
```

```
(* not ‘h‘ in leads to pass next *)
Lemma phil_not_hungry_then_next_controller_pass:
  forall (n: nat) (ss: nat -> the * cmd)
    (ts: nat -> cmd * maybe choice * the)
    (TRACE_SN: ValidTrace system38 ss ts (S n))
    (NOTHUNGRY: fst (ss n) <> h),
    snd (ss (S n)) = cmd_pass.
(* Phil. state in the next-odd-state = cur-state *)
Lemma phil_even_state_next_state:
  forall (n: nat) (ss: nat -> the * cmd)
    (ts: nat -> cmd * maybe choice * the)
    (TRACE_SN: ValidTrace system38 ss ts ((S n)))
    (BOTTOM_EVEN: ...)
    (NEVEN: even n = true),
    fst (ss (S n)) = fst (ss n).
(* Phil. state in the cur-odd-state = prev-state *)
Lemma phil_odd_state_prev_state:
  forall (n: nat) (ss: nat -> the * cmd)
    (ts: nat -> cmd * maybe choice * the)
    (TRACE_SN: ValidTrace system38 ss ts ((S n)))
    (BOTTOM_EVEN: ...)
    (SNODD: odd (S n) = true),
    fst (ss (S n)) = fst (ss n).
(* state after odd cycle has taken transition *)
Lemma odd_state_next_phil_state:
  forall (n: nat) (ss: nat -> the * cmd)
    (ts: nat -> cmd * maybe choice * the)
    (TRACE_SN: ValidTrace system38 ss ts ((S n)))
    (BOTTOM_EVEN: ...)
    (SNODD: odd n = true),
    fst (ss (S n)) = trans37fn  (fst (ss n)) (fst (ts n)).
```

Thank you

All code available at:

https://github.com/bollu/IIIT-H-code/tree/master/softwarefoundations/project

Phrasing of the Tabuada connection

Phrased as:

$$\mathcal{I} \subseteq S_X \times S_Y \times U_X \times U_Y$$

Phrasing of the Tabuada connection

Phrased as:

$$\mathcal{I} \subseteq S_X \times S_Y \times U_X \times U_Y$$

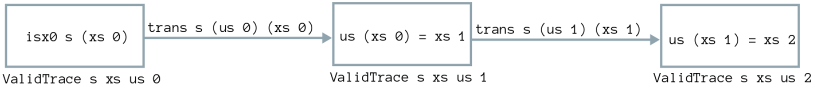
This definition has no sense of the *causality* inherent to the notion of `ValidTrace`.

Phrasing of the Tabuada connection

Phrased as:

$$\mathcal{I} \subseteq S_X \times S_Y \times U_X \times U_Y$$

This definition has no sense of the *causality* inherent to the notion of `ValidTrace`.

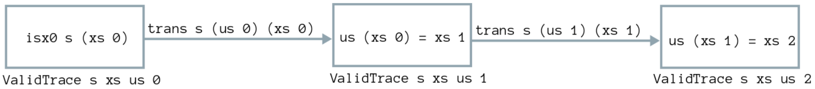


Phrasing of the Tabuada connection

Phrased as:

$$\mathcal{I} \subseteq S_X \times S_Y \times U_X \times U_Y$$

This definition has no sense of the *causality* inherent to the notion of `ValidTrace`.



Better definition:

$$\mathcal{I} : S_X \times S_Y \to 2^{U_X \times U_Y}$$