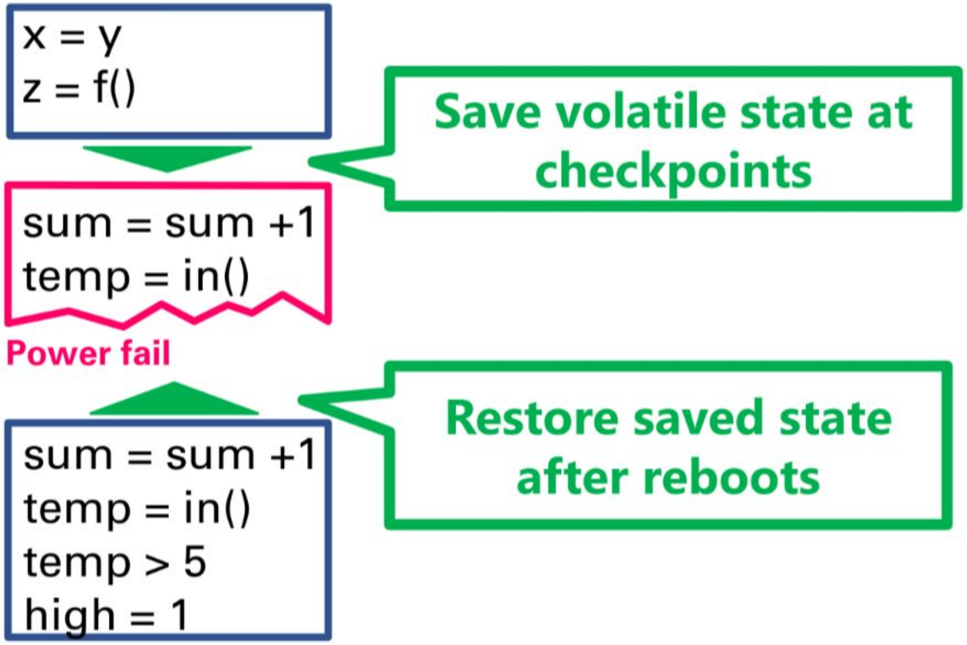
Indistinguishability Reasoning for Interaction Trees or, How to Tolerate a Lack of Productivity

Justine Frank

Crash recovery systems



Crash recovery systems

- Intermittent power harvesting systems
- Transactional file systems
- Distributed systems with persistent memory

Crashing and recovering should be *invisible to outside observers*

Atomic regions

```
atomic {
  if !x < !y
  then max := !y
  else max := !x
}
```

```
atomic {
  x := !x + 1
}
```

```
atomic (x) {
  y := !x;
  x := !y + 1
}
```

Interaction trees for crash recovery systems

- Embedding of impure non-terminating computations in proof assistants
- Has a rich theory mechanized in Rocq
- Crashing can be modeled as an effect that is nondeterministically triggered
- Crash recovery is then a handler for crash effects

```
Variant crashE : Type → Type :=
| Crash : crashE unit.
```

Interaction trees*

```
CoInductive itree (E : Type → Type) (A : Type) :=
| Ret (r : A) : itree E A
| Tau (t : itree E A) : itree E A
| Vis B (e : E B) (k : B → itree E A) : itree E A.
```

Interaction trees*

```
CoInductive itree (E : Type → Type) (A : Type) :=
| Ret (r : A) : itree E A
| Tau (t : itree E A) : itree E A
| Vis B (e : E B) (k : B → itree E A) : itree E A.
```

Interaction trees*

```
CoInductive itree (E : Type → Type) (A : Type) :=
| Ret (r : A) : itree E A
| Tau (t : itree E A) : itree E A
| Vis B (e : E B) (k : B → itree E A) : itree E A.
```

Interaction trees*

```
CoInductive itree (E : Type → Type) (A : Type) :=
| Ret (r : A) : itree E A
| Tau (t : itree E A) : itree E A
| Vis B (e : E B) (k : B → itree E A) : itree E A.
```

Interaction trees*

```
CoInductive itree (E : Type → Type) (A : Type) :=
| Ret (r : A) : itree E A
| Tau (t : itree E A) : itree E A
| Vis B (e : E B) (k : B → itree E A) : itree E A.
```

Interaction trees*

```
CoInductive itree (E : Type → Type) (A : Type) :=
| Ret (r : A) : itree E A
| Tau (t : itree E A) : itree E A
| Vis B (e : E B) (k : B → itree E A) : itree E A.
```

Interaction trees*

```
CoInductive itree (E : Type → Type) (A : Type) :=
| Ret (r : A) : itree E A
| Tau (t : itree E A) : itree E A
| Vis B (e : E B) (k : B → itree E A) : itree E A.
```

Interaction trees*

```
CoInductive itree (E : Type → Type) (A : Type) :=
| Ret (r : A) : itree E A
| Tau (t : itree E A) : itree E A
| Vis B (e : E B) (k : B → itree E A) : itree E A.
```

Interaction trees*

```
CoInductive itree (E : Type → Type) (A : Type) :=
| Ret (r : A) : itree E A
| Tau (t : itree E A) : itree E A
| Vis B (e : E B) (k : B → itree E A) : itree E A.
```

Interaction trees*

```
CoInductive itree (E : Type → Type) (A : Type) :=
| Ret (r : A) : itree E A
| Tau (t : itree E A) : itree E A
| Vis B (e : E B) (k : B → itree E A) : itree E A.
```

```
Variant mapE : Type → Type :=
| Get : string → mapE (option nat)
| Set : string → nat → mapE unit.
```

Interaction trees*

```
CoInductive itree (E : Type → Type) (A : Type) :=
| Ret (r : A) : itree E A
| Tau (t : itree E A) : itree E A
| Vis B (e : E B) (k : B → itree E A) : itree E A.
```

```
Variant mapE : Type → Type :=
| Get : string → mapE (option nat)
| Set : string → nat → mapE unit.
```

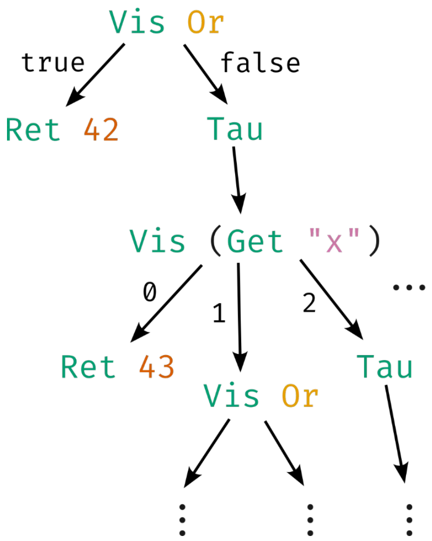
```
Variant nondetE : Type → Type :=
| Or : nondetE bool.
```

Interaction trees*

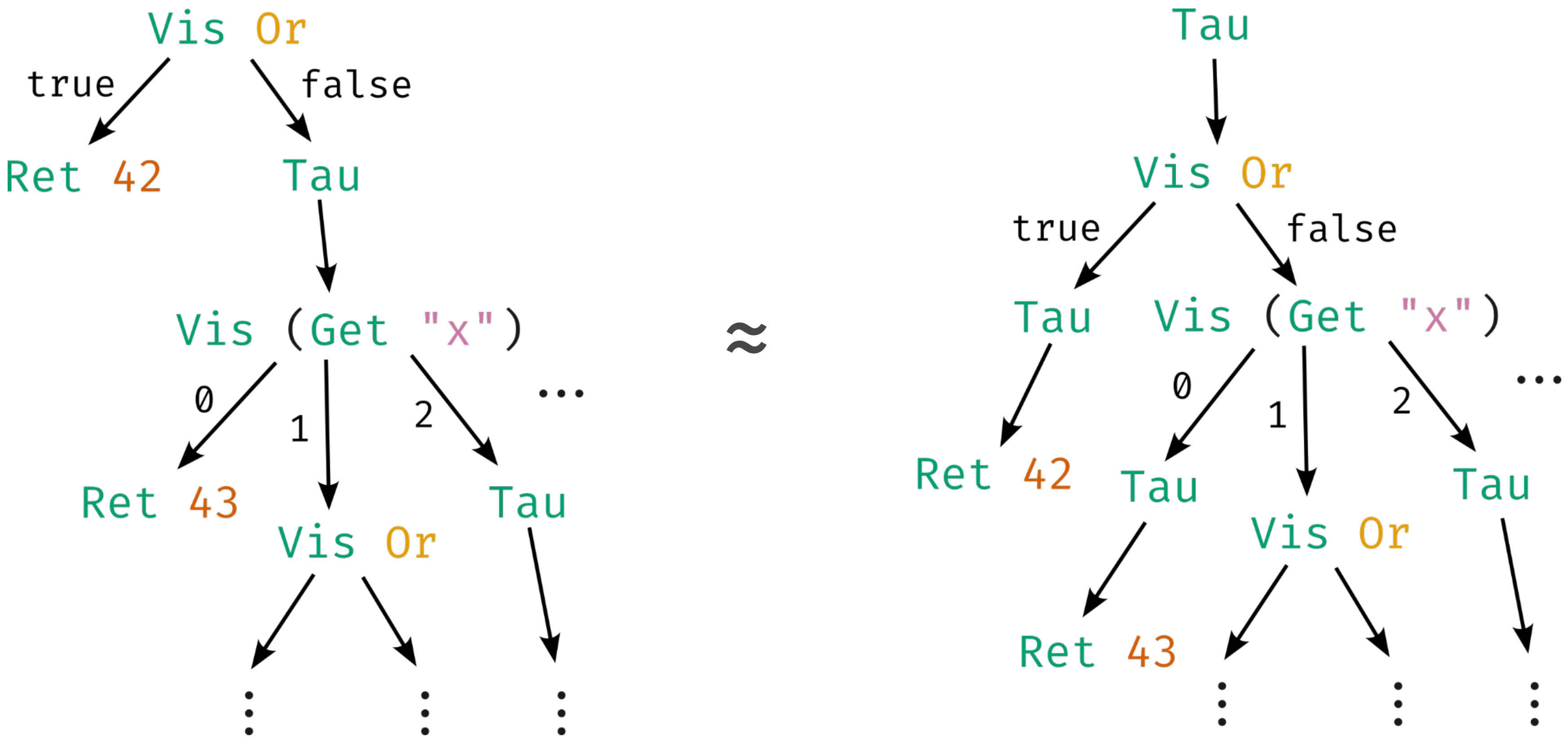
```
CoInductive itree (E : Type → Type) (A : Type) :=
| Ret (r : A) : itree E A
| Tau (t : itree E A) : itree E A
| Vis B (e : E B) (k : B → itree E A) : itree E A.
```

```
Variant mapE : Type → Type :=
| Get : string → mapE (option nat)
| Set : string → nat → mapE unit.
```

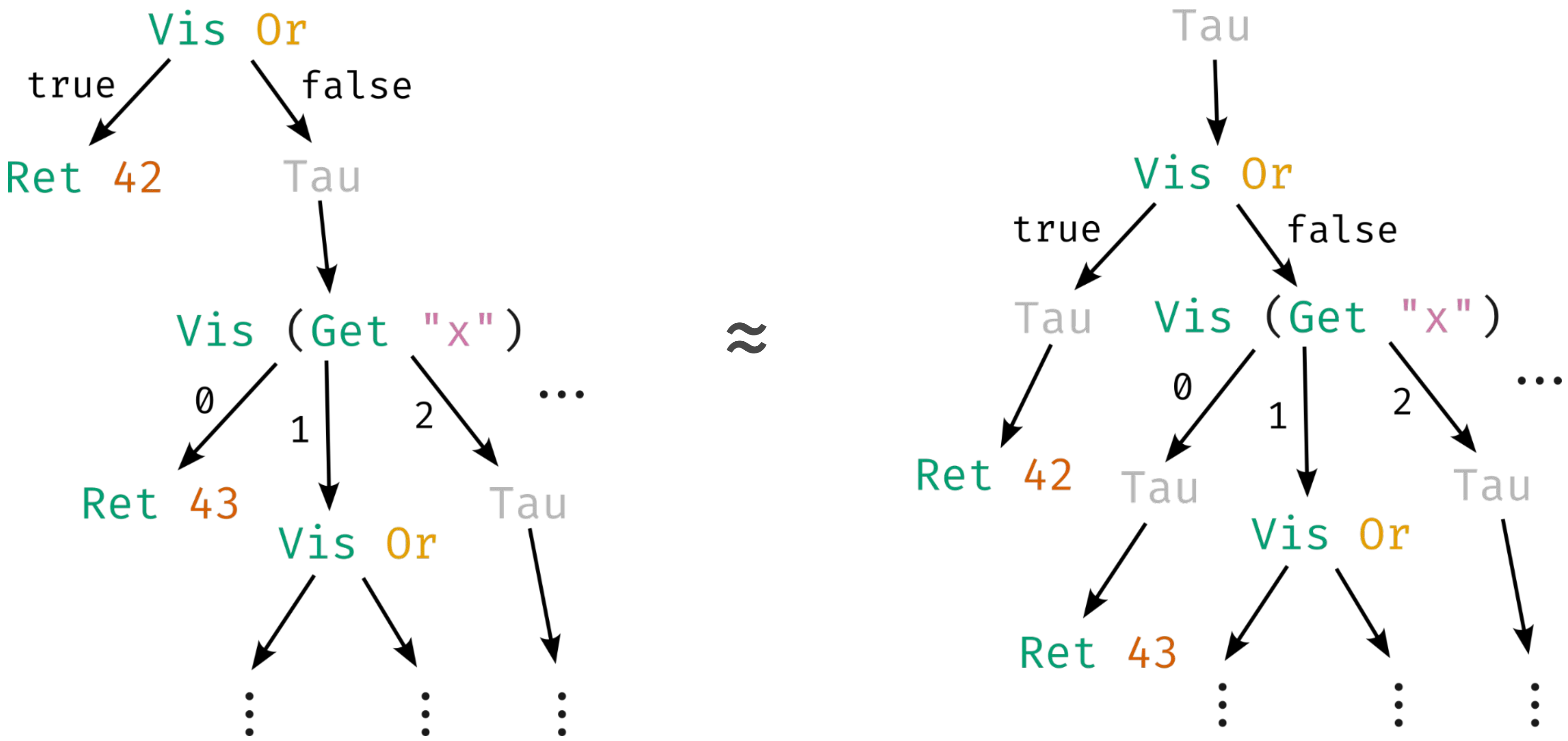
```
Variant nondetE : Type → Type :=
| Or : nondetE bool.
```



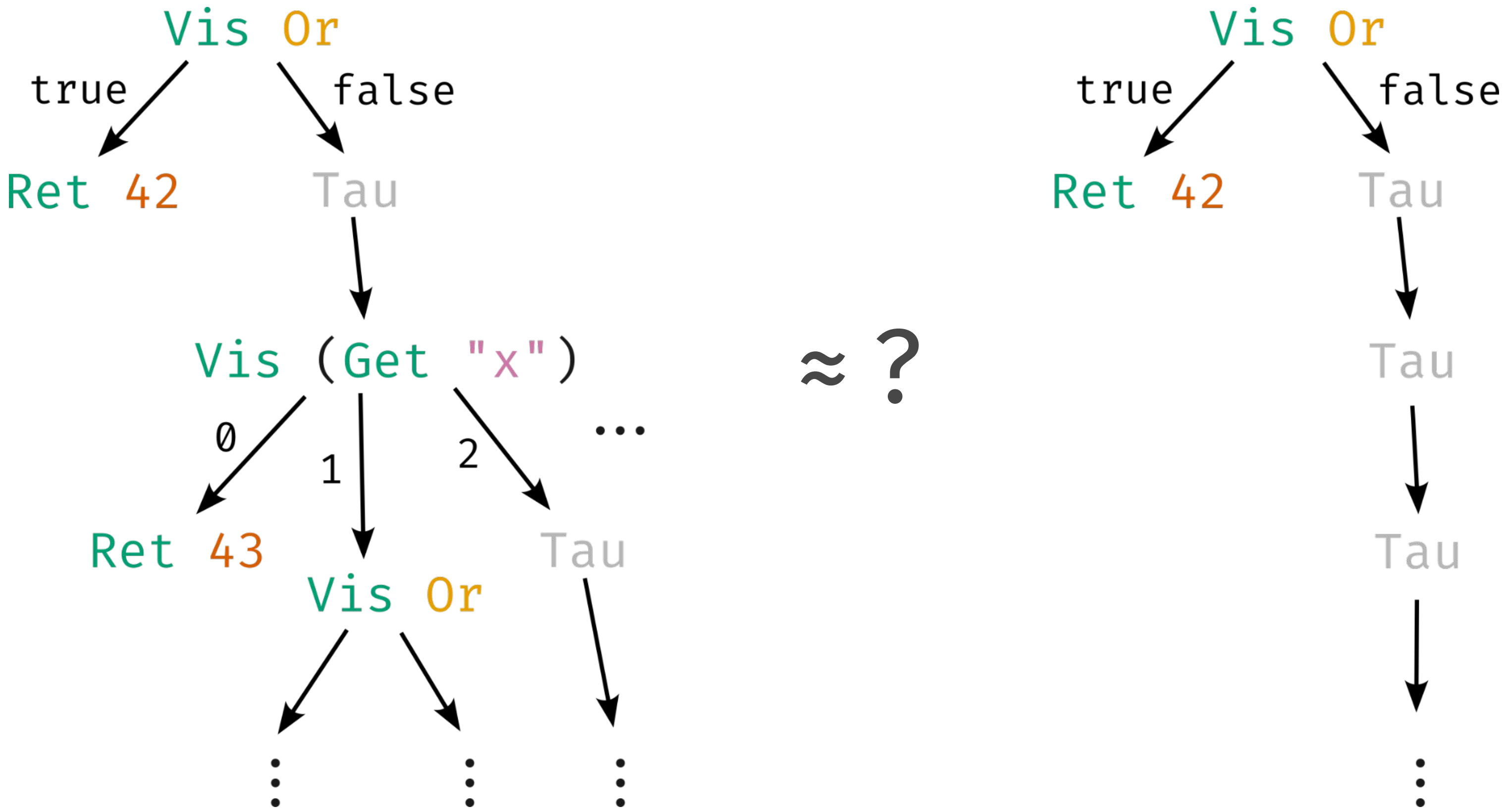
Equivalence up to taus (eutt)



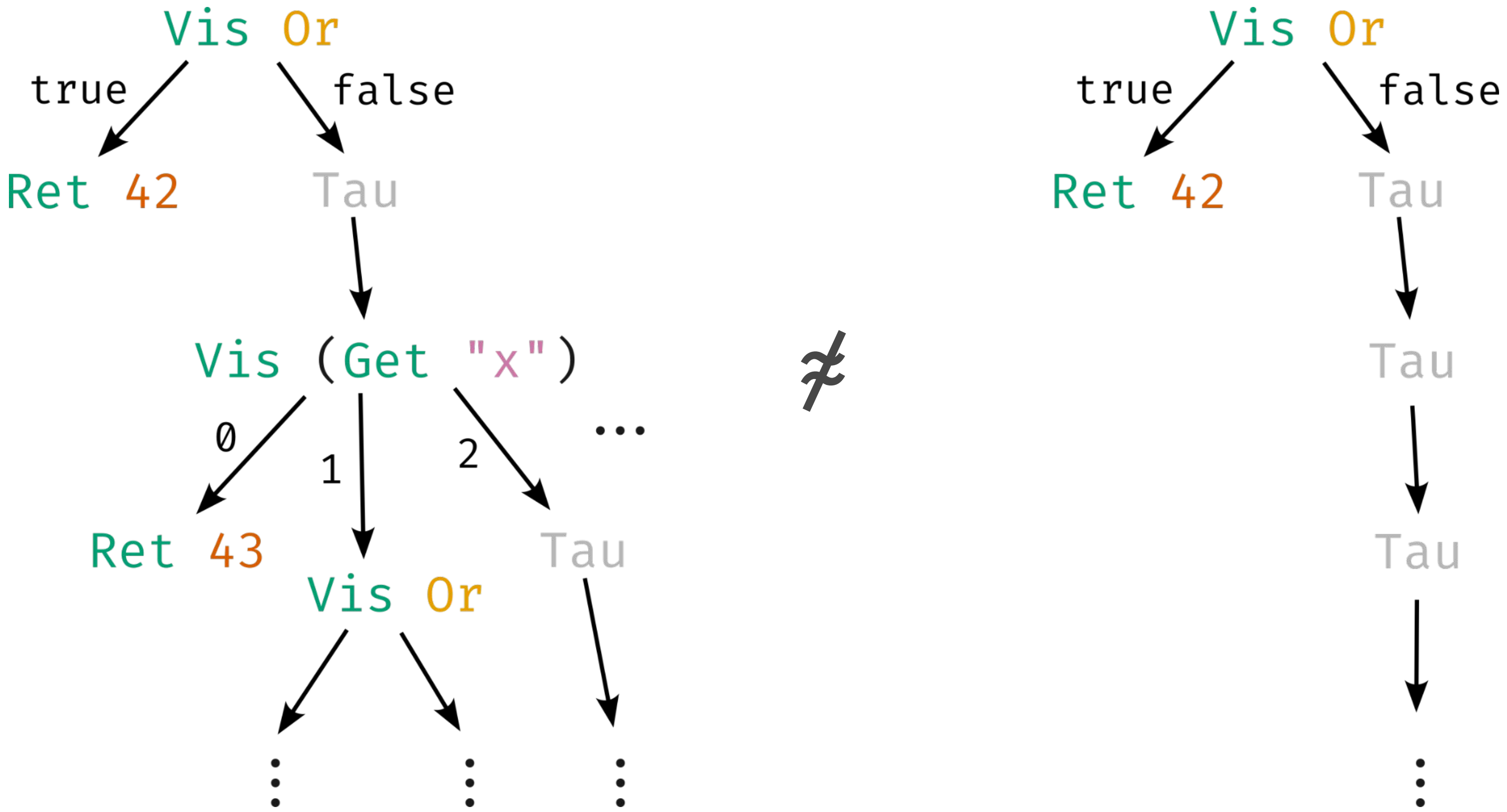
Equivalence up to taus (eutt)



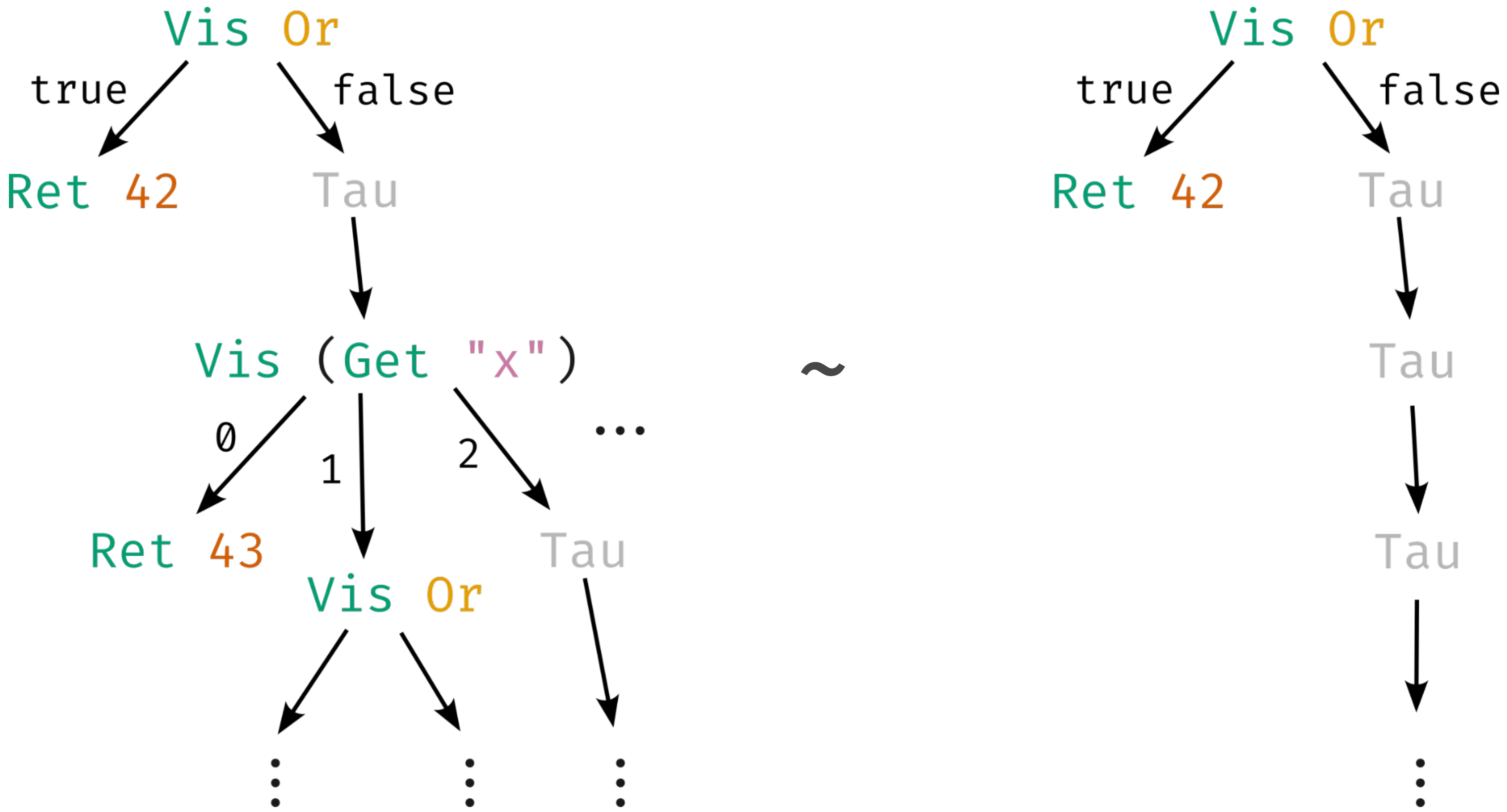
Equivalence up to taus (eutt)



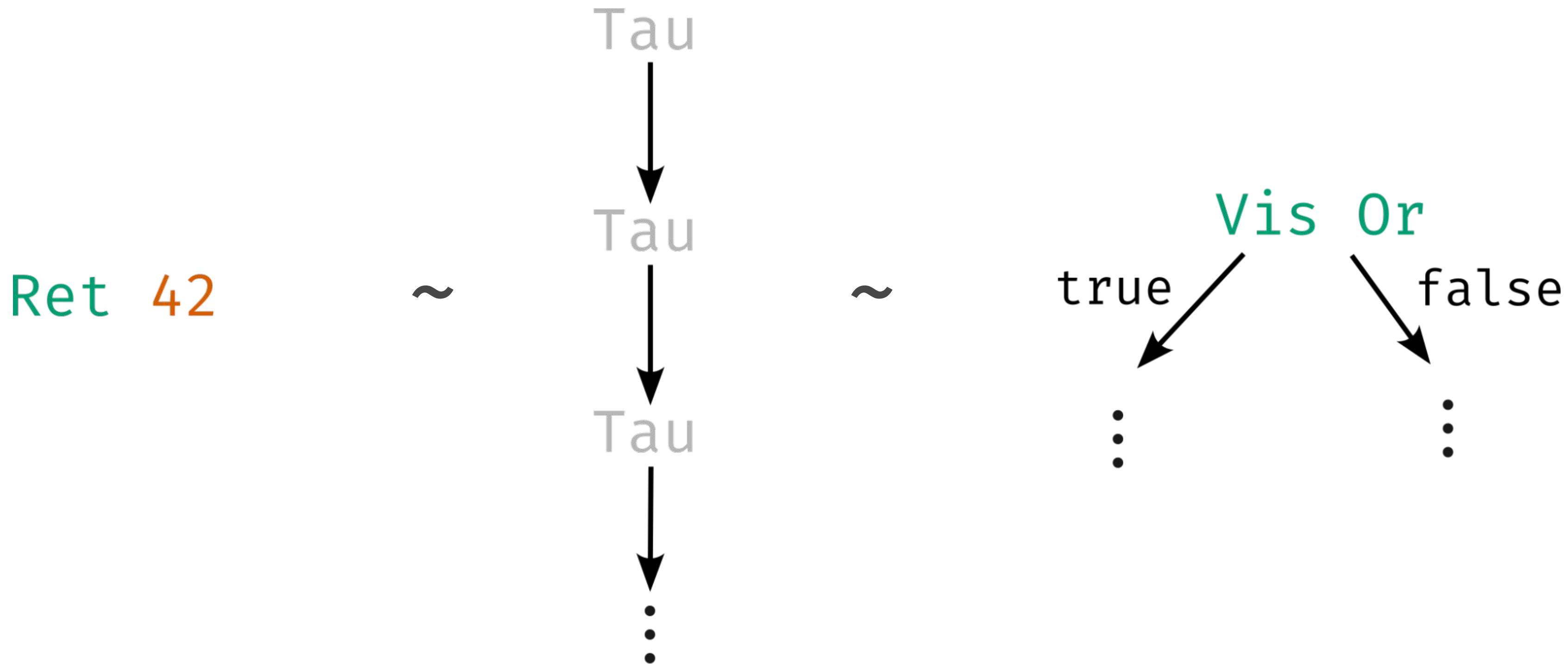
Equivalence up to taus (eutt)



Tolerance up to taus (tutt)



Tolerance up to taus



Tolerance up to taus

$$\frac{}{t \sim t}\ \text{Refl} \qquad \frac{t_1 \sim t_2}{t_2 \sim t_1}\ \text{Sym}$$

$$\frac{t_1 \sim t_2 \quad t_1 \approx t_3 \quad t_2 \approx t_4}{t_3 \sim t_4}\ \text{EuttCompat}$$

$$\frac{t_1 \sim t_2 \quad \forall v, k_1\ v \sim k_2\ v}{t_1 \gg= k_1 \sim t_2 \gg= k_2}\ \text{Bind}$$

Proof sketch

```
atomic (x) {
  y := !x;
  x := !y + 1
}
```

Proof sketch

$$
\begin{tikzcd}
\mathrm{run}\ (y := !x;\ x := !y + 1)\ \sigma_1 \arrow[d] & \sim & \mathrm{run}\ (y := !x;\ x := !y + 1)\ \sigma_2 \arrow[d] \\
\mathrm{run}\ (x := !y + 1)\ \sigma_1[y \mapsto v_x] \arrow[d] & \sim & \mathrm{run}\ (x := !y + 1)\ \sigma_2[y \mapsto v_x] \arrow[d] \\
\mathrm{ret}\ \sigma_1[y \mapsto v_x][x \mapsto v_x + 1] \arrow[dr] & \sim & \mathrm{ret}\ \sigma_2[y \mapsto v_x][x \mapsto v_x + 1] \arrow[dl] \\
& \sigma_1[y \mapsto v_x][x \mapsto v_x + 1] \ = \ \sigma_2[y \mapsto v_x][x \mapsto v_x + 1] &
\end{tikzcd}
$$

Proof sketch

$$
\begin{tikzcd}
\mathrm{run}\ (y := !x;\ x := !y + 1)\ \sigma_1 \arrow[d] \arrow[loop left, "\lightning"] \arrow[r, phantom, "\sim"] & \mathrm{run}\ (y := !x;\ x := !y + 1)\ \sigma_2 \arrow[d] \arrow[loop right, "\lightning"] \\
\mathrm{run}\ (x := !y + 1)\ \sigma_1[y \mapsto v_x] \arrow[d] \arrow[u, bend left=60, "\lightning"] \arrow[r, phantom, "\sim"] & \mathrm{run}\ (x := !y + 1)\ \sigma_2[y \mapsto v_x] \arrow[d] \arrow[u, bend right=60, "\lightning"'] \\
\mathrm{ret}\ \sigma_1[y \mapsto v_x][x \mapsto v_x + 1] \arrow[d] \arrow[uu, bend left=70, "\lightning"] \arrow[r, phantom, "\sim"] & \mathrm{ret}\ \sigma_2[y \mapsto v_x][x \mapsto v_x + 1] \arrow[d] \arrow[uu, bend right=70, "\lightning"'] \\
\sigma_1[y \mapsto v_x][x \mapsto v_x + 1] \arrow[r, phantom, "="] & \sigma_2[y \mapsto v_x][x \mapsto v_x + 1]
\end{tikzcd}
$$

Proof sketch

$$
\begin{tikzcd}
\mathsf{run}\ (y := !x;\ x := !y + 1)\ \sigma_1 \arrow[d] \arrow[loop left, "\lightning"] \arrow[r, phantom, "\sim"] & \mathsf{run}\ (y := !x;\ x := !y + 1)\ \sigma_2 \arrow[d] \arrow[loop right, "\lightning"] \\
\mathsf{run}\ (x := !y + 1)\ \sigma_1[y \mapsto v_x] \arrow[d] \arrow[u, bend left=60, "\lightning"] \arrow[r, phantom, "\sim"] & \mathsf{run}\ (x := !y + 1)\ \sigma_2[y \mapsto v_x] \arrow[d] \arrow[u, bend right=60, "\lightning"'] \\
\mathsf{ret}\ \sigma_1[y \mapsto v_x][x \mapsto v_x + 1] \arrow[d] \arrow[uu, bend left=70, "\lightning"] \arrow[r, phantom, "\sim"] & \mathsf{ret}\ \sigma_2[y \mapsto v_x][x \mapsto v_x + 1] \arrow[d] \arrow[uu, bend right=70, "\lightning"'] \\
\sigma_1[y \mapsto v_x][x \mapsto v_x + 1] \arrow[r, phantom, "="] & \sigma_2[y \mapsto v_x][x \mapsto v_x + 1]
\end{tikzcd}
$$

Proof sketch

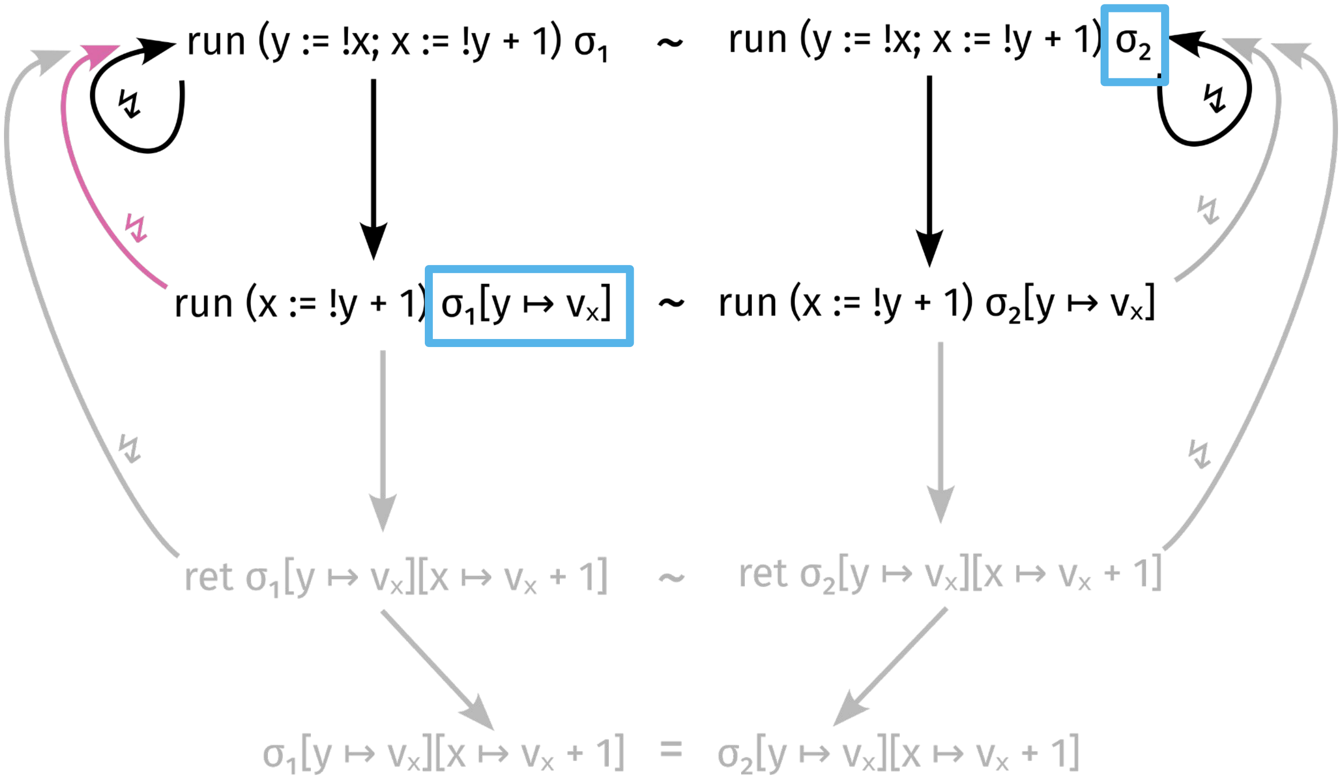
$$
\begin{tikzcd}
\mathrm{run}\ (y := {!x};\ x := {!y} + 1)\ \sigma_1 \arrow[d] \arrow[loop left, "\lightning"] & \sim & \mathrm{run}\ (y := {!x};\ x := {!y} + 1)\ \sigma_2 \arrow[d] \arrow[loop right, "\lightning"] \\
\mathrm{run}\ (x := {!y} + 1)\ \sigma_1[y \mapsto v_x] \arrow[d] \arrow[u, bend left, "\lightning"] & \sim & \mathrm{run}\ (x := {!y} + 1)\ \sigma_2[y \mapsto v_x] \arrow[d] \arrow[u, bend right, "\lightning"'] \\
\mathrm{ret}\ \sigma_1[y \mapsto v_x][x \mapsto v_x + 1] \arrow[d] \arrow[uu, bend left, "\lightning"] & \sim & \mathrm{ret}\ \sigma_2[y \mapsto v_x][x \mapsto v_x + 1] \arrow[d] \arrow[uu, bend right, "\lightning"'] \\
\sigma_1[y \mapsto v_x][x \mapsto v_x + 1] & = & \sigma_2[y \mapsto v_x][x \mapsto v_x + 1]
\end{tikzcd}
$$

Proof sketch

$$
\begin{tikzcd}
\mathrm{run}\ (y := {!x};\ x := {!y} + 1)\ \sigma_1 \arrow[loop left, "\lightning"] \arrow[d] \arrow[r, phantom, "\sim"] & \mathrm{run}\ (y := {!x};\ x := {!y} + 1)\ \boxed{\sigma_2} \arrow[loop right, "\lightning"] \arrow[d] \\
\mathrm{run}\ (x := {!y} + 1)\ \boxed{\sigma_1[y \mapsto v_x]} \arrow[u, bend left, "\lightning"] \arrow[d] \arrow[r, phantom, "\sim"] & \mathrm{run}\ (x := {!y} + 1)\ \sigma_2[y \mapsto v_x] \arrow[u, bend right, "\lightning"'] \arrow[d] \\
\mathrm{ret}\ \sigma_1[y \mapsto v_x][x \mapsto v_x + 1] \arrow[uu, bend left, "\lightning"] \arrow[d] \arrow[r, phantom, "\sim"] & \mathrm{ret}\ \sigma_2[y \mapsto v_x][x \mapsto v_x + 1] \arrow[uu, bend right, "\lightning"'] \arrow[d] \\
\sigma_1[y \mapsto v_x][x \mapsto v_x + 1] \arrow[r, phantom, "="] & \sigma_2[y \mapsto v_x][x \mapsto v_x + 1]
\end{tikzcd}
$$

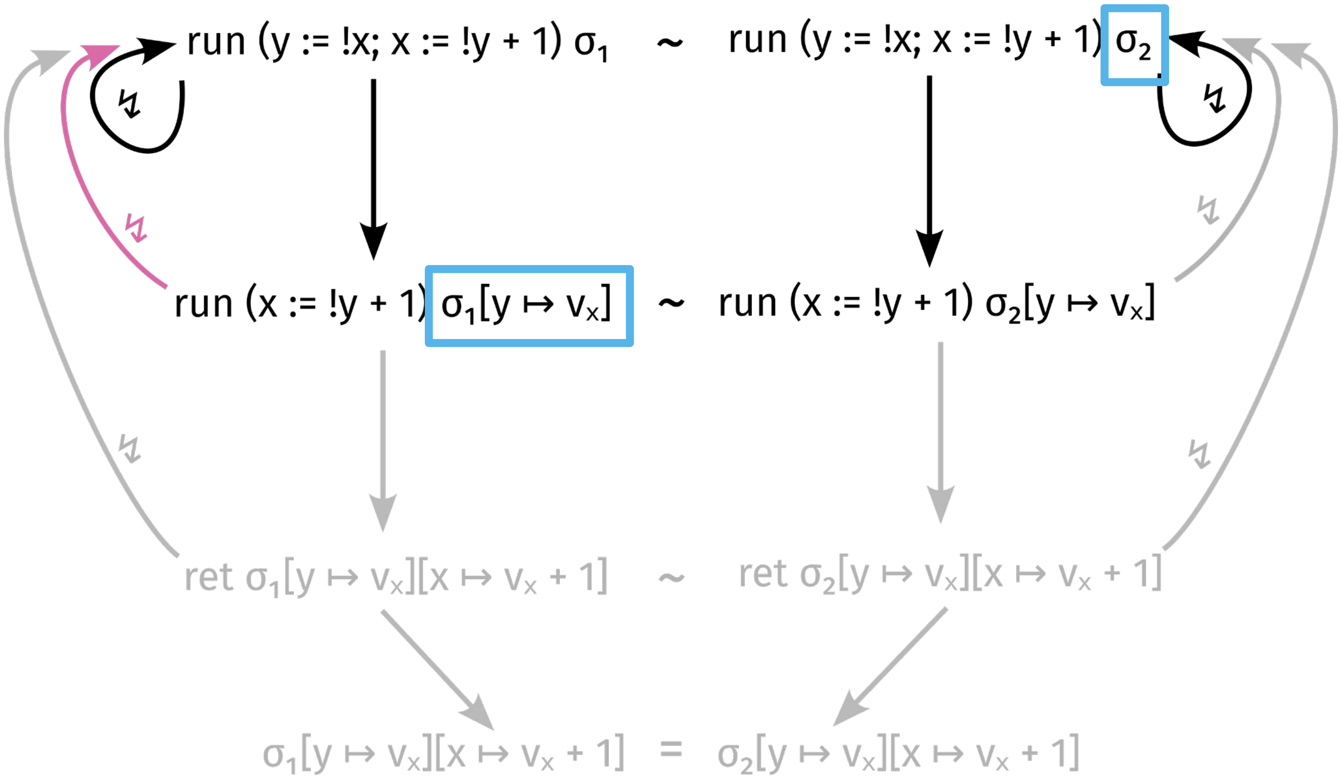
Proof sketch

$\mathrm{Inv}(\sigma_1, \sigma_2) := \forall\ k \in K, \sigma_1(k) = \sigma_2(k)?$



Proof sketch

$\mathrm{Inv}(\sigma_1, \sigma_2) := \forall\ k \in K/\{y\}, \sigma_1(k) = \sigma_2(k).$



Proof sketch

$\mathrm{Inv}(\sigma_1, \sigma_2) := \forall\ k \in K/\{y\}, \sigma_1(k) = \sigma_2(k).$

$$
\begin{tikzcd}
\mathrm{run}\ (y := !x;\ x := !y + 1)\ \sigma_1 \arrow[d] \arrow[loop left, "↯"] & \sim & \mathrm{run}\ (y := !x;\ x := !y + 1)\ \sigma_2 \arrow[d] \arrow[loop right, "↯"'] \\
\mathrm{run}\ (x := !y + 1)\ \sigma_1[y \mapsto v_x] \arrow[d] \arrow[u, bend left, "↯"] & \sim & \mathrm{run}\ (x := !y + 1)\ \sigma_2[y \mapsto v_x] \arrow[d] \arrow[u, bend right, "↯"'] \\
\mathrm{ret}\ \sigma_1[y \mapsto v_x][x \mapsto v_x + 1] \arrow[dr] \arrow[uu, bend left=60, "↯"] & \sim & \mathrm{ret}\ \sigma_2[y \mapsto v_x][x \mapsto v_x + 1] \arrow[dl] \arrow[uu, bend right=60, "↯"'] \\
\sigma_1[y \mapsto v_x][x \mapsto v_x + 1] & = & \sigma_2[y \mapsto v_x][x \mapsto v_x + 1]
\end{tikzcd}
$$

Proof sketch

$\mathrm{Inv}(\sigma_1, \sigma_2) := \forall\ k \in K/\{y\}, \sigma_1(k) = \sigma_2(k).$

$$
\begin{tikzcd}
\text{run } (y := !x;\ x := !y + 1)\ \sigma_1 \arrow[d] \arrow[loop left, "\lightning"] & \sim & \text{run } (y := !x;\ x := !y + 1)\ \sigma_2 \arrow[d] \arrow[loop right, "\lightning"'] \\
\text{run } (x := !y + 1)\ \sigma_1[y \mapsto v_x] \arrow[d] \arrow[u, bend left=50, "\lightning"] & \sim & \text{run } (x := !y + 1)\ \sigma_2[y \mapsto v_x] \arrow[d] \arrow[u, bend right=50, "\lightning"'] \\
\text{ret } \sigma_1[y \mapsto v_x][x \mapsto v_x + 1] \arrow[d] \arrow[uu, bend left=60, "\lightning"] & \sim & \text{ret } \sigma_2[y \mapsto v_x][x \mapsto v_x + 1] \arrow[d] \arrow[uu, bend right=60, "\lightning"'] \\
\sigma_1[y \mapsto v_x][x \mapsto v_x + 1] & = & \sigma_2[y \mapsto v_x][x \mapsto v_x + 1]
\end{tikzcd}
$$

Proof sketch

$\mathrm{Inv}(\sigma_1, \sigma_2) := \forall\ k \in K/\{y\}, \sigma_1(k) = \sigma_2(k).$

$$\begin{tikzcd}
\mathrm{run}\ (y := !x;\ x := !y + 1)\ \sigma_1 \arrow[d] \arrow[loop left, "\lightning"] \arrow[r, phantom, "\sim"] & \mathrm{run}\ (y := !x;\ x := !y + 1)\ \sigma_2 \arrow[d] \arrow[loop right, "\lightning"] \\
\mathrm{run}\ (x := !y + 1)\ \sigma_1[y \mapsto v_x] \arrow[d] \arrow[u, bend left, "\lightning"] \arrow[r, phantom, "\sim"] & \mathrm{run}\ (x := !y + 1)\ \sigma_2[y \mapsto v_x] \arrow[d] \arrow[u, bend right, "\lightning"'] \\
\mathrm{ret}\ \sigma_1[y \mapsto v_x][x \mapsto v_x + 1] \arrow[d] \arrow[uu, bend left=60, "\lightning"] \arrow[r, phantom, "\sim"] & \mathrm{ret}\ \sigma_2[y \mapsto v_x][x \mapsto v_x + 1] \arrow[d] \arrow[uu, bend right=60, "\lightning"'] \\
\sigma_1[y \mapsto v_x][x \mapsto v_x + 1] \arrow[r, phantom, "="] & \sigma_2[y \mapsto v_x][x \mapsto v_x + 1]
\end{tikzcd}$$

Proof sketch

$\mathrm{Inv}(\sigma_1, \sigma_2) := \forall\ k \in K/\{y\}, \sigma_1(k) = \sigma_2(k).$

$$
\begin{tikzcd}
\mathrm{run}\ (y := !x;\ x := !y + 1)\ \sigma_1 \arrow[d] \arrow[loop left, "↯"] \arrow[r, phantom, "\sim"] & \mathrm{run}\ (y := !x;\ x := !y + 1)\ \sigma_2 \arrow[d] \arrow[loop right, "↯"] \\
\mathrm{run}\ (x := !y + 1)\ \sigma_1[y \mapsto v_x] \arrow[d] \arrow[u, bend left, "↯"] \arrow[r, phantom, "\sim"] & \mathrm{run}\ (x := !y + 1)\ \sigma_2[y \mapsto v_x] \arrow[d] \arrow[u, bend right, "↯"'] \\
\mathrm{ret}\ \sigma_1[y \mapsto v_x][x \mapsto v_x + 1] \arrow[d] \arrow[uu, bend left=60, "↯"] \arrow[r, phantom, "\sim"] & \mathrm{ret}\ \sigma_2[y \mapsto v_x][x \mapsto v_x + 1] \arrow[d] \arrow[uu, bend right=60, "↯"'] \\
\sigma_1[y \mapsto v_x][x \mapsto v_x + 1] \arrow[r, phantom, "="] & \sigma_2[y \mapsto v_x][x \mapsto v_x + 1]
\end{tikzcd}
$$

Proof sketch

$\mathrm{Inv}(\sigma_1, \sigma_2) := \forall\ k \in K/\{y\},\ \sigma_1(k) = \sigma_2(k).$

$$
\begin{tikzcd}
\mathrm{run}\ (y := {!x};\ x := {!y} + 1)\ \sigma_1 \arrow[d] \arrow[loop left, "↯"] & \sim & \mathrm{run}\ (y := {!x};\ x := {!y} + 1)\ \sigma_2 \arrow[d] \arrow[loop right, "↯"] \\
\mathrm{run}\ (x := {!y} + 1)\ \sigma_1[y \mapsto v_x] \arrow[d] \arrow[u, bend left=60, "↯"] & \sim & \mathrm{run}\ (x := {!y} + 1)\ \sigma_2[y \mapsto v_x] \arrow[d] \arrow[u, bend right=60, "↯"'] \\
\mathrm{ret}\ \sigma_1[y \mapsto v_x][x \mapsto v_x + 1] \arrow[d] \arrow[uu, bend left=70, "↯"] & \sim & \mathrm{ret}\ \sigma_2[y \mapsto v_x][x \mapsto v_x + 1] \arrow[d] \arrow[uu, bend right=70, "↯"'] \\
\sigma_1[y \mapsto v_x][x \mapsto v_x + 1] & = & \sigma_2[y \mapsto v_x][x \mapsto v_x + 1]
\end{tikzcd}
$$

Proof sketch

$\mathrm{Inv}(\sigma_1, \sigma_2) := \forall\ k \in K/\{y\}, \sigma_1(k) = \sigma_2(k).$

$$
\begin{tikzcd}
\mathrm{run}\ (y := !x;\ x := !y + 1)\ \sigma_1 \arrow[loop left, "\lightning"] \arrow[d] & \sim & \mathrm{run}\ (y := !x;\ x := !y + 1)\ \sigma_2 \arrow[loop right, "\lightning"'] \arrow[d] \\
\mathrm{run}\ (x := !y + 1)\ \sigma_1[y \mapsto v_x] \arrow[d] \arrow[u, bend left=60, "\lightning"] & \sim & \mathrm{run}\ (x := !y + 1)\ \sigma_2[y \mapsto v_x] \arrow[d] \arrow[u, bend right=60, "\lightning"'] \\
\mathrm{ret}\ \sigma_1[y \mapsto v_x][x \mapsto v_x + 1] \arrow[d] \arrow[uu, bend left=70, "\lightning"] & \sim & \mathrm{ret}\ \sigma_2[y \mapsto v_x][x \mapsto v_x + 1] \arrow[d] \arrow[uu, bend right=70, "\lightning"'] \\
\sigma_1[y \mapsto v_x][x \mapsto v_x + 1] & = & \sigma_2[y \mapsto v_x][x \mapsto v_x + 1]
\end{tikzcd}
$$

Tolerating a lack of productivity, in summary

- Eutt is too strict for verification of crash recovery systems.
- Tutt enables reasoning up to indistinguishability.
- Mechanized in ROCQ
- Some additional properties:
    - Tutt of two *productive* itrees implies they are eutt.
    - Can rewrite by itree equivalences while writing coinductive proofs of tutt.

Show definitions here

```
Inductive euttF sim :
    itree E R1 → itree E R2 → Prop :=
| EqRet : ∀ r1 r2,
    RR r1 r2 →
    euttF sim (Ret r1) (Ret r2)
| EqTau : ∀ m1 m2,
    sim m1 m2 →
    euttF sim (Tau m1) (Tau m2)
| EqVis : ∀ u (e : E u) k1 k2,
    (∀ v, sim (k1 v) (k2 v)) →
    euttF sim (Vis e k1) (Vis e k2)
| EqTauL : ∀ t1 t2,
    euttF sim t1 t2 →
    euttF sim (Tau t1) t2
| EqTauR : ∀ t1 t2,
    euttF sim t1 t2 →
    euttF sim t1 (Tau t2)
.
```

```
Variant tuttF sim :
    itree E R1 → itree E R2 → Prop :=
| TolRet : ∀ r1 r2,
    RR r1 r2 →
    tuttF sim (Ret r1) (Ret r2)
| TolTau : ∀ m1 m2,
    sim m1 m2 →
    tuttF sim (Tau m1) (Tau m2)
| TolVis : ∀ u (e : E u) k1 k2,
    (∀ v, sim (k1 v) (k2 v)) →
    tuttF sim (Vis e k1) (Vis e k2)
| TolTauL : ∀ t1 t2,
    sim t1 t2 →
    tuttF sim (Tau t1) t2
| TolTauR : ∀ t1 t2,
    sim t1 t2 →
    tuttF sim t1 (Tau t2)
.
```

Show definitions here

```
Inductive euttF sim :
    itree E R1 → itree E R2 → Prop :=
| EqRet : ∀ r1 r2,
    RR r1 r2 →
    euttF sim (Ret r1) (Ret r2)
| EqTau : ∀ m1 m2,
    sim m1 m2 →
    euttF sim (Tau m1) (Tau m2)
| EqVis : ∀ u (e : E u) k1 k2,
    (∀ v, sim (k1 v) (k2 v)) →
    euttF sim (Vis e k1) (Vis e k2)
| EqTauL : ∀ t1 t2,
    euttF sim t1 t2 →
    euttF sim (Tau t1) t2
| EqTauR : ∀ t1 t2,
    euttF sim t1 t2 →
    euttF sim t1 (Tau t2)
.
```

```
Variant tuttF sim :
    itree E R1 → itree E R2 → Prop :=
| TolRet : ∀ r1 r2,
    RR r1 r2 →
    tuttF sim (Ret r1) (Ret r2)
| TolTau : ∀ m1 m2,
    sim m1 m2 →
    tuttF sim (Tau m1) (Tau m2)
| TolVis : ∀ u (e : E u) k1 k2,
    (∀ v, sim (k1 v) (k2 v)) →
    tuttF sim (Vis e k1) (Vis e k2)
| TolTauL : ∀ t1 t2,
    sim t1 t2 →
    tuttF sim (Tau t1) t2
| TolTauR : ∀ t1 t2,
    sim t1 t2 →
    tuttF sim t1 (Tau t2)
.
```

Show definitions here

```
Inductive euttF sim :
    itree E R1 → itree E R2 → Prop :=
| EqRet : ∀ r1 r2,
    RR r1 r2 →
    euttF sim (Ret r1) (Ret r2)
| EqTau : ∀ m1 m2,
    sim m1 m2 →
    euttF sim (Tau m1) (Tau m2)
| EqVis : ∀ u (e : E u) k1 k2,
    (∀ v, sim (k1 v) (k2 v)) →
    euttF sim (Vis e k1) (Vis e k2)
| EqTauL : ∀ t1 t2,
    euttF sim t1 t2 →
    euttF sim (Tau t1) t2
| EqTauR : ∀ t1 t2,
    euttF sim t1 t2 →
    euttF sim t1 (Tau t2)
.
```

```
Variant tuttF sim :
    itree E R1 → itree E R2 → Prop :=
| TolRet : ∀ r1 r2,
    RR r1 r2 →
    tuttF sim (Ret r1) (Ret r2)
| TolTau : ∀ m1 m2,
    sim m1 m2 →
    tuttF sim (Tau m1) (Tau m2)
| TolVis : ∀ u (e : E u) k1 k2,
    (∀ v, sim (k1 v) (k2 v)) →
    tuttF sim (Vis e k1) (Vis e k2)
| TolTauL : ∀ t1 t2,
    sim t1 t2 →
    tuttF sim (Tau t1) t2
| TolTauR : ∀ t1 t2,
    sim t1 t2 →
    tuttF sim t1 (Tau t2)
.
```