Type Theory of Processes
A beginning

Uday S. Reddy

University of Birmingham
(Joint work with Claudio Hermida and Edmund Robinson)

Imperial Concurrency Workshop, 2015
Section 1

The Big Picture
The basic intuitions go a long way back:
  - Felix Klein - Erlangen Programme.
  - Henri Poincare.

Whenever we define a mathematical concept, we are forced to include some essential information as well as some inessential information.

The inessential information gives rise to symmetries, i.e., differences that cannot be observed within the theory.

In programming languages, these symmetries show up in observational equivalences.
Reynolds "type theory"

- Reynolds’s idea was that we could characterize the essential/inessential information by writing types.
- If we have the “right” type, then we get the right notion of symmetries and the right observational equivalences.
- If we don’t get the right equivalences, then we must go back and find the right types.
- So, the types are everything!
- It is a paradigm of denotational semantics, extending Strachey’s idea of “domains for denotational semantics” ("domain" being Strachey’s term for a semantic type).
In Klein-Poincare times, the “symmetries” were isomorphisms.

In our times, the “symmetries” are logical relations.

Relations have a long history:

- Turing: virtual types — logical partial equivalence relations.
- Tarski: logical notion.
- Tait, Martin-Lof, Howard: logical predicates.
- Ginzburg & Yeoli (automata theory): generalized homomorphisms; Milner: simulation relations.
- Gordon & Plotkin: logical relations, Reynolds: admissible relations.
- Reynolds [1983]: Types, abstraction and parametric polymorphism.
- O’Hearn & Tennent [1993]: Parametricity and Local Variables.
Logical Relations and Parametricity —
A Reynolds Programme for Category Theory
and Programming Languages

Claudio Hermida
Uday S. Reddy
Edmund P. Robinson

Dedicated to the memory of John C. Reynolds, 1935-2013

[Power and Wingfield: Workshop on Algebra, Coalgebra and Topology (WACT 2014)]
Three levels of type theories

- **Set theory**: types (sets).
- **Category theory**: types, morphisms.
- **Reynolds type theory**: types, morphisms, relations.
- Category theory introduces *distinctions*.
  - E.g., *Complete lattices* and *complete semilattices* are distinguished by their morphisms (even though the types are the same).
- Reynolds type theory introduces *further distinctions*.
  - E.g., *Groups* and *monoids with inverses* are distinguished by their logical relations (even though the types and morphisms are the same).
The Big Picture

- The objective of this work is to demonstrate these ideas for concurrent process theory.
- The “inessential information” in formulating processes is in the states.
  - The states are completely hidden; not observable to the outside.
- Hence, relations between states appear as “symmetries” in process theory.
- **Note**: “Symmetry” means a change that cannot be observed.
The Big Picture (Parametricity)

We write \( t_A [F(R) \rightarrow G(R)] t_{A'} \) to represent the square, and mean

\[
\forall x, x'. x [F(R)] x' \implies t_A(x) [G(R)] t_{A'}(x')
\]
Section 2

Processes
Processes

- Understanding processes semantically is difficult.
  - They are reactive.
  - They are nondeterministic.
  - No agreement on what is observable.

- Three well-known equivalences.
  - **Trace equivalence**: If two processes may accept the same traces. [Automata theory]
  - **Bisimilarity**: If two processes maintain equivalence at every step. [Milner and Park]
  - **Testing equivalence**: If two processes pass the same tests. [de Nicola and Hennessy]
Example processes

- Three examples

\[ X : \; ab(c + d) \quad Y : \; a(bc + bd) \quad Z : \; abc + abd \]

- Trace equivalence identifies all three.
- Bisimilarity distinguishes all three.
- Testing equivalence identifies \( Y \) and \( Z \), while distinguishing them from \( X \).
Classical distinctions

▶ “may” vs “must”:
  ▶ \( X \) may accept \( abc \); it also must accept \( abc \).
  ▶ \( Y \) and \( Z \) may accept \( abc \); \neg\)(they must accept \( abc \)).
  ▶ trace equivalence only captures may acceptance.

▶ “linear time” vs “branching time”:
  ▶ trace equivalence is regarded as a “linear time” idea because traces represent a linear progression of time.
  ▶ bisimilarity is regarded as a “branching time” idea (time “branches” at each choice point).
  ▶ what about testing equivalence?

▶ reactive vs transformational:
  ▶ trace equivalence only looks at the net effect of an entire run.
  ▶ testing equivalence and bisimilarity look at what is possible at each point in the run.
  ▶ what exactly is observable at each point?
Confused?

Type theory to the rescue!
Effects

- **Effects** are computational phenomena other than values (or in addition to values).
  - **Divergence** or **undefinedness**: A computation may not produce a result.
  - **Nondeterminism**: A computation may produce one out of a possible set of results.
- In normal programming languages, effects are observable only at the **top-level**, i.e., for entire runs of programs.
- In **reactive systems**, effects may also be observable at intermediate steps.
Effects examples

- The Three examples

\[ X : \, ab(c + d) \quad Y : \, a(bc + bd) \quad Z : \, abc + abd \]

- Observing divergence at intermediae steps:
  - E.g., the \( Y \) process, when given \( abc \), may get stuck after \( ab \).
  - Is nondeterminism observable at intermediate steps, e.g., \( Y \) vs. \( Z \)?
    - This is called “branching time” [van Glabbeek].
    - We might also think of it as “snap back.”
Equivalences in terms of effects

- **Trace equivalence** assumes that no effects are observable at intermediate steps. Both divergence and nondeterminism are observable only for entire runs.

- **Bisimilarity** assumes that both divergence and nondeterminism are observable at intermediate steps.

- **Testing equivalence** assumes that divergence is observable at intermediate steps, but nondeterminism only for the entire run.
Monads for effects

- Effects are represented in type theories as monads [Moggi].
- A monad $T = \langle T, \eta, \mu \rangle$ is a structure on an endofuctor $T : C \to C$.
  - unit $\eta_X : X \to TX$ views a value is a (null) computation.
  - multiplication $\mu_X : TTX \to TX$ collapses cascaded computations.
- Call-by-value languages are modelled using Kleisli composition:

  $X \xrightarrow{f} TY \quad \frac{Y \xrightarrow{g} TZ}{TY \xrightarrow{Tg} TTZ}$

  $X \xrightarrow{f} TY \xrightarrow{Tg} TTX \xrightarrow{\mu_Z} TTX \xrightarrow{T} TZ$

- For reactive systems, it seems that we just cascade computations without collapsing them:

  $X \xrightarrow{f_0} TX \xrightarrow{Tf_1} TTX \xrightarrow{TTf_2} TTTX \xrightarrow{TTTf_3} \ldots$
The Monads

- **Divergence:** \( P^1 : \text{Set} \to \text{Set} \) (the “subsinglesons”). \( P^1 X \) includes \( \emptyset \) and singletons \( \{x\} \).
- **Real nondeterminism:** \( P^+ : \text{Set} \to \text{Set} \) (nonempty powerset). \( P^+ X \) contains the nonempty subsets of \( X \).
- **Combined nondeterminism:** \( P : \text{Set} \to \text{Set} \) (powerset).

In all three cases:

- unit \( \eta_X : X \to TX \) is the singleton operation: \( x \mapsto \{x\} \).
- multiplication \( \mu_X : TTX \to TX \) is union. For example, for \( \mu_X : P^1 P^1 X \to P^1 X \), the mapping is:

\[
\begin{align*}
\emptyset & \mapsto \emptyset \\
\{\emptyset\} & \mapsto \emptyset \\
\{\{x\}\} & \mapsto \{x\}
\end{align*}
\]

- It can be shown that \( P \cong P^1 P^+ \) is the **composite** monad. This involves a distributivity operation \( \lambda_X : P^+ P^1 X \to P^1 P^+ X \) given by

\[
\begin{align*}
\{\emptyset\} & \mapsto \emptyset \\
\{\ldots, u_i, \ldots\} & \mapsto \{\bigcup_i u_i\}
\end{align*}
\]
Section 3

Labelled transition systems
Labelled transition systems

- A labelled transition system (LTS), for an alphabet of symbols $A$, is a pair

  $\langle Q, \{ a \rightarrow \} a \in A \rangle$

  where $a \rightarrow$ is a binary relation on $Q$.

- $s \rightarrow$ for a sequence $s \in A^*$ is the obvious extension of the $a \rightarrow$ relation.

- Write $x \downarrow_s$ if there exists $x'$ such that $x \xrightarrow{s} x'$.

- A process is an LTS together with an initial state $x_0$.

  $\langle Q, \{ a \rightarrow \} a \in A, x_0 \rangle$
Process behaviour

- The traces behavior of a process $P$ is
  
  \[
  \text{traces}(P) = \{ s \mid x_0 \Downarrow s \}
  \]

- $\text{traces}(X) = \text{traces}(Y) = \text{traces}(Z)$. It is the prefix closure of $\{abc, abd\}$.

- The testing behavior of a process $P$ is the collection of responses for each trace. A “response” is a maximal successful subtrace of the trace.

  \[
  \text{testing}(X) = \{(abc, abc), (abd, abd)\}.
  \]

- The tree behaviour of a process is an unordered “tree”.
  \[
  \text{Tree} = \mathcal{P}(A \times \text{Tree}).
  \]
  This is a recursive (coinductive) definition!

  \[
  \text{tree}(X) = \{a : \{b : \{c : \emptyset, d : \emptyset\}\}\}\}
  \]
Testing behaviour

- Three examples:

\[
X : \ ab(c + d) \quad Y : \ a(bc + bd) \quad Z : \ abc + abd
\]

\[
\begin{array}{ccc}
0 & 0 & 0 \\
| & | & | \\
a & a & a \\
1 & 1 & 11 \\
b & b & b \\
c & 2 & 21 \\
d & d & 22 \\
31 & 32 & 31 & 32 & 31 & 32 \\
\end{array}
\]

- \(\text{testing}(X) = \{(abc, abc), (abd, abd)\}\).
- \(\text{testing}(Y) = \{(abc, abc), (abc, ab), (abd, ab), (abd, abd)\}\).
- \(\text{testing}(Z) = \{(abc, abc), (abc, ab), (abd, ab), (abd, abd)\}\).
- This definition of testing behaviour is new.
- It is equivalent (?) to the de Nicola and Hennessy definition as well as the failures semantics.
Tree behaviour

- Three examples:

\[ X : \ ab(c + d) \quad Y : \ a(bc + bd) \quad Z : \ abc + abd \]

\[
\begin{array}{c}
0 \\
| a \\
| b \\
| c & d \\
31 & 32
\end{array}
\begin{array}{c}
0 \\
| a \\
| b \\
| c \\
31
\end{array}
\begin{array}{c}
a \\
| b \\
| c \\
31 \\
| d \\
32
\end{array}
\]

- \( \text{tree}(X) = \{ a : \{ b : \{ c : \emptyset, \ d : \emptyset \} \} \} \). 
- \( \text{tree}(Y) = \{ a : \{ b : \{ c : \emptyset \}, \ b : \{ d : \emptyset \} \} \} \). 
- \( \text{tree}(Z) = \{ a : \{ b : \{ c : \emptyset \} \}, \ a : \{ b : \{ d : \emptyset \} \} \} \).
Process equivalence

- The traces behaviour, testing behaviour and the tree behaviour are increasingly refined.

\[
\text{tree}(P_1) = \text{tree}(P_2) \\
\implies \text{testing}(P_1) = \text{testing}(P_2) \\
\implies \text{traces}(P_1) = \text{traces}(P_2)
\]

- **Bisimulation** is a reasoning principle for the equivalence of tree behaviour (also called **bisimilarity**).

- Classical **automata-theoretic** techniques provide reasoning methods for the traces behaviour.

- For testing behaviour, there is no widely known reasoning principle, even though Cleaveland and Hennessy have provided the essential idea.
Section 4

Type theory of tree behaviour
Type theory of tree behaviour (bisimilarity)

- A process is a triple:

\[ P = \langle Q, \{ \overset{a}{\to} \}_{a \in A}, x_0 \rangle \]

- Consider various types for the “transition relation:”
  - \( F(Q) = \mathcal{P}(A \times Q \times Q) \) - set-theoretic view.
  - \( F(Q) = A \to \mathcal{P}(Q \times Q) \) - monoid view.
  - \( F(Q) = A \to Q \to \mathcal{P}Q \) - nondeterministic functions.
  - \( F(Q) = Q \to \mathcal{P}(A \times Q) \) - with outputs.

- All these types are isomorphic as sets. However, their type-theoretic interpretation varies.
  - \( F(R) = \mathcal{P}(I_A \times R \times R) \).
  - \( F(R) = I_A \to \mathcal{P}(R \times R) \).
  - \( F(R) = I_A \to R \to \mathcal{P}R \).
  - \( F(R) = R \to \mathcal{P}(I_A \times R) \).

And there are even more variations, as we shall see later.
Relation actions (background)

- Relation actions for “×” and “→”:
  
  \[ p \ [ R \times S ] p' \iff \pi_1(p) \ [ R ] \pi_1(p') \land \pi_2(p) \ [ R ] \pi_2(p') \]
  \[ f \ [ R \rightarrow S ] f' \iff (\forall x, x'. x \ [ R ] x' \implies f(x) \ [ S ] f'(x')) \]

- Relation action for \( \mathcal{P} \) (the “Egli-Milner” powerset relator):

  \[ u \ [ \mathcal{P}R ] u' \iff (\forall x \in u. \exists x' \in u'. x \ [ R ] x') \land
  (\forall x' \in u'. \exists x \in u. x \ [ R ] x') \]

- Sample parametric operations:

  \[
  \mathcal{P}_{AB} : (A \rightarrow B) \rightarrow (\mathcal{P}A \rightarrow \mathcal{P}B) \\
  \{-\} : A \rightarrow \mathcal{P}A \\
  \cup_A : \mathcal{P}A \times \mathcal{P}A \rightarrow \mathcal{P}A \\
  \bigcup_A : \mathcal{P}(\mathcal{P}A) \rightarrow \mathcal{P}A
  \]

- Intersection is not parametric.
Relational structure for bisimilarity

- The relational structure $\text{Bisim}(A)$ is defined to have:
  - types: processes $P = \langle Q, \alpha : A \rightarrow [Q \rightarrow \mathcal{P} Q], x_0 : Q \rangle$.
  - relations: $R : \langle Q, \alpha, x_0 \rangle \leftrightarrow \langle Q', \alpha', x'_0 \rangle$ are relations $R \subseteq Q \times Q'$ such that:
    $$\alpha \left[ I_A \rightarrow [R \rightarrow \mathcal{P} R] \right] \alpha'$$
    $$x_0 \left[ R \right] x'_0$$
  - identity relations are the usual ones.

A logical relation between processes of this kind is called a bisimulation.

- **Fact**: The tree behaviour is parametric (in $Q$):
  $$\text{tree} : \forall Q [A \rightarrow [Q \rightarrow \mathcal{P} Q]] \rightarrow Q \rightarrow \text{Tree}(A)$$

- **Fact**: Bisimulation is sound and complete for tree equivalence:
  $$(\exists R. P \left[ R \right] P') \iff \text{tree}(P) = \text{tree}(P')$$

So, bisimulations precisely capture the symmetries of the tree behaviour.
Bisimulations and behaviours

- The traces behaviour and the testing behaviour are more abstract than tree behaviour.
- So, bisimulations are also symmetries for them:

  traces : $\forall Q [A \rightarrow [Q \rightarrow \mathcal{P}Q]] \rightarrow Q \rightarrow \hat{\mathcal{P}}(A^*)$

  testing : $\forall Q [A \rightarrow [Q \rightarrow \mathcal{P}Q]] \rightarrow Q \rightarrow \hat{\mathcal{P}}(A^* \times A^*)$

Ergo, bisimulations represent a sound reasoning principle for trace equivalence and testing equivalence as well.

- However, the converse is not true. There are more symmetries in the traces and testing behaviour that are not captured by bisimulations.
Section 5

Type theory of traces behaviour
Algebraic view of nondeterminism

- Nondeterministic functions $X \to \mathcal{P}Y$ can also be viewed as additive functions $\mathcal{P}X \to \mathcal{P}Y$.
- “Additive” means preserve union:
  \[ h(\bigcup_{i \in I} u_i) = \bigcup_{i \in I} h(u_i) \]
- Since every set $\{x_i\}_{i \in I} \in \mathcal{P}X$ can be written as a union $\bigcup_{i \in I} \{x_i\}$, we have:
  \[ h(\bigcup_{i \in I} \{x_i\}) = \bigcup_{i \in I} h(\{x_i\}) \]

So $h$ is uniquely determined by its action on the singletons.

- $X \to \mathcal{P}Y$ represents the morphisms of the Kleisli category of the $\mathcal{P}$ monad.
- $A \to B$ represents the homomorphisms of $\mathcal{P}$-algebras, i.e., the morphisms of the Eilenberg-Moore category of the $\mathcal{P}$ monad.
New distinctions

- The equivalence between $[X \to \mathcal{P}Y]$ and $[\mathcal{P}X \circ \mathcal{P}Y]$ does not extend to relations.

- Logical relations of $\mathcal{P}$-algebras $S : \mathcal{P}X \leftrightarrow \mathcal{P}Y$ are **additive relations**:

$$\left( \forall i \in I. \, u_i \left[ S \right] u_i' \right) \implies \bigcup_{i \in I} u_i \left[ S \right] \bigcup_{i \in I} u_i'$$

Not all additive relations need be of the form $\mathcal{P}R$.

- An additive relation is an “algebraic relation”. A relation of the form $\mathcal{P}R$ is a “free algebraic relation.”

- The concept of **Kleisli category** splits into two when we consider relations (in Reynolds type theory), currently code-named the “Moggi category” and “Reddy category” respectively.

- The “Moggi category” represents call-by-value effects (allows “snap backs”). The “Reddy category” represents call-by-name effects (no “snap back”, **irreversible** effects).
Relational structure for traces behaviour

- The relational structure $\textbf{Aut}(A)$ is defined to have:
  - types: processes $P = \langle Q, \alpha : A \rightarrow [\mathcal{P}Q \circ \mathcal{P}Q], x_0 : Q \rangle$.
  - relations: $S : \langle Q, \alpha, x_0 \rangle \leftrightarrow \langle Q', \alpha', x_0' \rangle$ are additive relations $S : \mathcal{P}Q \leftrightarrow \mathcal{P}Q'$ such that:
    $$\alpha \left[ I_A \rightarrow [S \circ S] \right] \alpha'$$
    $$\{x_0\} \left[ S \right] \{x_0'\}$$
    and also strict:
    $$u \left[ S \right] u' \Longrightarrow (u = \emptyset \iff u' = \emptyset)$$
- identity relations are the usual ones.
- A logical relation between automata of this kind is called an **automatic relation**.
- **Fact**: Automatic relations are sound and complete for trace equivalence:
  $$(\exists S. P \left[ S \right] P') \iff \text{traces}(P) = \text{traces}(P')$$

So, automatic relations precisely capture the **symmetries** of the traces behaviour.
Example

The three examples:

\[ X : ab(c + d) \quad Y : a(bc + bd) \quad Z : abc + abd \]

Automatic relation \( \mathcal{P}Q_X \leftrightarrow \mathcal{P}Q_Y \) and \( \mathcal{P}Q_Y \leftrightarrow \mathcal{P}Q_Z \):

\[
\begin{align*}
\{0\} & \quad \{0\} & \quad \{0\} \\
\{1\} & \quad \{1\} & \quad \{11, 12\} \\
\{2\} & \quad \{21, 22\} & \quad \{21, 22\} \\
\{31\} & \quad \{31\} & \quad \{31\} \\
\{32\} & \quad \{32\} & \quad \{32\}
\end{align*}
\]

and other tuples generated by additivity.
Information hiding aspects of traces

- Note that $S : \mathcal{P}Q \leftrightarrow \mathcal{P}Q'$ is not a free relation.
- The automata theorists knew this a long time ago!
- The structure of relations $S : \mathcal{P}Q \leftrightarrow \mathcal{P}Q'$ means that states ($Q$) are hidden and also the effects ($\mathcal{P}$).
- In other words, no effects are observable at the intermediate steps, only observable for entire runs.
Section 6

Type theory of testing behaviour
Separating nondeterminism and deadlock

- The $\mathcal{P}$ monad combines two effects:
  - *divergence* (or undefinedness): $\emptyset$.
  - *nondeterminism*: $\{x_1, \ldots, x_n\}$.

- The testing behaviour says that divergence is **observable** at intermediate steps but nondeterminism is **not observable**. So, we need to **separate** the two effects.

- Recall the isomorphism $\mathcal{P} \cong \mathcal{P}^1 \mathcal{P}^+$:
  - $\mathcal{P}^1X$ is the type of "subsingeltons" (sets of at most one element). We use the "lifting" notation, *i.e.*,
    \[
    \emptyset \leadsto \bot \quad \{x\} \leadsto x
    \]
  - $\mathcal{P}^+X$ is the type of nonempty subsets.
  The relation action is the same as that of $\mathcal{P}X$ (Egli-Milner).

- However, this decomposition does not achieve anything.
A functor for divergence

Consider the functor $\mathcal{P}^2$ such that $\mathcal{P}^2X (\cong \mathcal{P}^+\mathcal{P}^1X)$ has three kinds of elements (using square brackets instead of set braces):

- $[\bot]$ - a computation that has definitely diverged.
- $[x]$ - a computation with a defined result.
- $[\bot, x]$ - a computation that has the possibility of divergence as well as a defined result.

The relation action $\mathcal{P}^2R$ is similar to $\mathcal{P}^+(\mathcal{P}^1R)$.

- $[\bot]$ is only related to $[\bot]$.
- $[x]$ is related to $[x']$ iff $x [R] x'$.
- $[\bot, x]$ is related to $[\bot, x']$ iff $x [R] x'$.

This functor has a distributive law:

$$\lambda_X : \mathcal{P}^+\mathcal{P}^2X \to \mathcal{P}^2\mathcal{P}^+X$$

$$\lambda_X (\{z_i\}_{i \in I}) = [\bot | \exists i \in I. \bot \in z_i] \cup [\{x | \exists i \in I. x \in z_i\}]$$

This implies that the $\mathcal{P}^2$ functor lifts to $\mathcal{P}^+$-algebras.

$$\tilde{\mathcal{P}}^2 : \text{Alg}(\mathcal{P}^+) \to \text{Alg}(\mathcal{P}^+)$$
Relational structure for testing behaviour

- The relational structure $\text{Proc}(A)$ is defined to have:
  - types: processes
    \[ P = \langle Q, \alpha : A \to [P^+ Q \rightarrow \mathcal{P}^2 P^+ Q], x_0 : Q \rangle. \]
  - relations: $S : \langle Q, \alpha, x_0 \rangle \leftrightarrow \langle Q', \alpha', x'_0 \rangle$ are additive relations
    \[ S : P^+ Q \leftrightarrow P^+ Q' \] such that:
    \[ \alpha \left[ I_A \rightarrow [S \rightarrow \mathcal{P}^2 S] \right] \alpha' \]
    \[ \{x_0\} [S] \{x'_0\} \]
  - identity relations are the usual ones.
- $\mathcal{P}^2$ represents divergence, which is visible at intermediate steps.
- $\mathcal{P}^+$ represents real nondeterminism, which is invisible.
- **Fact**: The process relations are sound for testing equivalence:
  \[ (\exists S. P [S] P') \implies \text{testing}(P) = \text{testing}(P') \]
  Completeness yet to be determined
Example

Consider the examples $X$, $Y$ and $Z$:

$X : \ ab(c + d)$

$Y : \ a(bc + bd)$

$Z : \ abc + abd$

The transition behaviour of the processes $X$ can be seen by:

$X : \{0\} \xrightarrow{\alpha_a} \{1\} \xrightarrow{P^2 \alpha_b} [[[2]]] \xrightarrow{P^2 P^2 \alpha_c} [[[31]]]$

$Y : \{0\} \xrightarrow{\alpha_a} \{1\} \xrightarrow{P^2 \alpha_b} [[[21, 22]]] \xrightarrow{P^2 P^2 \alpha_c} [[[31}, \bot]]$

$Z : \{0\} \xrightarrow{\alpha_a} \{11, 12\} \xrightarrow{P^2 \alpha_b} [[[21, 22]]] \xrightarrow{P^2 P^2 \alpha_c} [[[31}, \bot]]$
Example (contd)

- The examples $X$, $Y$ and $Z$:

\[
X : \ ab(c + d) \quad Y : \ a(bc + bd) \quad Z : \ abc + abd
\]

- The transition behaviour of the processes is given by:

\[
X : \ \alpha_{abc}(\{0\}) = [\alpha_{bc}(\{1\})] = [[[\alpha_c(\{2\})]]] = [[[\{31\}]]]
\]

\[
Y : \ \alpha_{abc}(\{0\}) = [\alpha_{bc}(\{1\})] = [[[\alpha_c(\{21, 22\})]]] = [[[\{31\}, \bot]]]
\]

\[
Z : \ \alpha_{abc}(\{0\}) = [\alpha_{bc}(\{11, 12\})] = [[[\alpha_c(\{21, 22\})]]] = [[[\{31\}, \bot]]]
\]
Conclusion

- The notions of equivalence that arise in process theory can be explained by types.
- Logical relations and parametricity that arise in type theory match up with equivalences in process theory.
To be done

- There is no **concurrency** yet!
  - The focus was on **reactivity** (and nondeterminism).
  - Concurrency involves **silent transitions**, which I expect to be a tough challenge for a type-theoretic treatment.
  - Silent transitions should be hidden, but they make a difference!
- There is still a lot that needs to be understood about reactivity and testing.