Type Theory of Processes

A beginning

Uday S. Reddy[1]

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

Imperial Concurrency Workshop, 2015

Section 1

The Big Picture

Reynolds "type theory" (Erlangen Programme)

- 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 denotatational semantics" ("domain" being Strachey's term for a semantic type).

Relations as symmetries

- 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 Picutre (Parametricity)

$$
\begin{tikzcd}
A \arrow[dd, leftrightarrow, "R"'] & & F(A) \arrow[rr, "t_A"] \arrow[dd, leftrightarrow, "F(R)"'] & & G(A) \arrow[dd, leftrightarrow, "G(R)"] \\
& & & & \\
A' & & F(A') \arrow[rr, "t_{A'}"] & & G(A')
\end{tikzcd}
$$

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

$$\forall x, x'.\ x \left[F(R)\right] x' \implies t_A(x) \left[G(R)\right] t_{A'}(x')$$

Section 2

Processes

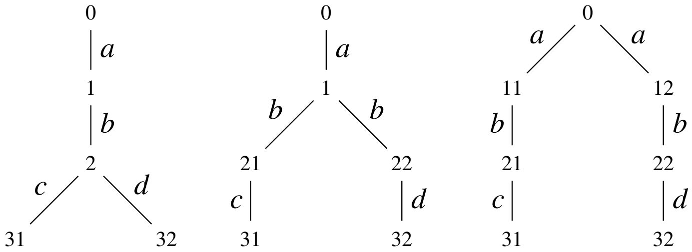
Processes

- 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

$X:\ ab(c+d)$ $\qquad$ $Y:\ a(bc+bd)$ $\qquad$ $Z:\ abc+abd$



- 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 equivlance 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!

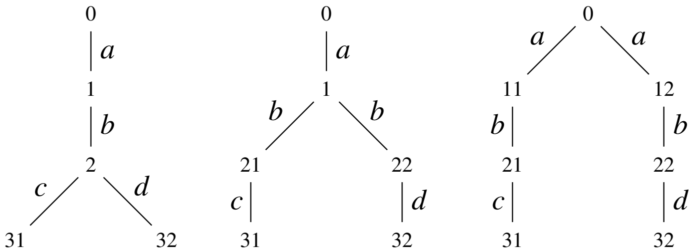
Effects

- 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

$X:\ ab(c+d)$ $\qquad$ $Y:\ a(bc+bd)$ $\qquad$ $Z:\ abc+abd$



- 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 $T : \mathcal{C} \to \mathcal{C}$.
  - 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:

$$\frac{X \xrightarrow{f} TY \qquad \dfrac{Y \xrightarrow{g} TZ}{TY \xrightarrow{Tg} TTZ}}{X \xrightarrow{f} TY \xrightarrow{Tg} TTZ \xrightarrow{\mu_Z} TZ}$$

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

$$X \xrightarrow{f_0} TX \xrightarrow{Tf_1} TTX \xrightarrow{TTf_2} TTTX \xrightarrow{TTTf_3} \cdots$$

The Monads

- Divergence: $\mathcal{P}^1 : \mathbf{Set} \to \mathbf{Set}$ (the "subsingletons"). $\mathcal{P}^1 X$ includes $\emptyset$ and singletons $\{x\}$.
- Real nondeterminism: $\mathcal{P}^+ : \mathbf{Set} \to \mathbf{Set}$ (nonempty powerset). $\mathcal{P}^+ X$ contains the nonempty subsets of X.
- Combined nondeterminism: $\mathcal{P} : \mathbf{Set} \to \mathbf{Set}$ (powerset).
- In all three cases:
  - unit $\eta_X : X \to TX$ is the singleton opearation: $x \mapsto \{x\}$.
  - multiplication $\mu_X : TTX \to TX$ is union. For example, for $\mu_X : \mathcal{P}^1\mathcal{P}^1 X \to \mathcal{P}^1 X$, the mapping is:

$$\begin{array}{rcl} \emptyset & \mapsto & \emptyset \\ \{\emptyset\} & \mapsto & \emptyset \\ \{\{x\}\} & \mapsto & \{x\} \end{array}$$

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

$$\begin{array}{rcl} \{\emptyset\} & \mapsto & \emptyset \\ \{\ldots, u_i, \ldots\} & \mapsto & \{\bigcup_i u_i\} \end{array}$$

Section 3

Labelled transition systems

Labelled transition systems

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

$$\langle Q, \{\xrightarrow{a}\}_{a\in A}\rangle$$

  where $\xrightarrow{a}$ is a binary relation on Q.
- $\xrightarrow{s}$ for a sequence $s \in A^*$ is the obvious extension of the $\xrightarrow{a}$ 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.

$$\langle Q,\ \{\xrightarrow{a}\}_{a\in A},\ x_0\rangle$$

Process behaviour

- The traces behavior of a process P is

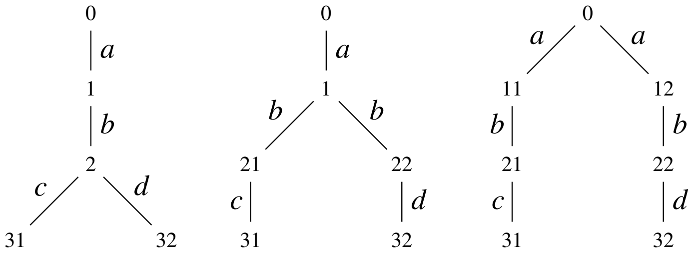
$$\textit{traces}(P) = \{\, s \mid x_0 \Downarrow_s \,\}$$

- $\textit{traces}(X) = \textit{traces}(Y) = \textit{traces}(Z)$. 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.
- $\textit{testing}(X) = \{(abc, abc),\ (abd, abd)\}$.
- The tree behaviour of a process is an unordered "tree".
  $\textit{Tree} = \mathcal{P}(A \times \textit{Tree})$.
  This is a recursive (coinductive) definition!
- $\textit{tree}(X) = \{a : \{b : \{c : \emptyset,\ d : \emptyset\}\}\}$

Testing behaviour

- Three examples:

$X:\ ab(c+d)$ $\qquad$ $Y:\ a(bc+bd)$ $\qquad$ $Z:\ abc+abd$

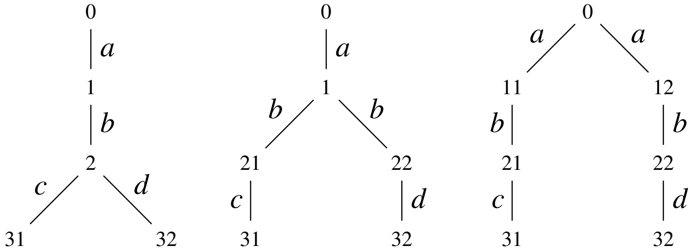


- *testing*$(X) = \{(abc, abc),\ (abd, abd)\}$.
- *testing*$(Y) = \{(abc, abc),\ (abc, ab),\ (abd, ab),\ (abd, abd)\}$.
- *testing*$(Z) = \{(abc, abc),\ (abc, ab),\ (abd, ab),\ (abd, abd)\}$.
- This definition of testing behaviour is new.
- It is equivalent (?) to the de Nicola and Hennessy definition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

Tree behaviour

- Three examples:

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



- *tree*$(X) = \{a : \{b : \{c : \emptyset,\ d : \emptyset\}\}\}$.
- *tree*$(Y) = \{a : \{b : \{c : \emptyset\},\ b : \{d : \emptyset\}\}\}$.
- *tree*$(Z) = \{a : \{b : \{c : \emptyset\}\},\ a : \{b : \{d : \emptyset\}\}\}$.

Process equivalence

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

$$\begin{aligned} &\textit{tree}(P_1) = \textit{tree}(P_2) \\ &\implies \textit{testing}(P_1) = \textit{testing}(P_2) \\ &\implies \textit{traces}(P_1) = \textit{traces}(P_2) \end{aligned}$$

- 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 Hennessey have provided the essential idea.

Section 4

Type theory of tree behaviour

Type theory of tree behaviour (bisimilarity)

- A process is a triple:

$$P = \langle Q, \{\xrightarrow{a}\}_{a \in A}, x_0 \rangle$$

- 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 "$\times$" and "$\rightarrow$":

$$
\begin{array}{rcl}
p \; \big[R \times S\big] \; p' & \iff & \pi_1(p) \; \big[R\big] \; \pi_1(p') \wedge \pi_2(p) \; \big[R\big] \; \pi_2(p') \\
f \; \big[R \rightarrow S\big] \; f' & \iff & (\forall x, x'.\, x \; \big[R\big] \; x' \implies f(x) \; \big[S\big] \; f'(x'))
\end{array}
$$

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

$$
\begin{array}{rcl}
u \; \big[\mathcal{P}R\big] \; u' & \iff & (\forall x \in u.\, \exists x' \in u'.\, x \; \big[R\big] \; x') \wedge \\
& & (\forall x' \in u'.\, \exists x \in u.\, x \; \big[R\big] \; x')
\end{array}
$$

- Sample parametric operations:

$$
\begin{array}{rcl}
\mathcal{P}_{AB} & : & (A \rightarrow B) \rightarrow (\mathcal{P}A \rightarrow \mathcal{P}B) \\
\{-\}_A & : & A \rightarrow \mathcal{P}A \\
\cup_A & : & \mathcal{P}A \times \mathcal{P}A \rightarrow \mathcal{P}A \\
\bigcup_A & : & \mathcal{P}(\mathcal{P}A) \rightarrow \mathcal{P}A
\end{array}
$$

- Intersection is not parametric.

Relational structure for bisimilarity

- The relational structure **Bisim**(A) is defined to have:
  - types: processes $P = \langle Q,\ \alpha : A \to [Q \to \mathcal{P}Q],\ x_0 : Q\rangle$.
  - relations: $R : \langle Q, \alpha, x_0\rangle \leftrightarrow \langle Q', \alpha', x_0'\rangle$ are relations $R \subseteq Q \times Q'$ such that:

    $$\alpha\ \left[I_A \to [R \to \mathcal{P}R]\right]\ \alpha'$$
    $$x_0\ \left[R\right]\ x_0'$$

  - 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):

  $$\mathit{tree} : \forall_Q\, [A \to [Q \to \mathcal{P}Q]] \to Q \to \mathit{Tree}(A)$$

- **Fact**: Bisimulation is sound and complete for tree equivalence:

  $$(\exists R.\, P\ \left[R\right]\ P') \iff \mathit{tree}(P) = \mathit{tree}(P')$$

  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:

$$\textit{traces} : \forall_Q \, [A \to [Q \to \mathcal{P}Q]] \to Q \to \widehat{\mathcal{P}}(A^*)$$
$$\textit{testing} : \forall_Q \, [A \to [Q \to \mathcal{P}Q]] \to Q \to \widehat{\mathcal{P}}(A^* \times A^*)$$

  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 $\mathcal{P}X \multimap \mathcal{P}Y$.
- "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.
- $A \multimap B$ 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 $[\mathcal{P}X \multimap \mathcal{P}Y]$ does not extend to relations.
- Logical relations of $\mathcal{P}$-algebras $S : \mathcal{P}X \leftrightarrow \mathcal{P}Y$ are additive relations:

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

  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 **Aut**(A) is defined to have:
  - types: processes $P = \langle Q,\ \alpha : A \to [\mathcal{P}Q \multimap \mathcal{P}Q],\ x_0 : Q\rangle$.
  - 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:

    $$\begin{array}{l} \alpha\ \left[I_A \to [S \multimap S]\right]\ \alpha' \\ \{x_0\}\ \left[S\right]\ \{x_0'\} \end{array}$$

    and also strict:

    $$u\ \left[S\right]\ u' \implies (u = \emptyset \iff u' = \emptyset)$$

  - 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:

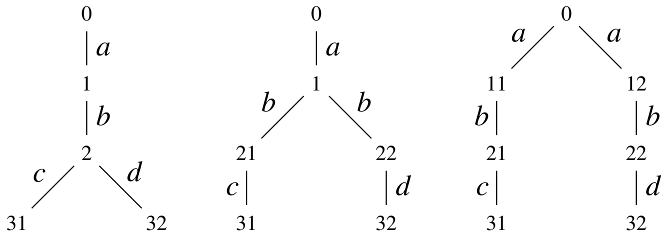
  $$(\exists S.\ P\ \left[S\right]\ P') \iff \textit{traces}(P) = \textit{traces}(P')$$

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

Example

- The three examples:

$X: \; ab(c+d)$ $\qquad$ $Y: \; a(bc+bd)$ $\qquad$ $Z: \; abc+abd$



- Automatic relation $\mathcal{P}Q_X \leftrightarrow \mathcal{P}Q_Y$ and $\mathcal{P}Q_Y \leftrightarrow \mathcal{P}Q_Z$:

$$\begin{array}{ccc}
\{0\} & \{0\} & \{0\} \\
\{1\} & \{1\} & \{11,12\} \\
\{2\} & \{21,22\} & \{21,22\} \\
\{31\} & \{31\} & \{31\} \\
\{32\} & \{32\} & \{32\}
\end{array}$$

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 comibines 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 $\mathcal{P} \cong \mathcal{P}^1\mathcal{P}^+$:
  - $\mathcal{P}^1 X$ is the type of "subsingletons" (sets of at most one element). We use the "lifting" notation, *i.e.*,

$$\begin{array}{ccc} \emptyset & \rightsquigarrow & \perp \\ \{x\} & \rightsquigarrow & x \end{array}$$

  - $\mathcal{P}^+ X$ 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 $\mathcal{P}^2 X(\cong \mathcal{P}^+\mathcal{P}^1 X)$ 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 $\mathcal{P}^2 R$ is similar to $\mathcal{P}^+(\mathcal{P}^1 R)$.
  - $[\bot]$ 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:

$$\lambda_X : \mathcal{P}^+\mathcal{P}^2 X \to \mathcal{P}^2\mathcal{P}^+ X$$
$$\lambda_X(\{z_i\}_{i\in I}) = [\, \bot \mid \exists i \in I.\, \bot \in z_i \,] \cup [\{\, x \mid \exists i \in I.\, x \in z_i \,\}]$$

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

$$\widetilde{\mathcal{P}^2} : \mathbf{Alg}(\mathcal{P}^+) \to \mathbf{Alg}(\mathcal{P}^+)$$

Relational structure for testing behaviour

- The relational structure **Proc**(A) is defined to have:
  - types: processes
    $P = \langle Q,\, \alpha : A \to [\mathcal{P}^+Q \multimap \mathcal{P}^2\mathcal{P}^+Q],\, x_0 : Q\rangle$.
  - 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:

$$\begin{array}{l} \alpha\ \left[I_A \to [S \multimap \mathcal{P}^2S]\right]\ \alpha' \\ \{x_0\}\ \left[S\right]\ \{x_0'\} \end{array}$$

  - 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:

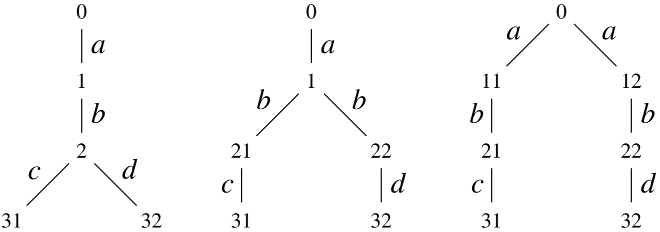
$$(\exists S.\, P\ \left[S\right]\ P') \implies \textit{testing}(P) = \textit{testing}(P')$$

Completeness yet to be determined

Example

- Consider the examples X, Y and Z:

$X:\ ab(c+d) \qquad Y:\ a(bc+bd) \qquad Z:\ abc+abd$



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

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

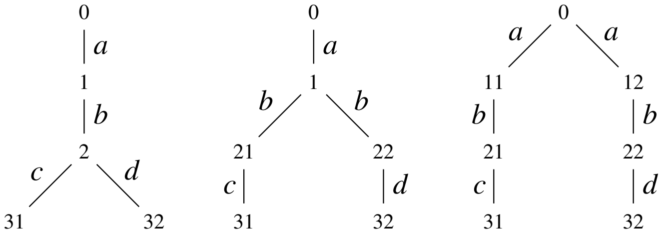
$$Y: \quad \{0\} \xmapsto{\alpha_a} [\{1\}] \xmapsto{\mathcal{P}^2\alpha_b} [[\{21,22\}]] \xmapsto{\mathcal{P}^2\mathcal{P}^2\alpha_c} [[[\{31\},\perp]]]$$

$$Z: \quad \{0\} \xmapsto{\alpha_a} [\{11,12\}] \xmapsto{\mathcal{P}^2\alpha_b} [[\{21,22\}]] \xmapsto{\mathcal{P}^2\mathcal{P}^2\alpha_c} [[[\{31\},\perp]]]$$

Example (contd)

- The examples X, Y and Z:

$X: \; ab(c+d)$ $\qquad$ $Y: \; a(bc+bd)$ $\qquad$ $Z: \; abc+abd$



- The transition behaviour of the processes is given by:

$$X: \alpha_{abc}(\{0\}) = [\alpha_{bc}(\{1\})] = [[\alpha_c(\{2\})]] = [[[\{31\}]]]$$
$$Y: \alpha_{abc}(\{0\}) = [\alpha_{bc}(\{1\})] = [[\alpha_c(\{21,22\})]] = [[[\{31\},\perp]]]$$
$$Z: \alpha_{abc}(\{0\}) = [\alpha_{bc}(\{11,12\})] = [[\alpha_c(\{21,22\})]] = [[[\{31\},\perp]]]$$

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.