Automata-theoretic Model of Idealized Algol with Passive Expressions
(In memory of John Reynolds, 1935-2013)

Uday S. Reddy

1University of Birmingham

John C. Reynolds, 1935-2013
Emmy Noether, 1882-1935

- According to Mac Lane, she “emphasized the importance of homomorphisms.”
- Within 10 years of her passing, category theory was born, taking “homomorphisms” as the primary elements, and formulating naturality as the preservation of homomorphisms.
Emmy Noether, 1882-1935

According to Mac Lane, she “emphasized the importance of homomorphisms.”

Within 10 years of her passing, category theory was born, taking “homomorphisms” as the primary elements, and formulating naturality as the preservation of homomorphisms.
Emmy Noether, 1882-1935

- According to Mac Lane, she “emphasized the importance of homomorphisms.”
- Within 10 years of her passing, category theory was born, taking “homomorphisms” as the primary elements, and formulating naturality as the preservation of homomorphisms.
John Reynolds emphasized the importance of logical relations, formulating relational parametricity as the preservation of logical relations.

Can we hope for a new theory within 10 years of his passing?

Do “logical relations” pose a challenge to the supremacy of “homomorphisms”?

Do they give us a better handle on the “mathematical reality?” At least as seen from the Computer Science point of view?

Or, perhaps more generally, are there aspects of mathematical phenomena hidden from our view which might be unveiled by understanding “logical relations”?
John Reynolds emphasized the importance of logical relations, formulating relational parametricity as the preservation of logical relations.

Can we hope for a new theory within 10 years of his passing?

Do “logical relations” pose a challenge to the supremacy of “homomorphisms”?

Do they give us a better handle on the “mathematical reality?” At least as seen from the Computer Science point of view?

Or, perhaps more generally, are there aspects of mathematical phenomena hidden from our view which might be unveiled by understanding “logical relations”? 
John Reynolds emphasized the importance of logical relations, formulating relational parametricity as the preservation of logical relations.

Can we hope for a new theory within 10 years of his passing?

Do “logical relations” pose a challenge to the supremacy of “homomorphisms”?

Do they give us a better handle on the “mathematical reality?” At least as seen from the Computer Science point of view?

Or, perhaps more generally, are there aspects of mathematical phenomena hidden from our view which might be unveiled by understanding “logical relations”? 
John Reynolds emphasized the importance of logical relations, formulating relational parametricity as the preservation of logical relations.

Can we hope for a new theory within 10 years of his passing?

Do “logical relations” pose a challenge to the supremacy of “homomorphisms”?

Do they give us a better handle on the “mathematical reality?” At least as seen from the Computer Science point of view?

Or, perhaps more generally, are there aspects of mathematical phenomena hidden from our view which might be unveiled by understanding “logical relations”?
John Reynolds emphasized the importance of logical relations, formulating relational parametricity as the preservation of logical relations.

Can we hope for a new theory within 10 years of his passing?

Do “logical relations” pose a challenge to the supremacy of “homomorphisms”?

Do they give us a better handle on the “mathematical reality?” At least as seen from the Computer Science point of view?

Or, perhaps more generally, are there aspects of mathematical phenomena hidden from our view which might be unveiled by understanding “logical relations”?
Logical relations

- **Logical relations** are relations compatible with structure, just like **homomorphisms** are functions preserving structure.

- Other names for logical relations in the literature:
  - Regular relation, Homogeneous relation, Compatible relation (algebra)
  - Congruence relation (algebra - for *equivalence* relations)
  - Covering relation (automata theory)
  - Simulation relation, Bisimulation relation (Milner, CCS & pi-calculus)
  - Refinement mapping (Abadi & Lamport, Dist. systems)

- **Relational parametricity** asks for “parametrically polymorphic” functions to preserve all logical relations.
Logical relations

- **Logical relations** are relations compatible with structure, just like **homomorphisms** are functions preserving structure.

- **Other names for logical relations** in the literature:
  - Regular relation, Homogeneous relation, Compatible relation (algebra)
  - Congruence relation (algebra - for *equivalence* relations)
  - Covering relation (automata theory)
  - Simulation relation, Bisimulation relation (Milner, CCS & pi-calculus)
  - Refinement mapping (Abadi & Lamport, Dist. systems)

- **Relational parametricity** asks for “parametrically polymorphic” functions to preserve all logical relations.

![Diagram](image-url)
Logical relations

- Logical relations are relations compatible with structure, just like homomorphisms are functions preserving structure.

- Other names for logical relations in the literature:
  - Regular relation, Homogeneous relation, Compatible relation (algebra)
  - Congruence relation (algebra - for equivalence relations)
  - Covering relation (automata theory)
  - Simulation relation, Bisimulation relation (Milner, CCS & pi-calculus)
  - Refinement mapping (Abadi & Lamport, Dist. systems)

- Relational parametricity asks for “parametrically polymorphic” functions to preserve all logical relations.
Between 1972-1981, Reynolds taught a graduate course on programming at Syracuse.

The material he developed for the course is published as *The Craft of Programming*. (Prentice-Hall International Series in Computer Science, 1981. Now available on Reynolds’s home page.)
The Craft of Programming (contd)

Contains a wealth of information about:

- what imperative programming means,
- how to develop imperative programs rigorously,
- the type structure of imperative programs,
- reasoning principles (both practical, e.g., arrays, and abstract, e.g., Specification Logic),
- how to reason about data abstraction (or information hiding).

Separation Logic, 2000, may be seen as a continuation of this body of work.
Contains a wealth of information about:

- what imperative programming \textit{means},
- how to \textit{develop} imperative programs rigorously,
- the \textit{type structure} of imperative programs,
- reasoning principles (both \textit{practical}, e.g., arrays, and \textit{abstract}, e.g., Specification Logic),
- how to reason about \textit{data abstraction} (or \textit{information hiding}).

\textbf{Separation Logic}, 2000, may be seen as a continuation of this body of work.
The Craft of Programming (contd)

- The Craft of Programming apparently gave rise to a series of landmark papers.
- 1978: Syntactic Control of Interference.
- 1984: Polymorphism is not set-theoretic.
The Craft of Programming (contd)

- The Craft of Programming apparently gave rise to a series of landmark papers.
- 1978: Syntactic Control of Interference.
- 1984: Polymorphism is not set-theoretic.
The Craft of Programming (contd)

- The Craft of Programming apparently gave rise to a series of landmark papers.
- 1978: Syntactic Control of Interference.
- 1984: Polymorphism is not set-theoretic.
The Craft of Programming (contd)

- The Craft of Programming apparently gave rise to a series of landmark papers.
- 1978: Syntactic Control of Interference.
- 1984: Polymorphism is not set-theoretic.
The Craft of Programming (contd)

- The Craft of Programming apparently gave rise to a series of landmark papers.
- 1978: Syntactic Control of Interference.
- 1984: Polymorphism is not set-theoretic.
The Craft of Programming (contd)

- The Craft of Programming apparently gave rise to a series of landmark papers.
- 1978: Syntactic Control of Interference.
- 1984: Polymorphism is not set-theoretic.
The Craft of Programming (contd)

- The Craft of Programming apparently gave rise to a series of landmark papers.
- 1978: Syntactic Control of Interference.
- 1984: Polymorphism is not set-theoretic.
The Craft of Programming (contd)

- 1983: *Types, Abstraction and Parametric Polymorphism*.

One might find it strange that he did not tie together the two most important papers in this series, even though they were clearly related. But he apparently knew that it had to be done and thought about it.

In the run up to the work published as *From Algol to Polymorphic Linear Lambda Calculus* (O’Hearn & Reynolds, 2000), he brought out handwritten notes from those times and explained to us why he couldn’t make it work. This had apparently to do with the tension between category theory and relational parametricity.

That tension continues to exist, and it is not always easy to resolve. It can be summarized by the slogan:

*Parametricity implies naturality.*

If it doesn’t imply, we have a problem.
The Craft of Programming (contd)


One might find it strange that he did not tie together the two most important papers in this series, even though they were clearly related. But he apparently knew that it had to be done and thought about it.

In the run up to the work published as From Algol to Polymorphic Linear Lambda Calculus (O’Hearn & Reynolds, 2000), he brought out handwritten notes from those times and explained to us why he couldn’t make it work. This had apparently to do with the tension between category theory and relational parametricity.

That tension continues to exist, and it is not always easy to resolve. It can be summarized by the slogan: 

*Parametricity implies naturality.*

If it doesn’t imply, we have a problem.


One might find it strange that he did not tie together the two most important papers in this series, even though they were clearly related. But he apparently knew that it had to be done and thought about it.

In the run up to the work published as From Algol to Polymorphic Linear Lambda Calculus (O’Hearn & Reynolds, 2000), he brought out handwritten notes from those times and explained to us why he couldn’t make it work. This had apparently to do with the tension between category theory and relational parametricity.

That tension continues to exist, and it is not always easy to resolve. It can be summarized by the slogan: *Parametricity implies naturality.*

If it doesn’t imply, we have a problem.
The Craft of Programming (contd)


One might find it strange that he did not tie together the two most important papers in this series, even though they were clearly related. But he apparently knew that it had to be done and thought about it.

In the run up to the work published as From Algol to Polymorphic Linear Lambda Calculus (O’Hearn & Reynolds, 2000), he brought out handwritten notes from those times and explained to us why he couldn’t make it work. This had apparently to do with the tension between category theory and relational parametricity.

That tension continues to exist, and it is not always easy to resolve. It can be summarized by the slogan: *Parametricity implies naturality.*

If it doesn’t imply, we have a problem.
Remarkable facts about Algol:

- Algol is a (call-by-name) typed lambda calculus.
- The types of Algol are functors (CPO-valued presheaves).
- The store is manipulated by Algol as if it is an automaton.

Considerable deviation from Christopher Strachey, whose views can be summarized as:

- Algol procedures are “complex” operational mechanisms.
- The types of Algol are sets (domains).
- The store is a set of locations.
The Essence of Algol

- Remarkable facts about Algol:
  - Algol is a (call-by-name) typed lambda calculus.
  - The types of Algol are functors (CPO-valued presheaves).
  - The store is manipulated by Algol as if it is an automaton.
- Considerable deviation from Christopher Strachey, whose views can be summarized as:
  - Algol procedures are “complex” operational mechanisms.
  - The types of Algol are sets (domains).
  - The store is a set of locations.
Strachey was attempting to reduce imperative programming to functional programming.

Functional programming is classical set theory (with some twists for divergence).

Imperative programming is intuitionistic topos theory.

This difference is now being increasingly appreciated, via the use of Kripke logical relations (Pitts, Stark, Appel & McAllester, Ahmed, Dreyer, Birkedal,...)

We have only begun to scratch the surface of the deep mysteries that underlie the theory of imperative programming.
Please join us in this effort!
Section 2

Introduction
The challenge for denotational semantics

- How to capture the intensional aspects of computations in an extensional model?

**Example of extensionality:**

\[ \text{gv}(x) \implies (x := !x + 1; x := !x + 1) \equiv (x := !x + 2) \]

**Intensional models**

- Object spaces [Reddy, O’Hearn, McCusker]
- Action traces [Hoare, Brookes]
- Game semantics [Abramsky, McCusker, Honda, Murawski]

Distinguish between the two sides of the equivalence.

\[ \text{read}_x(2), \text{write}_x(3), \text{read}_x(3), \text{write}_x(4) \]
\[ \text{read}_x(2), \text{write}_x(4) \]

We are after **extensional** models.

**Another example of extensionality:**

\[ \text{stack}(s) \implies (s.push(v); s.pop) \equiv \text{skip} \]
The challenge for denotational semantics

- How to capture the intensional aspects of computations in an extensional model?

**Example of extensionality:**

\[
gv(x) \quad \implies \quad (x := !x + 1; x := !x + 1) \equiv (x := !x + 2)
\]

- Intensional models
  - Object spaces [Reddy, O’Hearn, McCusker]
  - Action traces [Hoare, Brookes]
  - Game semantics [Abramsky, McCusker, Honda, Murawski]
  
  distinguish between the two sides of the equivalence.

\[
\text{read}_x(2), \text{write}_x(3), \text{read}_x(3), \text{write}_x(4) \\
\text{read}_x(2), \text{write}_x(4)
\]

- We are after extensional models.

**Another example of extensionality:**

\[
stack(s) \quad \implies \quad (s.push(v); s.pop) \equiv \text{skip}
\]
The challenge for denotational semantics

- How to capture the intensional aspects of computations in an extensional model?

**Example of extensionality:**

\[
gv(x) \implies (x := !x + 1; x := !x + 1) \equiv (x := !x + 2)
\]

- Intensional models
  - Object spaces [Reddy, O’Hearn, McCusker]
  - Action traces [Hoare, Brookes]
  - Game semantics [Abramsky, McCusker, Honda, Murawski]

Intensional models distinguish between the two sides of the equivalence.

\[
\text{read}_x(2), \text{write}_x(3), \text{read}_x(3), \text{write}_x(4)
\]

\[
\text{read}_x(2), \text{write}_x(4)
\]

- We are after extensional models.

- Another example of extensionality:

\[
\text{stack}(s) \implies (s.push(v); s.pop) \equiv \text{skip}
\]
The challenge for denotational semantics

- How to capture the intensional aspects of computations in an extensional model?

- **Example of extensionality:**

  \[ \text{gv}(x) \implies (x := !x + 1; x := !x + 1) \equiv (x := !x + 2) \]

- Intensional models
  - Object spaces [Reddy, O’Hearn, McCusker]
  - Action traces [Hoare, Brookes]
  - Game semantics [Abramsky, McCusker, Honda, Murawski]

  distinguish between the two sides of the equivalence.

  \[
  \begin{align*}
  &\text{read}_x(2), \text{write}_x(3), \text{read}_x(3), \text{write}_x(4) \\
  &\text{read}_x(2), \text{write}_x(4)
  \end{align*}
  \]

- We are after **extensional** models.

- Another example of extensionality:

  \[ \text{stack}(s) \implies (s.\text{push}(v); s.\text{pop}) \equiv \text{skip} \]
The challenge for denotational semantics

- How to capture the intensional aspects of computations in an extensional model?

**Example of extensionality:**

\[ gv(x) \implies (x := !x + 1; x := !x + 1) \equiv (x := !x + 2) \]

- Intensional models
  - Object spaces [Reddy, O’Hearn, McCusker]
  - Action traces [Hoare, Brookes]
  - Game semantics [Abramsky, McCusker, Honda, Murawski]

They distinguish between the two sides of the equivalence.

\[
\begin{align*}
\text{read}_x(2), \text{write}_x(3), \text{read}_x(3), \text{write}_x(4) \\
\text{read}_x(2), \text{write}_x(4)
\end{align*}
\]

- We are after **extensional models**.

**Another example of extensionality:**

\[ stack(s) \implies (s.push(v); s.pop) \equiv \text{skip} \]
Naive extensional models have “junk”

- Command snapback (with divergence):

  \[
  \text{try : com} \rightarrow \text{com} \\
  \text{try } c = \lambda s. \begin{cases} 
  s, & \text{if } c(s) \neq \bot \\
  \bot, & \text{if } c(s) = \bot
  \end{cases}
  \]

  State changes should be irreversible.

- Expression snapback:

  \[
  \text{do_result : com} \times \text{exp} \rightarrow \text{exp} \\
  \text{do } c \text{ result } e = \lambda s. e(c(s))
  \]

  Expressions should only read the state (passivity).

- Intensional models can eliminate such “junk” relatively easily.

- Eliminating “junk” in extensional models involves inventing mathematical structure.
Naive extensional models have “junk”

- **Command snapback** (with divergence):

  \[
  \text{try : com} \rightarrow \text{com} \\
  \text{try } c = \lambda s. \begin{cases} 
  s, & \text{if } c(s) \neq \bot \\
  \bot, & \text{if } c(s) = \bot 
  \end{cases}
  \]

  State changes should be **irreversible**.

- **Expression snapback**:

  \[
  \text{do_result : com} \times \text{exp} \rightarrow \text{exp} \\
  \text{do } c \text{ result } e = \lambda s. e(c(s))
  \]

  Expressions should only read the state (**passivity**).

- Intensional models can eliminate such “junk” relatively easily.

- Eliminating “junk” in extensional models involves inventing mathematical structure.
Naive extensional models have “junk”

- **Command snapback** (with divergence):

  ```
  try : com → com
  try c = λs. \(\begin{cases} \text{s, if } c(s) \neq \bot \\
  \bot, \text{ if } c(s) = \bot \end{cases}\)
  ```

  State changes should be **irreversible**.

- **Expression snapback**:

  ```
  do_result_ : com × exp → exp
  do c result e = λs. e(c(s))
  ```

  Expressions should only read the state (**passivity**).

- Intensional models can eliminate such “junk” relatively easily.

- Eliminating “junk” in extensional models involves inventing mathematical structure.
Naive extensional models have “junk”

- **Command snapback** (with divergence):

  \[
  \text{try} : \text{com} \rightarrow \text{com} \\
  \text{try } c = \lambda s. \begin{cases} 
  s, & \text{if } c(s) \neq \bot \\
  \bot, & \text{if } c(s) = \bot
  \end{cases}
  \]

  State changes should be **irreversible**.

- **Expression snapback**:

  \[
  \text{do}\_\text{result}\_ : \text{com} \times \text{exp} \rightarrow \text{exp} \\
  \text{do } c \text{ result } e = \lambda s. e(c(s))
  \]

  Expressions should only read the state (**passivity**).

- Intensional models can eliminate such “junk” relatively easily.

- Eliminating “junk” in extensional models involves inventing mathematical structure.
Example equivalences

- **Irreversible state change** is involved in this equivalence. Suppose \( p : \text{com} \rightarrow \text{com} \).

\[
\begin{align*}
\text{new } x. \; &\text{let } inc = (x := !x + 1) \\
&\text{in } x := 0; \; p(inc); \text{if } !x > 0 \text{ then diverge} \\
\equiv &\; p(\text{diverge})
\end{align*}
\]

- The first command diverges if \( p \) runs its argument command and terminates if \( p \) ignores its argument. The second command has exactly the same effect.

- The command snapback would break this equivalence. If \( p = \text{try} \), then it can run the argument command and set the state back to the initial state.
Example equivalences

- **Irreversible state change** is involved in this equivalence. Suppose $p : \text{com} \rightarrow \text{com}$.

  \[
  \text{new } x. \ \text{let } inc = (x := !x + 1) \\
  \quad \text{in } x := 0; \ p(inc); \ \text{if } !x > 0 \ \text{then diverge} \\
  \equiv p(\text{diverge})
  \]

- The first command diverges if $p$ runs its argument command and terminates if $p$ ignores its argument. The second command has exactly the same effect.

- The **command snapback** would break this equivalence. If $p = \text{try}$, then it can run the argument command and set the state back to the initial state.
Example equivalences (contd)

- **Passivity** is involved in this equivalence (remember call-by-name):

  \[
  \text{if } !x = 0 \text{ then } f(!x) \text{ else } 2 \equiv \text{if } !x = 0 \text{ then } f(0) \text{ else } 2
  \]

  where \( x : \text{var} \) and \( f : \exp \rightarrow \exp \).

- Since \( f(!x) \) is a passive expression, it can only read \( x \) and it gets the value 0.

- **Expression snapback** would break this equivalence. Suppose:

  \[
  f = \lambda e. \text{do } x := !x + 1 \text{ result } e
  \]

  then \( f(!x) \) would have the effect of \( f(1) \).
Example equivalences (contd)

- **Passivity** is involved in this equivalence (remember call-by-name):

  \[
  \text{if } !x = 0 \text{ then } f(!x) \text{ else } 2 \equiv \text{if } !x = 0 \text{ then } f(0) \text{ else } 2
  \]

  where \( x : \text{var} \) and \( f : \text{exp} \to \text{exp} \).

- Since \( f(!x) \) is a passive expression, it can only read \( x \) and it gets the value 0.

- **Expression snapback** would break this equivalence. Suppose:

  \[
  f = \lambda e. \text{do } x := !x + 1 \text{ result } e
  \]

  then \( f(!x) \) would have the effect of \( f(1) \).
Reynolds model (Essence of Algol)

- Quite amazingly, Reynolds’s 1981 model had the right structure to eliminate the command snapback. (But this requires relational parametricity, which Reynolds didn’t have in 1981.)
- The types of Algol are functors parameterized by “store shapes”:
  - \( \text{Com}(X) = \) the collection of state transformations for stores of shape \( X \).
  - \( \text{Exp}(X) = \) the collection of state readers for stores of shape \( X \).
  - \( (F \Rightarrow G)(X) = \) the collection of procedures/functions for stores of shape \( X \), which can work at all future stores \( Y \) of \( X \).

\[
\forall f : Y \leftarrow X \text{Com}(Y) \rightarrow \text{Exp}(Y) \\
\int f : Y \leftarrow X \text{Com}(Y) \rightarrow \text{Exp}(Y)
\]

(Think of this is an intuitionistic function space.)
- \( \text{Var}(X) = \text{Exp}(X) \times [\text{Int} \rightarrow \text{Com}(X)] \), pairs of read and write operations for a variable.
Reynolds model (Essence of Algol)

- Quite amazingly, Reynolds’s 1981 model had the right structure to eliminate the command snapback. (But this requires relational parametricity, which Reynolds didn’t have in 1981.)

- The types of Algol are **functors** parameterized by “store shapes”:
  - $\text{COM}(X) =$ the collection of state transformations for stores of shape $X$.
  - $\text{EXP}(X) =$ the collection of state readers for stores of shape $X$.
  - $(F \Rightarrow G)(X) =$ the collection of procedures/functions for stores of shape $X$, which can work at all future stores $Y$ of $X$.

\[
\forall_{f:Y \leftarrow X} F(Y) \rightarrow G(Y) \quad \int_{f:Y \leftarrow X} F(Y) \rightarrow G(Y)
\]

(Take of this is an intuitionistic function space.)

- $\text{VAR}(X) = \text{EXP}(X) \times [\text{Int} \rightarrow \text{COM}(X)]$, pairs of read and write operations for a variable.
Reynolds model (Essence of Algol)

- Quite amazingly, Reynolds’s 1981 model had the right structure to eliminate the command snapback. (But this requires relational parametricity, which Reynolds didn’t have in 1981.)

- The types of Algol are **functors** parameterized by “store shapes”:
  - \( \text{Com}(X) = \) the collection of state transformations for stores of shape \( X \).
  - \( \text{Exp}(X) = \) the collection of state readers for stores of shape \( X \).
  - \( (F \Rightarrow G)(X) = \) the collection of procedures/functions for stores of shape \( X \), which can work at all future stores \( Y \) of \( X \).

  \[
  \forall f: Y \leftarrow X \, F(Y) \to G(Y) \quad \int_{f: Y \leftarrow X} F(Y) \to G(Y)
  \]

  (Think of this is an intuitionistic function space.)

- \( \text{Var}(X) = \text{Exp}(X) \times [\text{Int} \to \text{Com}(X)] \), pairs of read and write operations for a variable.
Reynolds model (Essence of Algol)

- Quite amazingly, Reynolds’s 1981 model had the right structure to eliminate the command snapback. (But this requires relational parametricity, which Reynolds didn’t have in 1981.)

- The types of Algol are **functors** parameterized by “store shapes”:
  - \( \text{Com}(X) = \) the collection of state transformations for stores of shape \( X \).
  - \( \text{Exp}(X) = \) the collection of state readers for stores of shape \( X \).
  - \( (F \Rightarrow G)(X) = \) the collection of procedures/functions for stores of shape \( X \), which can work at all future stores \( Y \) of \( X \).

\[
\forall_{f: Y \leftarrow X} F(Y) \rightarrow G(Y) \quad \int_{f: Y \leftarrow X} F(Y) \rightarrow G(Y)
\]

(Think of this is an intuitionistic function space.)

- \( \text{Var}(X) = \text{Exp}(X) \times [\text{Int} \rightarrow \text{Com}(X)] \), pairs of read and write operations for a variable.
Reynolds model (Essence of Algol)

- Quite amazingly, Reynolds’s 1981 model had the right structure to eliminate the command snapback. (But this requires relational parametricity, which Reynolds didn’t have in 1981.)

- The types of Algol are functors parameterized by “store shapes”:
  - \( \text{Com}(X) = \) the collection of state transformations for stores of shape \( X \).
  - \( \text{Exp}(X) = \) the collection of state readers for stores of shape \( X \).
  - \( (F \ightarrow G)(X) = \) the collection of procedures/functions for stores of shape \( X \), which can work at all future stores \( Y \) of \( X \).

\[
\forall_{f:Y \leftarrow X} F(Y) \rightarrow G(Y) \quad \int_{f:Y \leftarrow X} F(Y) \rightarrow G(Y)
\]

(Think of this is an intuitionistic function space.)

- \( \text{Var}(X) = \text{Exp}(X) \times [\text{Int} \rightarrow \text{Com}(X)] \), pairs of read and write operations for a variable.
Reynolds model (Essence of Algol)

- Quite amazingly, Reynolds’s 1981 model had the right structure to eliminate the command snapback. (But this requires relational parametricity, which Reynolds didn’t have in 1981.)
- The types of Algol are **functors** parameterized by “store shapes”:
  - $\text{COM}(X) =$ the collection of state transformations for stores of shape $X$.
  - $\text{EXP}(X) =$ the collection of state readers for stores of shape $X$.
  - $(F \Rightarrow G)(X) =$ the collection of procedures/functions for stores of shape $X$, which can work at all future stores $Y$ of $X$.

$$\forall f: Y \leftarrow X \; F(Y) \rightarrow G(Y) \quad \int_{f: Y \leftarrow X} F(Y) \rightarrow G(Y)$$

(Think of this is an intuitionistic function space.)
- $\text{VAR}(X) = \text{EXP}(X) \times [\text{Int} \rightarrow \text{COM}(X)]$, pairs of read and write operations for a variable.
What should a store be?

- **Idea 1**: A store is a collection of locations.
- **Idea 2**: A store can be abstracted to a set of states.
- **Idea 3**: A store should be abstracted to a set of states along with its possible state transformations.

Reynolds arrived at Idea 3 in 1981! But, perhaps, he didn’t have a strong reason to pursue it.

Oles produced a variant of the model using Idea 2. It became standard from then on.

However, the tension between category theory and relational parametricity, which exists with the Oles model, is not present in the Reynolds model. (This problem led me to reinvent it in 1998.)
What should a store be?

- **Idea 1**: A store is a collection of locations.
- **Idea 2**: A store can be abstracted to a set of states.
- **Idea 3**: A store should be abstracted to a set of states along with its possible state transformations.

Reynolds arrived at Idea 3 in 1981! But, perhaps, he didn’t have a strong reason to pursue it.

Oles produced a variant of the model using Idea 2. It became standard from then on.

However, the tension between category theory and relational parametricity, which exists with the Oles model, is not present in the Reynolds model. (This problem led me to reinvent it in 1998.)
What should a store be?

- **Idea 1**: A store is a collection of locations.
- **Idea 2**: A store can be abstracted to a set of states.
- **Idea 3**: A store should be abstracted to a set of states along with its possible state transformations.
- Reynolds arrived at Idea 3 in 1981! But, perhaps, he didn’t have a strong reason to pursue it.
- Oles produced a variant of the model using Idea 2. It became standard from then on.
- However, the tension between category theory and relational parametricity, which exists with the Oles model, is not present in the Reynolds model. (This problem led me to reinvent it in 1998.)
What should a store be?

- **Idea 1**: A store is a collection of locations.
- **Idea 2**: A store can be abstracted to a set of states.
- **Idea 3**: A store should be abstracted to a set of states along with its possible state transformations.

Reynolds arrived at Idea 3 in 1981! But, perhaps, he didn’t have a strong reason to pursue it.

Oles produced a variant of the model using Idea 2. It became standard from then on.

However, the tension between category theory and relational parametricity, which exists with the Oles model, is not present in the Reynolds model. (This problem led me to reinvent it in 1998.)
What should a store be?

➤ **Idea 1**: A store is a collection of locations.

➤ **Idea 2**: A store can be abstracted to a set of states.

➤ **Idea 3**: A store should be abstracted to a set of states along with its possible state transformations.

➤ Reynolds arrived at Idea 3 in 1981! But, perhaps, he didn’t have a strong reason to pursue it.

➤ Oles produced a variant of the model using Idea 2. It became standard from then on.

➤ However, the tension between category theory and relational parametricity, which exists with the Oles model, is not present in the Reynolds model. (This problem led me to reinvent it in 1998.)
What should a store be?

- **Idea 1**: A store is a collection of locations.
- **Idea 2**: A store can be abstracted to a set of states.
- **Idea 3**: A store should be abstracted to a set of states along with its possible state transformations.

Reynolds arrived at Idea 3 in 1981! But, perhaps, he didn’t have a strong reason to pursue it.

Oles produced a variant of the model using Idea 2. It became standard from then on.

However, the tension between category theory and relational parametricity, which exists with the Oles model, is not present in the Reynolds model. (This problem led me to reinvent it in 1998.)
What should a store be?

- **Idea 1**: A store is a collection of locations.
- **Idea 2**: A store can be abstracted to a set of states.
- **Idea 3**: A store should be abstracted to a set of states along with its possible state transformations.

Reynolds arrived at Idea 3 in 1981! But, perhaps, he didn’t have a strong reason to pursue it.

Oles produced a variant of the model using Idea 2. It became standard from then on.

However, the tension between category theory and relational parametricity, which exists with the Oles model, is not present in the Reynolds model. (This problem led me to reinvent it in 1998.)
Reynolds transformation monoids

- A store $X$ is represented as a tuple $(Q_X, T_X, \alpha_X, \text{read}_X)$

$(\text{Reynolds transformation monoid})$ where:

- $Q_X$ - a (small) set of states,
- $T_X$ - a monoid of state transformations $T_X \subseteq [Q_X \rightarrow Q_X]$,
- $\alpha_X : T_X \rightarrow [Q_X \rightarrow Q_X]$ - the implicit monoid action,
- $\text{read}_X : [Q_X \rightarrow T_X] \rightarrow T_X$ - called “diagonalization”:

$$\text{read}_X p = \lambda x. p x x = \lambda x. \alpha_X(p x) x$$

allows a state transformation to be dependent on the initial state.
For example,

$$\text{cond}_X b c_1 c_2 = \text{read}_X \lambda s. \text{if } b(s) \neq 0 \text{ then } c_1 \text{ else } c_2$$

- Note: Transformation monoids in algebraic automata theory [Eilenberg, 1974] are triples $(Q_X, T_X, \alpha_X)$. 
Logical relations for RTM’s

- A logical relation $R : (Q_X, T_X) \leftrightarrow (Q_{X'}, T_{X'})$ is a pair $(R_q, R_t)$ where

$$
\begin{array}{c}
\xymatrix{ X 
\ar[r]^R 
\ar[d] & X' \ar[d] \\
Q_X 
\ar[r]_{R_q} & Q_{X'} \\
T_X 
\ar[r]_{R_t} & T_{X'} }
\end{array}
$$

- $R_q : Q_X \leftrightarrow Q_{X'}$ is a relation, and
- $R_t : T_X \leftrightarrow T_{X'}$ is a logical relation of monoids, such that

- $\alpha_X [R_t \rightarrow [R_q \rightarrow R_q]] \alpha_{X'}$, and
- $\text{read}_X [[R_q \rightarrow R_t] \rightarrow R_t] \text{read}_{X'}$. 

Morphisms for RTM’s

▶ A homomorphism \( f : (Q_X, T_X) \to (Q_Y, T_Y) \) is a pair \((f_q, f_t)\)

\[
\begin{pmatrix}
Y \\
\uparrow f \\
X
\end{pmatrix}
= \begin{pmatrix}
Q_Y & T_Y \\
\downarrow f_q & \quad \uparrow f_t \\
Q_X & T_X
\end{pmatrix}
\]

where

▶ \( f_q : Q_Y \to Q_X \) is a function, and
▶ \( f_t : T_X \to T_Y \) is a homomorphism of monoids,

such that \( (\langle f_q \rangle^\sim, \langle f_t \rangle) \) is a logical relation of RTM’s.

▶ Note that \( f_q \) and \( f_t \) run in opposite directions.
(Mixed variance)

▶ \( \langle f \rangle \) means the function graph (a function treated as a relation). \( R^\sim \) means the converse relation.
Morphisms for RTM’s

- A homomorphism $f : (Q_X, T_X) \rightarrow (Q_Y, T_Y)$ is a pair $(f_q, f_t)$

\[
\begin{array}{c}
Y \\
\downarrow^{f} \\
X
\end{array}
\begin{array}{c}
\begin{pmatrix}
Q_Y \\
\downarrow^{f_q} \\
Q_X
\end{pmatrix} \\
\downarrow^{f_t} \\
\begin{pmatrix}
T_Y \\
\uparrow^{f_t} \\
T_X
\end{pmatrix}
\end{array}
\]

where

- $f_q : Q_Y \rightarrow Q_X$ is a function, and
- $f_t : T_X \rightarrow T_Y$ is a homomorphism of monoids,

such that $(\langle f_q \rangle^\sim, \langle f_t \rangle)$ is a logical relation of RTM’s.

- Note that $f_q$ and $f_t$ run in opposite directions.

(Mixed variance)

- $\langle f \rangle$ means the function graph (a function treated as a relation). $R^\sim$ means the converse relation.
Morphisms for RTM’s

- A homomorphism \( f : (Q_X, T_X) \rightarrow (Q_Y, T_Y) \) is a pair \((f_q, f_t)\)

\[
\begin{align*}
\begin{pmatrix} Q_Y & T_Y \\ Q_X & T_X \end{pmatrix}^f &= \begin{pmatrix} f_q \\ f_t \end{pmatrix}
\end{align*}
\]

where
- \( f_q : Q_Y \rightarrow Q_X \) is a function, and
- \( f_t : T_X \rightarrow T_Y \) is a homomorphism of monoids,

such that \((\langle f_q \rangle^\frown, \langle f_t \rangle^\frown)\) is a logical relation of RTM’s.

- Note that \( f_q \) and \( f_t \) run in opposite directions.
  (Mixed variance)

- \( \langle f \rangle \) means the function graph (a function treated as a relation). \( R^\frown \) means the converse relation.
Interpretation of Algol types

- Algol types are now functors of type $\text{RTM} \rightarrow \text{Set}$ (ignoring divergence):

  \[
  \text{COM}(X) = \mathcal{T}_X \\
  \text{EXP}(X) = [Q_X \rightarrow \text{Int}] \\
  (F \Rightarrow G)(X) = \forall f: Y \leftarrow X F(Y) \rightarrow G(Y)
  \]

  This model does not have command snapback, i.e., models irreversible state change.

- **Fact:** $\text{Hom} (\text{COM} \rightarrow \text{COM}) \cong \mathbb{N}$, representable by

  $\lambda c. \text{skip}, \lambda c. c, \lambda c. (c; c), \ldots$

- But it has expression snapback (does not model passivity).
Interpretation of Algol types

- Algol types are now functors of type $\text{RTM} \to \text{Set}$ (ignoring divergence):

\[
\begin{align*}
\text{COM}(X) &= \mathcal{T}_X \\
\text{EXP}(X) &= [Q_X \to \text{Int}] \\
(F \Rightarrow G)(X) &= \forall f:Y \leftarrow X F(Y) \to G(Y)
\end{align*}
\]

This model does not have command snapback, i.e., models irreversible state change.

- Fact: $\text{Hom} (\text{COM} \to \text{COM}) \cong \mathbb{N}$, representable by

\[
\lambda c. \text{skip}, \lambda c. c, \lambda c. (c; c), \ldots
\]

- But it has expression snapback (does not model passivity).
Interpretation of Algol types

- Algol types are now functors of type $\text{RTM} \rightarrow \text{Set}$ (ignoring divergence):

  $\text{COM}(X) = T_X$

  $\text{EXP}(X) = [Q_X \rightarrow \text{Int}]$

  $(F \Rightarrow G)(X) = \forall f : Y \leftarrow X F(Y) \rightarrow G(Y)$

  This model does not have command snapback, i.e., models irreversible state change.

- **Fact:** $\text{Hom}(\text{COM} \rightarrow \text{COM}) \cong \mathbb{N}$, representable by

  $\lambda c. \text{skip}, \lambda c. c, \lambda c. (c; c), \ldots$

- But it has expression snapback (does not model passivity).
Section 3

Passivity
Passivity

- We have accumulated various pieces of knowledge about passivity over the years:
  - from Syntactic Control of Interference [Reynolds, Reddy, McCusker, O’Hearn & Tennent]
  - from Specification Logic [Reynolds, O’Hearn & Tennent]
- We can formulate a set of “requirements” for the semantic model based on this knowledge.
  - A passive computation only reads the state (does not change it).
  - For every semantic type $F$, we can envisage that there will be a subset of its values that are passive, which should form a passive subobject $\wp F \hookrightarrow F$. 
Passivity

- We have accumulated various pieces of knowledge about passivity over the years:
  - from Syntactic Control of Interference [Reynolds, Reddy, McCusker, O’Hearn & Tennent]
  - from Specification Logic [Reynolds, O’Hearn & Tennent]
- We can formulate a set of “requirements” for the semantic model based on this knowledge.
- A passive computation only reads the state (does not change it).
- For every semantic type $F$, we can envisage that there will be a subset of its values that are passive, which should form a passive subobject $\wp F \hookrightarrow F$. 
The passive subobject constructor should satisfy

$$\wp P \cong P$$

for passive functors \(P\)

$$\wp (F \times G) \cong \wp F \times \wp G$$

$$F \Rightarrow P \cong \wp F \Rightarrow P$$

for passive functors \(P\)

For example:

$$\wp \text{Exp} \cong \text{Exp}$$

$$\wp \text{Com} \cong \mathbf{1}$$

$$\wp (\text{Com} \times \text{Exp}) \cong \wp \text{Com} \times \wp \text{Exp} \cong \mathbf{1} \times \text{Exp} \cong \text{Exp}$$

$$\text{Com} \Rightarrow \text{Exp} \cong \wp \text{Com} \Rightarrow \text{Exp} \cong \mathbf{1} \Rightarrow \text{Exp} \cong \text{Exp}$$

The last isomorphism implies that all “functions” of type \(\text{Com} \Rightarrow \text{Exp}\) are constant functions.

So do \texttt{C result} \(E\) does not have an interpretation in a model with passivity.
The passive subobject constructor should satisfy

\[ \wp P \cong P \quad \text{for passive functors } P \]
\[ \wp (F \times G) \cong \wp F \times \wp G \]
\[ F \Rightarrow P \cong \wp F \Rightarrow P \quad \text{for passive functors } P \]

For example:

\[ \wp \text{EXP} \cong \text{EXP} \]
\[ \wp \text{COM} \cong 1 \]
\[ \wp (\text{COM} \times \text{EXP}) \cong \wp \text{COM} \times \wp \text{EXP} \cong 1 \times \text{EXP} \cong \text{EXP} \]
\[ \text{COM} \Rightarrow \text{EXP} \cong \wp \text{COM} \Rightarrow \text{EXP} \cong 1 \Rightarrow \text{EXP} \cong \text{EXP} \]

The last isomorphism implies that all “functions” of type \( \text{COM} \Rightarrow \text{EXP} \) are constant functions.

So do \texttt{C result} \( E \) does not have an interpretation in a model with passivity.
The passive subobject constructor should satisfy

\[ \varnothing P \cong P \quad \text{for passive functors } P \]
\[ \varnothing (F \times G) \cong \varnothing F \times \varnothing G \]
\[ F \Rightarrow P \cong \varnothing F \Rightarrow P \quad \text{for passive functors } P \]

For example:

\[ \varnothing \text{Exp} \cong \text{Exp} \]
\[ \varnothing \text{Com} \cong 1 \]
\[ \varnothing (\text{Com} \times \text{Exp}) \cong \varnothing \text{Com} \times \varnothing \text{Exp} \cong 1 \times \text{Exp} \cong \text{Exp} \]
\[ \text{Com} \Rightarrow \text{Exp} \cong \varnothing \text{Com} \Rightarrow \text{Exp} \cong 1 \Rightarrow \text{Exp} \cong \text{Exp} \]

The last isomorphism implies that all “functions” of type \( \text{Com} \Rightarrow \text{Exp} \) are constant functions.

So do \textit{C result} \( E \) does not have an interpretation in a model with passivity.
In the object spaces and games models, the passive types form a subcategory that is reflective and coreflective.

In Tennent’s model of Specification Logic, they form a bireflective subcategory.

McCusker has formulated a categorical model for the reflective-coreflective setting, and proved coherence for Syntactic Control of Interference.
Passivity by logical relations

- Suppose \( d \in F(X) \) is a value. When can we say that \( d \) is “passive”?

- The store \( X = (Q_X, T_X) \) allows transformations \( T_X \). But if \( d \) is passive, \( d \) should be independent of \( T_X \).

- For any logical relation \( R : X \leftrightarrow X \) of the form

\[
R = (\Delta_{Q_X} : Q_X \leftrightarrow Q_X, \ R_t : T_X \leftrightarrow T_X)
\]

(where the state set part fixes a state and the transformation part allows arbitrary substitution): it should be the case that \( d \ [F(R)] \ d \).

- **Intuition**: if we substitute one state transformation by another, \( d \) remains unaffected.

- Call such relations **transformer relations**.
Passivity by logical relations

- Suppose $d \in F(X)$ is a value. When can we say that $d$ is “passive”?
- The store $X = (Q_X, T_X)$ allows transformations $T_X$. But if $d$ is passive, $d$ should be independent of $T_X$.
- For any logical relation $R : X \leftrightarrow X$ of the form

  $$R = (\Delta_{Q_X} : Q_X \leftrightarrow Q_X, \ R_t : T_X \leftrightarrow T_X)$$

  (where the state set part fixes a state and the transformation part allows arbitrary substitution):
  it should be the case that $d \ [F(R)] \ d$.
- **Intuition**: if we substitute one state transformation by another, $d$ remains unaffected.
- Call such relations transformer relations.
Passivity by logical relations

- Suppose $d \in F(X)$ is a value. When can we say that $d$ is “passive”?
- The store $X = (Q_X, T_X)$ allows transformations $T_X$. But if $d$ is passive, $d$ should be independent of $T_X$.
- For any logical relation $R : X \leftrightarrow X$ of the form

\[ R = (\Delta_{Q_X} : Q_X \leftrightarrow Q_X, R_t : T_X \leftrightarrow T_X) \]

(where the state set part fixes a state and the transformation part allows arbitrary substitution): it should be the case that $d \ [F(R)] \ d$.

- **Intuition**: if we substitute one state transformation by another, $d$ remains unaffected.
- Call such relations transformer relations.
Definition: A value $d \in F(X)$ is **passive** if, for all transformer relations $R : X \leftrightarrow X$, we have $d \left[ F(R) \right] d$.

Definition: A functor $F$ is passive if, for all transformer relations $R : X \leftrightarrow X$, $F(R) = \Delta_{F(X)}$.

(Cf. [Benton et al, 2006] who use a similar-looking formulation for the semantics of read-only effects in effect systems.)

Example: All values of type $\text{Exp}(X)$ are passive.

Example: The only passive value of type $\text{Com}(X)$ is $\text{skip}_X$.

Example: $\text{Exp}$ is passive functor.

Example: $\text{Com}$ is not a passive functor.
Definition: A value $d \in F(X)$ is passive if, for all transformer relations $R : X \leftrightarrow X$, we have $d[F(R)]d$.

Definition: A functor $F$ is passive if, for all transformer relations $R : X \leftrightarrow X$, $F(R) = \Delta_{F(X)}$.

(Cf. [Benton et al, 2006] who use a similar-looking formulation for the semantics of read-only effects in effect systems.)

Example: All values of type $\text{Exp}(X)$ are passive.

Example: The only passive value of type $\text{Com}(X)$ is $\text{skip}_X$.

Example: $\text{Exp}$ is passive functor.

Example: $\text{Com}$ is not a passive functor.
Definition: A value \( d \in F(X) \) is **passive** if, for all transformer relations \( R : X \leftrightarrow X \), we have \( d [F(R)] d \).

Definition: A functor \( F \) is **passive** if, for all transformer relations \( R : X \leftrightarrow X \), \( F(R) = \Delta_{F(X)} \).
(Cf. [Benton et al, 2006] who use a similar-looking formulation for the semantics of read-only effects in effect systems.)

Example: All values of type \( \text{Exp}(X) \) are passive.

Example: The only passive value of type \( \text{Com}(X) \) is \( \text{skip}_X \).

Example: \( \text{Exp} \) is passive functor.

Example: \( \text{Com} \) is not a passive functor.
Definition: A value $d \in F(X)$ is passive if, for all transformer relations $R: X \leftrightarrow X$, we have $d [F(R)] d$.

Definition: A functor $F$ is passive if, for all transformer relations $R: X \leftrightarrow X$, $F(R) = \Delta_{F(X)}$.
(Cf. [Benton et al, 2006] who use a similar-looking formulation for the semantics of read-only effects in effect systems.)

Example: All values of type $\text{EXP}(X)$ are passive.

Example: The only passive value of type $\text{COM}(X)$ is $\text{skip}_X$.

Example: $\text{EXP}$ is passive functor.

Example: $\text{COM}$ is not a passive functor.
Are there enough transformer relations?

- Our intuition is that the $R_t$ component of a transformer relation can arbitrarily vary the state transformations.
- But this is not true in RTM, the reflexive graph category of Reynolds transformation monoids.

$$\alpha_X : \mathcal{T}_X \hookrightarrow [\mathcal{Q}_X \rightarrow \mathcal{Q}_X]$$

A logical relation of RTM’s should be compatible with $\alpha_X$, which forces

$$R_t \subseteq \Delta_{[\mathcal{Q}_X \rightarrow \mathcal{Q}_X]}$$

- Do we need the $\alpha_X$ component in the structure?
- The answer is NO. It is not used anywhere in the semantics.
- **Intuition:** States in imperative programs are “abstract.” They are not observable externally, but only by their effect on other commands. Therefore, the $\text{read}_X$ operation is enough.
Are there enough transformer relations?

- Our intuition is that the $R_t$ component of a transformer relation can arbitrarily vary the state transformations.
- But this is not true in **RTM**, the reflexive graph category of Reynolds transformation monoids.

$$
\alpha_X : \mathcal{T}_X \hookrightarrow [\mathcal{Q}_X \rightarrow \mathcal{Q}_X]
$$

A logical relation of RTM’s should be compatible with $\alpha_X$, which forces

$$
R_t \subseteq \Delta[\mathcal{Q}_X \rightarrow \mathcal{Q}_X]
$$

- Do we need the $\alpha_X$ component in the structure?
- The answer is NO. It is not used anywhere in the semantics.
- **Intuition**: States in imperative programs are “abstract.” They are not observable externally, but only by their effect on other commands. Therefore, the $\text{read}_X$ operation is enough.
Are there enough transformer relations?

- Our intuition is that the $R_t$ component of a transformer relation can arbitrarily vary the state transformations.
- But this is not true in RTM, the reflexive graph category of Reynolds transformation monoids.

\[ \alpha_X : T_X \hookrightarrow \{ Q_X \rightarrow Q_X \} \]

A logical relation of RTM’s should be compatible with $\alpha_X$, which forces

\[ R_t \subseteq \Delta[Q_X \rightarrow Q_X] \]

- Do we need the $\alpha_X$ component in the structure?
- The answer is NO. It is not used anywhere in the semantics.

- **Intuition**: States in imperative programs are “abstract.” They are not observable externally, but only by their effect on other commands. Therefore, the \texttt{read}_X operation is enough.
Are there enough transformer relations?

- Our intuition is that the $R_t$ component of a transformer relation can arbitrarily vary the state transformations.
- But this is not true in RTM, the reflexive graph category of Reynolds transformation monoids.

$$\alpha_X : T_X \hookrightarrow [Q_X \rightarrow Q_X]$$

A logical relation of RTM’s should be compatible with $\alpha_X$, which forces

$$R_t \subseteq \Delta[Q_X \rightarrow Q_X]$$

- Do we need the $\alpha_X$ component in the structure?
- The answer is NO. It is not used anywhere in the semantics.

**Intuition:** States in imperative programs are “abstract.” They are not observable externally, but only by their effect on other commands. Therefore, the $\text{read}_X$ operation is enough.
Definition: A Reynolds monoid is a triple \( X = (Q_X, T_X, \text{read}_X) \) where \( \text{read}_X(\rho) = \lambda x. \rho x x \).

In the reflexive graph category \( \mathbf{RM} \), we have a transformer relation \( \xi_X : X \leftrightarrow X \):

\[
\xi_X = (\Delta_{Q_X}, \{ (a, 1_X) \mid a \in T_X \})
\]

This allows us to substitute any transformation by the do-nothing transformation.
Passive subobjects 1

- For every store $X = (Q_X, T_X)$ there is a **passive store** $X_0 = (Q_X, 0_X)$ where $0_X = \{1_X\}$ is the trivial monoid.
- The passive store $X_0$ does not allow any stage changes.
- We have a monomorphism $p_X : X_0 \hookrightarrow X$ which sends $1_X$ to $1_X$.

\[
\begin{array}{c}
X \\
\uparrow p_X \\
X_0
\end{array}
= \left(
\begin{array}{c}
Q_X \\
\downarrow \text{id}_{Q_X} \\
Q_X
\end{array}
, \begin{array}{c}
T_X \\
\uparrow (p_X)_t \\
0_X
\end{array}
\right)
\]

- **Lemma**: A value $d \in F(X)$ is passive if and only if it “comes from” the passive store $X_0$, i.e., there is a value $d_0 \in F(X_0)$ such that $Fp_X : d_0 \mapsto d$. 
Passive subobjects 1

- For every store $X = (Q_X, T_X)$ there is a passive store $X_0 = (Q_X, 0_X)$ where $0_X = \{ 1_X \}$ is the trivial monoid.
- The passive store $X_0$ does not allow any stage changes.
- We have a monomorphism $p_X : X_0 \hookrightarrow X$ which sends $1_X$ to $1_X$.

\[
\begin{array}{c}
X \\
p_X \\
\uparrow \\
X_0
\end{array} = \begin{pmatrix}
Q_X \\
\downarrow \text{id}_{Q_X} \\
\uparrow (p_X)_t \\
Q_X \\
0_X
\end{pmatrix},
\]

- **Lemma**: A value $d \in F(X)$ is passive if and only if it "comes from" the passive store $X_0$, i.e., there is a value $d_0 \in F(X_0)$ such that $Fp_X : d_0 \leftrightarrow d$. 
Definition: The passive subobject $\varnothing F$ of $F$ is defined by

$$(\varnothing F)(X) = F(X_0)$$

Example: $(\varnothing \text{EXP})(X) = \text{EXP}(X_0) = \text{EXP}(X)$. So, $\varnothing \text{EXP} \cong \text{EXP}$.

Example: $(\varnothing \text{COM})(X) = \text{COM}(X_0) = \{1_X\}$. So, $\varnothing \text{COM} \cong 1$.

Example: $\varnothing(F \times G)(X) = (F \times G)(X_0) = F(X_0) \times G(X_0) = \varnothing F(X) \times \varnothing G(X) = (\varnothing F \times \varnothing G)(X)$. So, $\varnothing(F \times G) \cong \varnothing F \times \varnothing G$. 
Passive subobjects 2

**Definition:** The passive subobject $\wp F$ of $F$ is defined by

$$(\wp F)(X) = F(X_0)$$

**Example:** $(\wp \text{Exp})(X) = \text{Exp}(X_0) = \text{Exp}(X)$. So, $\wp \text{Exp} \cong \text{Exp}$.

**Example:** $(\wp \text{Com})(X) = \text{Com}(X_0) = \{1_X\}$. So, $\wp \text{Com} \cong 1$.

**Example:** $\wp (F \times G)(X) = (F \times G)(X_0) = F(X_0) \times G(X_0) = \wp F(X) \times \wp G(X) = (\wp F \times \wp G)(X)$. So, $\wp (F \times G) \cong \wp F \times \wp G$. 
Passivity retractions

- The isomorphism $F \Rightarrow P \cong \emptyset F \Rightarrow P$ is tricky.
- In the absence of divergence, we can note the following.
- The passivity monomorphisms have retractions $r_X$:

$$x_0 \xrightarrow{p_X} X \xrightarrow{r_X} x_0 = id_{x_0}$$

The state set part of $r_X$ is $id_{Q_X}$.
The transformation part sends $a \mapsto 1_X$.
- Using these retractions (called “passification” morphisms), we can show

$$FX \xrightarrow{Fr_X} FX_0 = (\emptyset F)X \xrightarrow{(t_0)_X} PX$$

- This makes the subcategory of passive functors a reflective subcategory.
(It is also a bireflective subcategory [Freyd et al. 1995].)
Passivity retractions

- The isomorphism $F \Rightarrow P \cong \wp F \Rightarrow P$ is tricky.
- In the absence of divergence, we can note the following.
- The passivity monomorphisms have retractions $r_X$:

$$
X_0 \xrightarrow{p_X} X \xrightarrow{r_X} X_0 = \text{id}_{X_0}
$$

The state set part of $r_X$ is $\text{id}_{Q_X}$.
The transformation part sends $a \mapsto 1_X$.

- Using these retractions (called “passification” morphisms), we can show

$$
FX_0 = (\wp F)X \xrightarrow{(t_0)_X} PX
$$

- This makes the subcategory of passive functors a reflective subcategory.
  (It is also a bireflective subcategory [Freyd et al. 1995].)
Passivity retractions

- The isomorphism $F \Rightarrow P \cong \emptyset F \Rightarrow P$ is tricky.
- In the absence of divergence, we can note the following.
- The passivity monomorphisms have retractions $r_X$:

$$
X_0 \xrightarrow{p_X} X \xrightarrow{r_X} X_0 = \text{id}_{X_0}
$$

The state set part of $r_X$ is $\text{id}_{Q_X}$.
The transformation part sends $a \mapsto 1_X$.
- Using these retractions (called "passification" morphisms), we can show

$$
FX_0 = (\emptyset F)X \xrightarrow{(t_0)_X} PX
$$

- This makes the subcategory of passive functors a reflective subcategory.
(It is also a bireflective subcategory [Freyd et al. 1995].)
In the presence of divergence

- Divergence is quite a bit more subtle.
- The passive store $X_0$ has all approximations of the do-nothing transformation $T_{X_0} = \{ a \mid a \subseteq 1_X \}$.
- The transformer relation $\xi_X : X \leftrightarrow X$ should be modified to:
  \[ \xi_X = \{ (a, a_0) \mid a_0 \subseteq a \cap 1_X \} \]
- We have a relation preservation square:

\[
\begin{array}{cc}
  FX & \xrightarrow{t_X} & PX \\
  F\xi_X & & \downarrow P\xi_X = \Delta_{PX} \\
  FX & \xrightarrow{t_X} & PX
\end{array}
\]

- The “range” of $F\xi_X$ is $F(X_0) = (\emptyset F)(X)$, this still implies that morphisms $t : F \to P$ are uniquely determined by morphisms $t_0 : \emptyset F \to P$.
- We have a natural injection $\text{Hom}(F, P) \hookrightarrow \text{Hom}(\emptyset F, P)$ rather than a bijection.
In the presence of divergence

- Divergence is quite a bit more subtle.
- The passive store $X_0$ has all approximations of the do-nothing transformation $\mathcal{T}_{X_0} = \{ a \mid a \subseteq 1_X \}$.
- The transformer relation $\xi_X : X \leftrightarrow X$ should be modified to:

$$\xi_X = \{ (a, a_0) \mid a_0 \subseteq a \cap 1_X \}$$

- We have a relation preservation square:

$$\begin{array}{ccc}
FX & \xrightarrow{t_X} & PX \\
\uparrow F\xi_X & & \uparrow P\xi_X = \Delta_{PX} \\
FX & \xrightarrow{t_X} & PX
\end{array}$$

- The “range” of $F\xi_X$ is $F(X_0) = (\varnothing F)(X)$, this still implies that morphisms $t : F \to P$ are uniquely determined by morphisms $t_0 : \varnothing F \to P$.
- We have a natural injection $\text{Hom}(F, P) \hookrightarrow \text{Hom}(\varnothing F, P)$ rather than a bijection.
In the presence of divergence

- Divergence is quite a bit more subtle.
- The passive store $X_0$ has all approximations of the do-nothing transformation $\mathcal{T}_{X_0} = \{ a \mid a \subseteq 1_X \}$.
- The transformer relation $\xi_X : X \leftrightarrow X$ should be modified to:

  $$\xi_X = \{ (a, a_0) \mid a_0 \subseteq a \cap 1_X \}$$

- We have a relation preservation square:

  $\begin{array}{ccc}
  F\xi_X & \xrightarrow{t_X} & P\xi_X \\
  \downarrow & & \uparrow \quad P\xi_X = \Delta_{P\xi_X} \\
  FX & \xrightarrow{t_X} & PX
  \end{array}$

- The “range” of $F\xi_X$ is $F(X_0) = (\wp F)(X)$, this still implies that morphisms $t : F \to P$ are uniquely determined by morphisms $t_0 : \wp F \to P$.
- We have a natural injection $\text{Hom}(F, P) \hookrightarrow \text{Hom}(\wp F, P)$ rather than a bijection.
In the presence of divergence

- Divergence is quite a bit more subtle.
- The passive store $X_0$ has all approximations of the do-nothing transformation $\mathcal{T}_{X_0} = \{ a \mid a \subseteq 1_X \}$.
- The transformer relation $\xi_X : X \leftrightarrow X$ should be modified to:

$$\xi_X = \{ (a, a_0) \mid a_0 \subseteq a \cap 1_X \}$$

- We have a relation preservation square:

$$
\begin{array}{ccc}
FX & \xrightarrow{t_X} & PX \\
\downarrow F\xi_X & & \uparrow P\xi_X = \Delta_{PX} \\
FX & \xrightarrow{t_X} & PX
\end{array}
$$

- The “range” of $F\xi_X$ is $F(X_0) = (\varnothing F)(X)$, this still implies that morphisms $t : F \rightarrow P$ are uniquely determined by morphisms $t_0 : \varnothing F \rightarrow P$.
- We have a natural injection $\text{Hom}(F, P) \hookrightarrow \text{Hom}(\varnothing F, P)$ rather than a bijection.
In the presence of divergence

- Divergence is quite a bit more subtle.
- The passive store $X_0$ has all approximations of the do-nothing transformation $\mathcal{T}_{X_0} = \{ a \mid a \subseteq 1_X \}$. 
- The transformer relation $\xi_X : X \leftrightarrow X$ should be modified to:
  \[
  \xi_X = \{ (a, a_0) \mid a_0 \subseteq a \cap 1_X \}
  \]
- We have a relation preservation square:

\[
\begin{array}{c}
FX & \xrightarrow{t_X} & PX \\
\downarrow{F\xi_X} & & \downarrow{P\xi_X = \Delta_{PX}} \\
FX & \xrightarrow{t_X} & PX
\end{array}
\]

- The “range” of $F\xi_X$ is $F(X_0) = (\wp F)(X)$, this still implies that morphisms $t : F \to P$ are uniquely determined by morphisms $t_0 : \wp F \to P$.
- We have a natural injection $\text{Hom}(F, P) \hookrightarrow \text{Hom}(\wp F, P)$ rather than a bijection.
We have what appears to be the first extensional semantic model that captures passivity.

The closest work in the literature is that of Tennent’s model (“Semantical analysis of Specification Logic”).

Tennent’s model is a generalization of the Oles’s model. Ours is a generalization of the Reynolds’s model.

The intuitions behind both the models are quite close.

Tennent’s model is intensional (deliberately). The intensional aspects seem hardwired. It does not seem possible to make it extensional.

Reynolds’s intuitions were right to begin with, even though it seems to have taken us a long time to realize that fact!