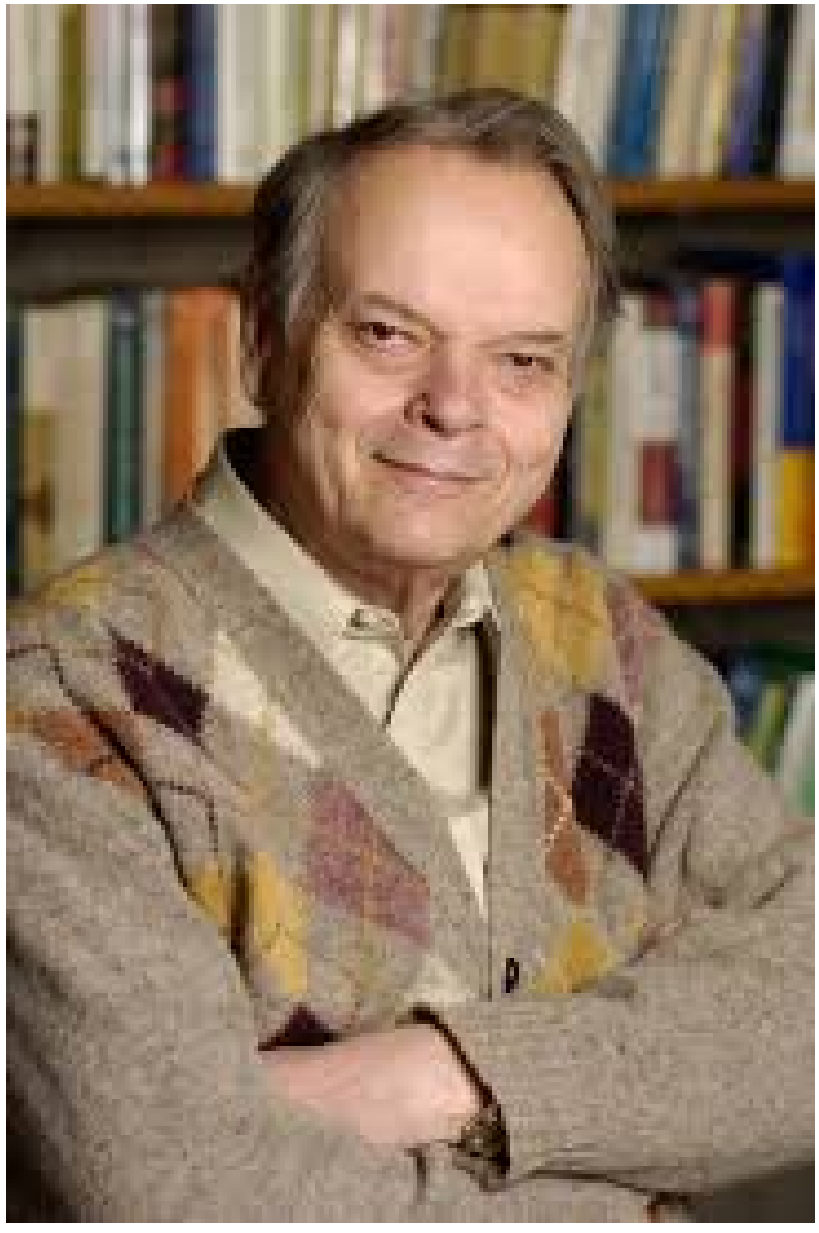
Transformation Monoids in Programming Language Semantics

Uday S. Reddy[1]

[1] University of Birmingham

NBSAN Manchester, July 2013



This talk is dedicated to the memory of
John C. Reynolds, 1935-2013

Section 1

Introduction

Mathamatics and Computer Science

The spirit of Arithmetic looks down on the quarrel between the new 'algorists' using written numerals and the traditional 'abacists' with their counting table, from an illustration in *Margarita Philosophica*, 1504.

Mathamatics and Computer Science

- Until the advent of digital computers (c. 1950), mathematics and computing science were a single joint discipline.
- Mathematicians routinely formulated algorithms (or constructions), and studied the properties of algorithms as well as the products of these algorithms.
- Examples:
  - algorithms for arithmetic (prehistoric times)
  - constructions for geometry (Euclid)
  - linear equations (Gauss, Jacobi, Seidel)
  - roots of functions (Newton, Viete, Bernoulli)
  - solutions for differential equations (Euler, Runge, Kutta)

Functional computation

- A good number of these algorithms were “functional,” i.e., the steps of computations were functions, joined together by function composition.
- Functional computation was given “semantics” by set theory (c. 1900).
- Formalisms (languages) for functional computation were devised by Godel, Church and Kleene (c. 1930).
- After the advent of Computer Science, Dana Scott and Gordon Plotkin extended the classical models to deal with recursion (domain theory).
- Moral: Functional computation is well-understood.

Procedural (imperative) computation

- Many iterative algorithms were "procedural," i.e., steps of computations modify the "state" of "variables."
- Examples:
  - Euclid's algorithm for greatest common divisor (GCD).
  - Gaussian elimination.
  - Iterative algorithms for roots, differential equations etc.
- No semantic models or formal languages were devised for procedural computation prior to Computer Science.
- Turing (1930's) and von Neumann (1940's) studied machine models and mechanical computation, which became the starting point for Computer Science.
- Automata theory (McCulloch & Pitts, 1943) arose as a machine model, and developed a significant algebraic theory (Kleene, Ginzburg, Eilenberg, Krohn, Rhodes, . . .)
- However, algebraic automata theory has not yet made a significant impact on programming theory.

Section 2

Procedural programming languages

Example: Euclid's algorithm

Elements, Book VII, Proposition 2
Given two (natural) numbers not prime to one another, to find their greatest common "measure".

Let *AB*, *CD* be the two given numbers.

...

If *CD* does not measure *AB* then:

- the lesser of the numbers *AB*, *CD* being *continually subtracted* from the greater,
- some number will be left which will measure the *one before it*.

...

Therefore, *CF* will be a common measure of *AB*, *CD*.

Euclid's algorithm in Algol 60

```
integer procedure gcd(integer a, integer b)
{   integer u, v;
    u := a; v := b;
    while u ≠ v do {
        if u > v then
            u := u − v
        else
            v := v − u
    }
    gcd := u
}
```

- From the outside gcd appears as a "function", e.g., gcd(10,4).
- However, internally, the procedure of gcd allocates local variables and continually modifies them.

Euclid's algorithm in Algol 60 (contd)

- Expressions: e.g., gcd(10, 4), gcd($x + y$, $2 * z$), $u > v$, $u - v$. They read the state, but do not change it.
- Commands: e.g., $u := u - v$. They modify the state.
- Variables: The symbols u and v name variables. They are modified during the execution.
- Procedures and parameters: The symbols gcd, a, and b are called "identifiers" in Algol 60. They name things, but there is no notion of modifying them.

Euclid's algorithm in Idealized Algol

```
gcd : exp[int] × exp[int] → exp[int]
   = λ(a, b). {
        blockexp[int] g.
        {    new[int] u.
             new[int] v.
             {
                  u := a; v := b;
                  ...
                  g := u
             }
        }
    }
```

John Reynolds noted in "*The Essence of Algol*" (1981) that the type structure of Algol was that of Church's simply typed lambda calculus.

Euclid's algorithm in Idealized Algol (contd)

- $\lambda(a, b)$. — function abstraction.
- **blockexp**[**int**] g. — Expression created from a command block with a local variable g. (The value of the expression is the final value of g.)
- **new**[**int**] u. — A command operator for creating a local variable.
- $u := a$ — an assignment command. (Change the value of u to the current value of a.)
- $-; -$ sequential composition operator for commands.

Idealized Algol

- Typed lambda calculus with base types:
  - **exp**[**int**], **exp**[**bool**]
  - **comm**
  - **var**[**int**], **var**[**bool**]
- Typed lambda calculus means: for any types A and B,
  - **unit** is a type.
  - $A \times B$ is a type.
  - $A \to B$ is a type.
- Given any terms T, T_1 and T_2, the following are terms:
  - $(\,)$, (T_1, T_2), $\mathit{fst}(T)$, $\mathit{snd}(T)$, $\lambda x.\ T$ and $T_1(T_2)$.
- All the primitive operators of Idealized Algol can be treated as constants of the typed lambda calculus. For example:

$$
\begin{array}{lcl}
- ; - & : & \mathbf{comm} \times \mathbf{comm} \to \mathbf{comm} \\
\mathbf{skip} & : & \mathbf{comm} \\
- := - & : & \mathbf{var}[X] \times \mathbf{exp}[X] \to \mathbf{comm} \\
\mathbf{new}[X] & : & (\mathbf{var}[X] \to \mathbf{comm}) \to \mathbf{comm}
\end{array}
$$

Defining semantics

- Find an appropriate mathematical structure to capture the meaning of the terms.
- Meaning function $\mathcal{M} : \text{Terms} \to \text{Structure}$
- Think of the terms as forming a “free algebra” generated by the operators of the language.
- The semantic structure is a meaningful “algebra” that identifies all computations with the same behaviour.
- $\mathcal{M}$ must be a structure-preserving map, e.g.,

$$\mathcal{M}[\![v := e]\!] \;=\; \mathcal{M}[\![v]\!] := \mathcal{M}[\![e]\!]$$

The brackets $[\![\cdots]\!]$ delineate the symbolic world from the mathematical world.

The objective of semantics

- The meaning function gives a notion of semantic equivalence:

$$T_1 \cong_{\text{sem}} T_2 \iff \mathcal{M}[\![T_1]\!] = \mathcal{M}[\![T_2]\!]$$

- On the other hand, by looking at the behaviour of the terms in all possible contexts, we obtain observational equivalence:

$$T_1 \cong_{\text{obs}} T_2 \iff \forall \text{ contexts } \mathcal{C},\ \mathcal{C}[T_1] \text{ and } \mathcal{C}[T_2] \text{ have same behaviour}$$

- A semantic model is said to be fully abstract if the two coincide:

$$T_1 \cong_{\text{sem}} T_2 \iff T_1 \cong_{\text{obs}} T_2$$

Example equivalences

- Typed lambda calculus equivalences:

$$\begin{aligned} \mathit{fst}(T_1, T_2) &\equiv T_1 \\ \mathit{snd}(T_1, T_2) &\equiv T_2 \\ (\lambda x.\{T\})(T') &\equiv T[x \to T'] \end{aligned}$$

  (where $T[x \to T']$ means substitution of T' for x).

- Basic equivalences for commands:

$$\begin{aligned} &(T_1; T_2); T_3 \equiv T_1; (T_2; T_3) \\ &T; \mathbf{skip} \equiv T \equiv \mathbf{skip}; T \\ &\mathbf{new}[X]\ x.\, \mathbf{skip} \equiv \mathbf{skip} \\ &\mathbf{new}[X]\ x.\, T_0 \equiv T_0 \\ &\mathbf{new}[X]\ x.\, \mathbf{new}[X]\ y.\, T \equiv \\ &\qquad\qquad \mathbf{new}[X]\ y.\, \mathbf{new}[X]\ x.\, T \end{aligned}$$

Example equivalences: Locality

- Since a local variable is only available inside its block of commands, any subterms that occur outside the scope do not have access to it.
- If $p : \mathbf{comm} \to \mathbf{comm}$,

$$\begin{aligned} \mathbf{new}[\mathbf{int}]\ x.\{x := 0;\ p(x := x + 1)\} \\ \equiv\ p(\mathbf{skip}) \\ \equiv\ \mathbf{new}[\mathbf{int}]\ x.\{x := 0;\ p(x := x - 1)\} \end{aligned}$$

- If $p : \mathbf{exp}[\mathbf{int}] \times \mathbf{comm} \to \mathbf{comm}$,

$$\begin{aligned} \mathbf{new}[\mathbf{int}]\ x.\{x := 0;\ p(x, x := x + 1)\} \\ \equiv\ \mathbf{new}[\mathbf{int}]\ x.\{x := 0;\ p(-x, x := x - 1)\} \end{aligned}$$

Example equivalences: irreversible state change

- The execution of a command changes the state irreversibly.
- Suppose p : **comm** $\to$ **comm**.

$$\begin{aligned}&\textbf{new}[\textbf{int}]\ x.\ \{\ x := 0;\\ &\qquad\qquad\quad p(x := x + 1);\\ &\qquad\qquad\quad \textbf{if } x > 0 \textbf{ then diverge else skip}\}\\ &\equiv p(\textbf{diverge})\end{aligned}$$

  where **diverge** is any diverging command, e.g., an infinite loop.
- If p ignores its argument (constant function), then both the sides are equivalent to p(**skip**).
- If p runs its argument command, then both the sides diverge.
- This notion of irreversible state change has proved very hard to capture in the mathematical models!

The challenge for denotational semantics

- How to capture the intensional aspects of computations in an extensional model?
- **Example of extensionality**:

$$\mathsf{GV}(x) \implies (x := !x + 1;\ x := !x + 1) \equiv (x := !x + 2)$$

- 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.

$$\mathit{read}_x(2),\ \mathit{write}_x(3),\ \mathit{read}_x(3),\ \mathit{write}_x(4)$$
$$\mathit{read}_x(2),\ \mathit{write}_x(4)$$

- We are after extensional models.
- **Another example of extensionality**:

$$\mathit{stack}(s) \implies (s.\mathit{push}(v);\ s.\mathit{pop}) \equiv \textbf{skip}$$

Naive extensional models have “junk”

- Command snapback (with divergence):

$$\textbf{try} : \textbf{comm} \to \textbf{comm}$$
$$\textbf{try}\ c \;=\; \lambda s.\ \begin{cases} s, & \text{if } c(s) \neq \bot \\ \bot, & \text{if } c(s) = \bot \end{cases}$$

  State changes should be irreversible.

- Expression snapback:

$$\textbf{do\_result\_} : \textbf{comm} \times \textbf{exp} \to \textbf{exp}$$
$$\textbf{do}\ c\ \textbf{result}\ e \;=\; \lambda s.\ e(c(s))$$

  Expressions should only read the state (passivity).

- Intensional models can eliminate such “junk” relatively easily.
- Eliminating “junk” in extensional models involves inventing mathematical structure.

Progress in understanding semantics

- Christopher Strachey & Dana Scott, 1960's:
  - models based on global state using sets with added structure for recursion ("domains").
- John Reynolds, 1981:
  - models based on presheaves
  - state varies based on allocation of variables
  - the structure of stores based on automata intuitions.
- John Reynolds, 1983:
  - Relational parametricty to capture type-based abstraction of data ("uniformity" or "information hiding")
- O'Hearn & Tennent, 1993:
  - combined presheaves with relational parametricity to achieve a model of local variables.
- Reddy & Dunphy, 1998–2013
  - categorical axiomatization of relational parametricity
  - development of automata-theoretic ideas to model irreversible state change.

Section 3

Semantic framework

Cartesian closed categories

- A model of a typed lambda calculus must be a cartesian closed category.
- For all objects A and B, there is a product $A \times B$ and an exponential $A \Rightarrow B$ (or B^A).
- A standard example of a cartesian closed category is **Set**, the category of sets and functions.
- From category theory, we also know that presheaves, i.e., functors of type $\mathcal{C} \to \mathbf{Set}$ (for any category $\mathcal{C}$), form a cartesian closed category.
- However, plain category theory is not good enough for our purposes.

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)
- For example, if A and A' are monoids, a logical relation $R : A \leftrightarrow A'$ is a relation between their underlying sets such that

$$1_A \left[R\right] 1_{A'}$$
$$x \left[R\right] x' \ \wedge \ y \left[R\right] y' \implies xy \left[R\right] x'y'$$

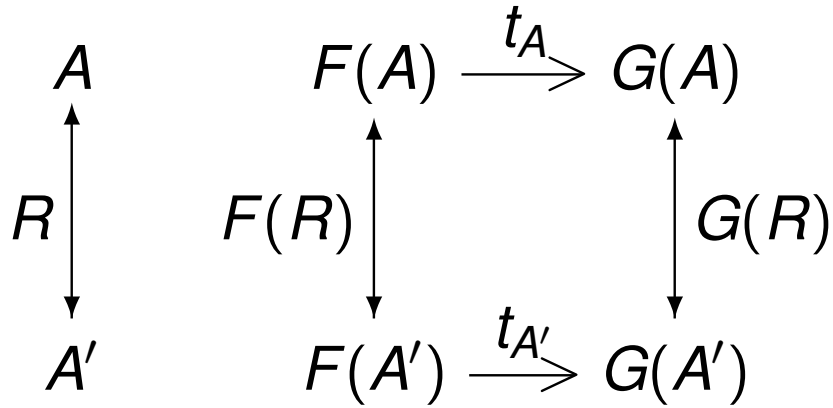
  A useful abbreviation for the second property is to write

$$\cdot \left[R \times R \to R\right] \cdot$$

  (which justifies the name “logical” relation).

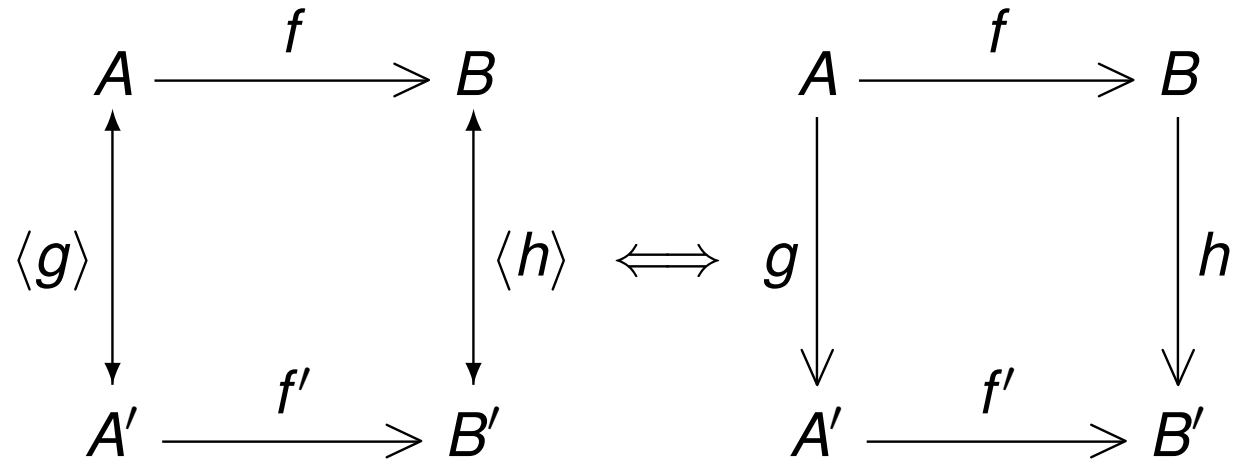
Relational parametricity

- Relational parametricity asks for "parametrically polymorphic" functions to preserve all logical relations.



- The polymorphic family $\mathbf{t} = \{t_A\}_A$ is said to have the type $\forall_A F(A) \to G(A)$.

- In all the structures we use, logical relations subsume homomorphisms. For every homomorphism $h : A \to B$, there is a logical relation $\langle h \rangle : A \leftrightarrow B$ which maps commutative squares to relation-preservation squares.



- Relational parametricity strengthens naturality of category theory as a uniformity criterion.

Why relational parametricity?

- Recall the equivalence (for $p : \mathbf{comm} \to \mathbf{comm}$):

$$\begin{aligned}&\mathbf{new}[\mathbf{int}]\ x.\{x := 0;\ p(x := x + 1)\}\\ &\quad\equiv\ \mathbf{new}[\mathbf{int}]\ x.\{x := 0;\ p(x := x - 1)\}\end{aligned}$$

- The argument depends on the fact:
  - "x is hidden from p."
- Equivalent to:
  - "p is parametrically polymorphic in the state space of x."
- I.e., p must preserve all possible relationships between the state spaces of x.
- For example, $R : \mathbb{Z} \leftrightarrow \mathbb{Z}$ given by:

$$n \left[R\right] n' \iff n \geq 0 \ \wedge\ n' = -n$$

- Since $x := x + 1$ and $x := x - 1$ preserve this relation, $p(x := x + 1)$ and $p(x := x - 1)$ must preserve it too.

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":
  - $\textsc{Comm}(X) =$ the collection of state transformations for stores of shape X.
  - $\textsc{Exp}(X) =$ 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.

    $$\forall_{f:Y \leftarrow X} F(Y) \to G(Y) \qquad \int_{f:Y \leftarrow X} F(Y) \to G(Y)$$

    (Think of this is an intuitionistic function space.)
  - $\textsc{Var}(X) = \textsc{Exp}(X) \times [\mathit{Int} \to \textsc{Comm}(X)]$, 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.)

Section 4

Transformation monoids

Reynolds transformation monoids

- A store X is represented as a tuple

$$(\mathcal{Q}_X, \mathcal{T}_X, \alpha_X, \mathit{read}_X)$$

(Reynolds transformation monoid) where:

- $\mathcal{Q}_X$ - a (small) set of states,
- $\mathcal{T}_X$ - a monoid of state transformations $\mathcal{T}_X \subseteq [\mathcal{Q}_X \to \mathcal{Q}_X]$,
- $\alpha_X : \mathcal{T}_X \hookrightarrow [\mathcal{Q}_X \to \mathcal{Q}_X]$ - the implicit monoid action,
- $\mathit{read}_X : [\mathcal{Q}_X \to \mathcal{T}_X] \to \mathcal{T}_X$ - called "diagonalization":

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

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

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

- **Note**: Transformation monoids in algebraic automata theory [Eilenberg, 1974] are triples $(\mathcal{Q}_X, \mathcal{T}_X, \alpha_X)$.

Logical relations for RTM's

- A logical relation $R : (\mathcal{Q}_X, \mathcal{T}_X) \leftrightarrow (\mathcal{Q}_{X'}, \mathcal{T}_{X'})$ is a pair (R_q, R_t) where

$$\begin{array}{c} X \\ \updownarrow R \\ X' \end{array} \quad = \quad \left(\begin{array}{cc} \mathcal{Q}_X & \mathcal{T}_X \\ \updownarrow R_q \ , & \updownarrow R_t \\ \mathcal{Q}_{X'} & \mathcal{T}_{X'} \end{array} \right)$$

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

  such that
  - $\alpha_X \left[R_t \to [R_q \to R_q]\right] \alpha_{X'}$, and
  - $\mathit{read}_X \left[[R_q \to R_t] \to R_t\right] \mathit{read}_{X'}$.

Morphisms for RTM's

- A homomorphism $f : (\mathcal{Q}_X, \mathcal{T}_X) \to (\mathcal{Q}_Y, \mathcal{T}_Y)$ is a pair (f_q, f_t)

$$\begin{array}{c} Y \\ \uparrow f \\ X \end{array} \quad = \quad \left(\begin{array}{c} \mathcal{Q}_Y \\ \downarrow f_q \\ \mathcal{Q}_X \end{array} \quad , \quad \begin{array}{c} \mathcal{T}_Y \\ \uparrow f_t \\ \mathcal{T}_X \end{array} \right)$$

  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 $(\langle f_q \rangle^\smile, \langle f_t \rangle)$ 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). $R^\smile$ means the converse relation.

Interpretation of Algol types

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

$$\begin{aligned} \textsc{Comm}(X) &= \mathcal{T}_X \\ \textsc{Exp}(X) &= [\mathcal{Q}_X \to \mathit{Int}] \\ (F \times G)(X) &= F(X) \times G(X) \\ (F \Rightarrow G)(X) &= \forall_{f:Y \leftarrow X} F(Y) \to G(Y) \end{aligned}$$

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

- **Fact**: $\mathrm{Hom}(\textsc{Comm} \to \textsc{Comm}) \cong \mathbb{N}$, representable by

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

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

What next?

- I know precious little about the structure of the category **RTM**.
- If we restrict to full transformation monoids (Reynolds, 1981), then every morphism $X \to Y$ is equivalent to an isomorphism $Y \cong X \times D$ for some D.
- We need similar characterisations for **RTM**.
- Functors $F : \mathcal{C} \to \mathbf{Set}$ that are subsumptive form a cartesian closed category. $F(\langle g \rangle) = \langle F(g) \rangle$.
- A more satisfying condition is that of fibration. In a fibred category with relations, we have preimages $(f, f')^* S$ which "pull S back" along f and f'.
- I don't yet have a cartesian closed category result for fibred functors. We would need something like sheaves instead presheaves to get such a category.
- Possible connections to Galois theory.