A Certain Loss of Identity

Geraint Jones
Oxford University Computing Laboratory
Oxford, England

Mary Sheeran
Informationsbehandling, Chalmers Tekniska Högskola
Göteborg, Sweden

Abstract

For pragmatic reasons it is useful to exclude the identity relation from the 'implementable subset' of Ruby. However there are many expressions in the relational calculus whose natural meaning is just this identity relation. This note gives an identity-free account of some of these expressions, and shows that there is no satisfactory identity-free account of some others. This is an exercise in writing about Ruby without drawing any pictures, in part because it is about those expressions which would correspond to blank pictures.

What there is when there is nothing there

In Ruby* one uses relations to represent circuit components, and the composition $R\,;S$ of relations corresponds to some connection of two components in which the parts of the R represented by its range are connected to the parts of the S represented by its domain. With this interpretation, the repeated composition R^n naturally represents a 'pipeline' of n components, each an R, connected in a linear array. At least, this is the natural interpretation for $n > 0$.

For strictly positive n and m it is clearly the case (and can be shown by induction) that R^{n+m} is the composition $R^n\,;R^m$ of smaller powers of R. This suggests that it might be desirable for any definition of R^0 to be consistent with $R^n = R^n\,;R^0 = R^0\,;R^n$. One possible choice of a value for R^0 consistent with these equations would be the identity (or equality) relation, id, for which $x\ id\ x$ for all possible x.

Sub-expressions whose values are as 'big' as the identity relation are generally undesirable in Ruby expressions which might be classed as 'implementations'. These expressions are the *results* of refinements, ones which we expect to be able to pass to an implementor, perhaps a silicon manufacturer, with the expectation that they will be accurately represented by some (probably finite) machine operating on a physical data representation. For example, in refining to a design to be implemented in digital electronics, we would not want to say that we had completed the design until all the components in the expressions were relations between tuples of binary signal values.

*This paper is not meant to be self-contained, and should be read in the context of an introduction to Ruby, such as reference [1].

If a circuit contains, for example, a component which is $\mathsf{fst}\,R$ for some R, where $\mathsf{fst}\,R = [R, id]$ is the parallel composition of an R with an instance of the identity, we should not necessarily expect a manufacturer to know how to implement the id which this requires. In fact we would normally not be too concerned about this particular case: so long as it appears in a composition, such as $S;\mathsf{fst}\,R$ it can be read as an instruction to the manufacturer to construct an S, and to attach an R to the first connection (of the pair) in its range. On its own, with nothing connected to the second wire in either the domain or range of $\mathsf{fst}\,R$, it is just as unrealistically unimplementable as was id alone.

If asked we could give an account of this in the refinement: suppose we know that the type of the data handled by the troublesome identity relation is T, that is suppose that there is some partial equivalence relation (per) T for which $S = S\,;\mathsf{snd}\,T$. We can calculate that $S\,;\mathsf{fst}\,R = S\,;\mathsf{snd}\,T\,;\mathsf{fst}\,R = S\,;[R,T]$, and we are asking the implementor to provide us with an S, and an R alongside a T. The specification of a T can usually be read as a request for 'a wire capable of carrying signals of type T', and would be feasible for those types T that had physical representations.

In a similar way, we could choose to say that an expression is only implementable if it contains no instances of R^n for which n is zero. This is unpleasant, both because it adds an arithmetic side-condition to the calculation, and because we find R^0 in some expressions that we certainly do want to be able to call implementable. In particular, the relation $\mathsf{tri}\,R$ constructs a triangular array of successively higher powers of R which are applied in parallel to tuples of signals, that is $\mathsf{tri}\,R = [R^0, R^1, R^2, \ldots]$, but we would often want to call $\mathsf{tri}\,R$ an implementable relation.

Although we could treat $\mathsf{tri}\,R$ in the same way as $\mathsf{fst}\,R$ it turns out to be more convenient to change the definition of R^0 so that $\mathsf{tri}\,R$ can often be called implementable even when not in a composition. For example, if we know that R is a component which only deals with Ts, it would be convenient to be able to say that $\mathsf{tri}\,R$ deals only with lists of Ts.

An idea of type, and an ideal type

First let us introduce some notation for type judgements. We write R^{-1} for the converse of a relation, and R^{-n} for $(R^{-1})^n = (R^n)^{-1}$. Our types are the partial equivalence relations, and we note that the following are equivalent: $\mathrm{per}\,T \Leftrightarrow (T^{-1} \subseteq T \supseteq T^2) \Leftrightarrow (T \supseteq T^2 \cup T^{-1}) \Leftrightarrow (T^{-1} = T = T^2) \Leftrightarrow (T = T\,;T^{-1})$.

We write $R \vdash T$ for $R = R\,;T$, and $U \dashv R$ for $U\,;R = R$, and say that 'T is a right type for R', and 'U is a left type for R'. Notice that since $T = T^{-1}$ for any per T, left types of R are just right types of R^{-1} and vice versa, $T \dashv R \Leftrightarrow R^{-1} \vdash T$. For conciseness, we write $R : U \sim T$ for $U \dashv R\;\&\; R \vdash T$. We will only use these symbols for type judgements when the relations in the positions of T and U are pers.

The idea is to define R^0 in such a way that if $R : T \sim T$, then $R^0 : T \sim T$, and it would follow that $\mathsf{tri}\,R : \mathsf{map}\,T \sim \mathsf{map}\,T$. Because R^0 is going to be absorbed on either side by (non-zero) powers of R, it seems that we should choose it to be both a right type and left type for R. The identity relation is certainly a candidate, but is too big to be implementable. We need to find a smaller left and right type.

A suitable value for R^0 would be the unique per for which $R : R^0 \sim R^0$ and $R : T \sim T \;\Rightarrow\; R^0 : T \sim T$. It is in fact the composition of all those pers T for which $R : T \sim T$. Of course it is not immediately plain that R^0 is well defined by either of these characterisations: it needs to be shown that there is such an optimal type; and it is not always going to be possible to give a meaning to the composition $\mathop{;} S$ of the elements of some set S, because composition is neither commutative nor idempotent.

Looking at one side of the problem

Let us begin with the simpler problem of justifying the use of the term '*the* right type' of a relation R. We say that a per T is *a* right type of R if $R \vdash T$. It turns out that there always is an optimal right type, one which we will call *the* right type and which we will write $R{\triangleright}$, and which exactly determines the set of right types of R.

The right-guaranteeing relation $\vdash$ turns out to be a partial order on pers (although not on relations in general). That is to say, it is transitive ($A \vdash B \,\&\, B \vdash C \;\Rightarrow\; A = A \,;\, B = A \,;\, (B \,;\, C) = (A \,;\, B) \,;\, C = A \,;\, C \;\Rightarrow\; A \vdash C$) and antisymmetric ($A \vdash B \,\&\, B \vdash A \;\Rightarrow\; A = A \,;\, B = (B^{-1} \,;\, A^{-1})^{-1} = (B \,;\, A)^{-1} = B^{-1} = B$). Of all the right types of R there is one which is least under this ordering, that is there is a per $R{\triangleright}$ for which both $R \vdash R{\triangleright}$, and $R \vdash T \;\Leftrightarrow\; R{\triangleright} \vdash T$.

Informally, a right type of R is a per for which the equivalence classes are classes of elements of the range of R that are treated in the same way by R. That is, x and y can only be equivalent if $\forall z.\, z\; R\; x \;\Leftrightarrow\; z\; R\; y$. Any right type must be a refinement of the coarsest equivalence relation of this kind, $\{x, y \mid \forall z.\, z \,R\, x \;\Leftrightarrow\; z \,R\, y\}$. The optimal right type is the coarsest per restricted to only those elements which it must mention because they are in the range of R – that is the relation with the smallest number of elements, and the smallest number of classes, of all right types of R. This relation is

$$\{x, y \mid (\exists z.\, z\; R\; x \;\&\; z\; R\; y) \;\&\; (\forall z.\, z\; R\; x \;\Leftrightarrow\; z\; R\; y)\}$$

Because $\forall z.\, z\; R\; x \;\Leftrightarrow\; z\; R\; y$, the condition $\exists z.\, z\; R\; x \;\&\; z\; R\; y$ is equivalent to $\exists z.\, z \,R\, x$, and says exactly that the smallest right type relates only elements of the range of R.

Define the *right view* of R by $R{>} = R^{-1} \,;\, R$. This relation will serve us as a rough description of the range of R, which is $\{y \mid \exists x.\, x\; R\; y\}$. Clearly $R{>}^{-1} = R{>}$, but it need not itself be a type because it is not necessarily transitive: consider the relation $\Sigma = \{\langle a, a\rangle, \langle a, b\rangle, \langle c, b\rangle, \langle c, c\rangle\}$ for which $a\,(\Sigma{>})\,b \,\&\, b\,(\Sigma{>})\,c$ but not $a\;(\Sigma{>})\;c$.

The right view does put a bound on the right type, because if $R \vdash T$ then $R{>} \,;T = R^{-1} \,; R ; T = R^{-1} \,; R = R{>}$ so $R{>} \vdash T$. Although the right view is not necessarily a type, it does always contain the type we need, and we will show that the smallest right type is

$$R{\triangleright} = \bigcup\{R{>} \cap T \mid R \vdash T\}$$

The attraction of this characterisation is that the family $\{R{>} \cap T \mid R \vdash T\}$ is clearly small enough to be a set, because $(R{>} \cap T) \in \mathbb{P}(\operatorname{ran} R \times \operatorname{ran} R)$.

If R is its own converse, its transitive closure $R^+ = \bigcup\{R^n \mid n \geq 1\}$ is a per. It is its own converse because each term in the union is its own converse, and $(R^+)^2 = \bigcup\{R^n \mid n \geq 1\}^2 = \bigcup\{R^n \mid n \geq 2\} \subseteq R^+$. We will need this in constructing a bound for two right types. If T and U are two pers, then $T \cup U$ is its own converse, so $(T \cup U)^+$ is a per. If $R \vdash T$ and $R \vdash U$, taking the union of the two guarantees, $R = R \cup R = (R ; T) \cup (R ; U) = R ; (T \cup U)$, then by induction $R = R ; (T \cup U)^n$, and taking a union $R = R ; (T \cup U)^+$, so $R \vdash (T \cup U)^+$.

The right stuff

First we must show that $R\triangleright$ is a per. It is clearly its own converse, because $R{>}^{-1} = R{>}$ and if T is a per then $T^{-1} = T$, so $R\triangleright^{-1} = \bigcup\{R{>} \cap T \mid R \vdash T\}^{-1} = \bigcup\{(R{>} \cap T)^{-1} \mid R \vdash T\} = R\triangleright$. To show that it is transitive, consider any two right types T and U of R, then $T \dashv R{>}$, $R{>} \vdash U$, and $T ; U \subseteq (T \cup U)^2 \subseteq (T \cup U)^+$, so $(R{>} \cap T) ; (R{>} \cap U) \subseteq R{>}^2 \cap (T ; R{>}) \cap (R{>} ; U) \cap (T ; U) \subseteq R{>} \cap (T \cup U)^+$. Moreover $(T \cup U)^+$ is itself a right type of R, so $R\triangleright^2 = \bigcup\{R{>} \cap T \mid R \vdash T\}^2 = \bigcup\{R{>} \cap T \mid R \vdash T\} ; \bigcup\{R{>} \cap U \mid R \vdash U\} = \bigcup\{(R{>} \cap T) ; (R{>} \cap U) \mid R \vdash T \;\&\; R \vdash U\} \subseteq \bigcup\{R{>} \cap T \mid R \vdash T\} = R\triangleright$.

Having shown that $R\triangleright$ is a type, we next show that it is a right type of R. Suppose that $R \vdash T$, and so that $R{>} \vdash T$. If $x\,(R ; T)\,y$ there must be a z for which $x\,R\,z$ and $z\,T\,y$, and so $z\,(R ; R^{-1} ; T)\,y$, but $R ; R^{-1} ; T = R{>}$. This z is therefore a witness to $x\,(R ; (R{>} \cap T))\,y$, so $R \subseteq R ; (R{>} \cap T)$, but $R ; (R{>} \cap T) \subseteq R ; T = R$, so $R = R ; (R{>} \cap T)$. Taking unions $R ; R\triangleright = R ; \bigcup\{R{>} \cap T \mid R \vdash T\} = \bigcup\{R ; (R{>} \cap T) \mid R \vdash T\} = \bigcup\{R \mid R \vdash T\} = R$, at least provided that every R has at least one right type.

The right type of a relation gives us another characterisation of a per. A relation is its own right type if and only if it is itself a per. One direction is obvious, because a relation equal to its own right type (which is a per) must be a per. In the other direction, if R is a per it is its own square, and its own right view, and $R \vdash R$. This means that $R \in \{R{>} \cap T \mid R \vdash T\}$ so $R \subseteq \bigcup\{R{>} \cap T \mid R \vdash T\}$. Conversely $\bigcup\{R{>} \cap T \mid R \vdash T\} \subseteq \{R{>} \mid R \vdash T\} = R{>} = R$, showing that if R is a per, $R = R\triangleright$.

It remains to show that $R\triangleright$ is the minimal right type of R, that is that $R \vdash T \;\Leftrightarrow\; R\triangleright \vdash T$, for any per T. Because $R \vdash R\triangleright$ we have that $R\triangleright \vdash T \;\Rightarrow\; R \vdash T$ by transitivity. Notice that we need to know that T must be a per to be able to show that the other implication holds, because although $R ; T = R \;\Rightarrow\; R{>} ; T = R{>}$ in general $R ; T = R \not\Rightarrow R\triangleright ; T = R\triangleright$. Consider the relation $R = \{\langle a, a\rangle, \langle a, b\rangle\}$, and $T = R$ for which $R ; T = R^2 = R$ but $R\triangleright = \{\langle a, a\rangle, \langle b, b\rangle\}$ so $R\triangleright ; T = T \neq R\triangleright$.

Suppose that $R \vdash T$, then certainly $R{>} \vdash T$, and for each U that is also a right type of R, so is $(T \cup U)^+$, so $(R{>} \cap U) ; T \subseteq (R{>} ; T) \cap (U ; T) \subseteq R{>} \cap (T \cup U)^+$, and taking unions $R\triangleright ; T = \bigcup\{R{>} \cap U \mid R \vdash U\} ; T = \bigcup\{(R{>} \cap U) ; T \mid R \vdash U\} \subseteq \bigcup\{R{>} \cap (T \cup U)^+ \mid R \vdash U\} \subseteq \bigcup\{R{>} \cap U \mid R \vdash U\} = R\triangleright$. Conversely suppose that $x\,R\triangleright\,y$, then there is some z for which $z\,R\,y$, but $R = R ; T$ and T is a per so $y\,T\,y$ and $x\,(R\triangleright ; T)\,y$. This shows that $R\triangleright ; T \supseteq R\triangleright$, completing the proof that $R \vdash T \;\Rightarrow\; R\triangleright \vdash T$.

On the other hand

It hardly needs saying, but there is a perfect symmetry between the left and right of a relation. Recall that $T \dashv R \Leftrightarrow R^{-1} \vdash T$, so defining $<R = R^{-1}>$ and $\lhd R = R^{-1} \rhd$ it can be seen that everything that has been said here about right types can be said dually about left types.

A meeting of types

A left and right type of R is just a type which is both a left type and a right type of R, that is a T for which $R : T \sim T$. Again, it turns out that there is an optimal one, which guarantees all the others. It is the smallest per which relates everything in the domain or in the range of R, but which is a refinement of both the left type and the right type of R. It will turn out to be $\bigcup\{(<R \cup R>) \cap T \mid R : T \sim T\}$.

Informally $\bigcup\{T \mid R : T \sim T\}$ is the coarsest refinement of both $\bigcup\{T \mid T \dashv R\}$ and $\bigcup\{T \mid R \vdash T\}$, and has as few classes as are needed to distinguish in the same way that R does all the elements of the domain and range. It has an additional equivalence class for all those things in neither the domain nor the range of R. The term $<R \cup R>$ excludes just this class.

We begin by formalising the idea of the smallest, coarsest mutual refinement of two pers. If U and V are pers, define

$$U \perp V = \bigcup\{(U \cup V) \cap T \mid U \vdash T \,\&\, V \vdash T\}$$

This is a per. If U, V and T are pers, $((U \cup V) \cap T)^{-1} = (U \cup V) \cap T$, so $U \perp V$ is its own converse. If each of T and T' are right types of (that is $\vdash$-greater than) both U and V, then $((U \cup V) \cap T) ; ((U \cup V) \cap T') \subseteq (T ; (U \cup V)) \cap (T ; T')$, and $T ; (U \cup V) = (T ; U) \cup (T ; V) = U \cup V$, and $(T ; T') \subseteq (T \cup T')^+$ which is a right type of both U and V. Therefore $(U \perp V)^2 \subseteq \bigcup\{(U \cup V) \cap (T \cup T')^+ \mid U, V \vdash T, T'\} \subseteq \bigcup\{(U \cup V) \cap T \mid U, V \vdash T\} = U \perp V$ showing that $U \perp V$ is transitive, and so is a per.

It is, moreover, a least upper bound of U and V with respect to the guaranteeing order. That is to say, $U \vdash T \,\&\, V \vdash T$ if and only if $(U \perp V) \vdash T$. First we will show that $U \vdash (U \perp V)$. From the definition, $U ; (U \perp V) = U ; \bigcup\{(U \cup V) \cap T \mid U, V \vdash T\} = \bigcup\{U ; ((U \cup V) \cap T) \mid U, V \vdash T\}$. Now $U ; ((U \cup V) \cap T) \subseteq (U ; T) = U$, but conversely if $x \, U \, y$, then y must be in the range of both U and T, and since each is a per, $y \, U \, y$ and $y \, T \, y$, therefore $x \, (U ; ((U \cup V) \cap T)) \, y$. So we know that $U = U ; ((U \cup V) \cap T)$, and taking unions $U ; (U \perp V) = \bigcup\{U \mid U, V \vdash T\} = U$, at least provided there is some T which is a right type of both U and V.

This shows that $U \vdash (U \perp V)$, and by symmetry $V \vdash (U \perp V)$. It follows by transitivity that if $(U \perp V) \vdash T$ then $U \vdash T$ and $V \vdash T$.

Now suppose that $U \vdash T$ and $V \vdash T$, then for any other T' which is both a right type of both U and V, the composition $(T' ; T) \subseteq (T' \cup T)^+$ which must also be a right type of both U and V. For these relations, $((U \cup V) \cap T') ; T \subseteq ((U \cup V) ; T) \cap (T' ; T) \subseteq (U \cup V) \cap (T' \cup T)^+$. Therefore, taking unions of both sides, $(U \perp V) ; T = \bigcup\{(U \cup V) \cap T' \mid U, V \vdash T'\} ; T \subseteq \bigcup\{(U \cup V) \cap (T' \cup T)^+ \mid U, V \vdash T'\} \subseteq \bigcup\{(U \cup V) \cap T' \mid U, V \vdash T'\} = U \perp V$.

Conversely, $U \perp V = \bigcup\{(U \cup V)\,;T' \mid U,V \vdash T'\} \subseteq \bigcup\{U \cup V \mid U,V \vdash T'\} = U \cup V$, since we know that U and V have at least one common right type, T. For this given type, $U \perp V \subseteq U \cup V = (U\,;T) \cup (V\,;T) = (U \cup V)\,;T$. Suppose that $x\,(U \perp V)\,y$, then y must be in the range of T, and since T is a per, $y\,T\,y$, so $x\,((U \perp V)\,;T)\,y$, showing that $(U \perp V) \subseteq ((U \perp V)\,;T)$. We have now shown that if $U \vdash T$ and $V \vdash T$ then $(U \perp V) \vdash T$, completing the proof that $U \perp V$ is the $\vdash$-l.u.b. of U and V.

From both sides now

We can now define the left-and-right type of R by $\lhd R \rhd = \lhd R \perp R \rhd$, and we will show that this is equal each of $\bigcup\{(\lhd R \cup R\rhd) \cap T \mid R : T \sim T\}$ and $\bigcup\{(<R \cup R>) \cap T \mid R : T \sim T\}$. First of all, we know that $R \vdash T$ if and only if $R\rhd \vdash T$, and similarly for the left type, so $R : T \sim T$ if and only if $T \dashv \lhd R \,\&\, R\rhd \vdash T$. Of course since $\lhd R$ and T are pers, the former condition is exactly that $\lhd R \vdash T$. Thus, immediately from the definition $\lhd R\rhd = \lhd R \perp R\rhd = \bigcup\{(\lhd R \cup R\rhd) \cap T \mid R : T \sim T\}$.

Now $R\rhd = \bigcup\{R> \cap T \mid R \vdash T\} \subseteq \bigcup\{R> \mid R \vdash T\} = R>$, at least provided every relation has some right type. Similarly, $\lhd R \subseteq {<}R$, so $\lhd R\rhd = \bigcup\{(\lhd R \cup R\rhd) \cap T \mid R : T \sim T\} \subseteq \bigcup\{(<R \cup R>) \cap T \mid R : T \sim T\}$.

Conversely $\bigcup\{(<R \cup R>) \cap T \mid R : T \sim T\} = \bigcup\{(<R \cap T) \cup (R> \cap T) \mid R : T \sim T\} = \bigcup\{<R \cap T \mid R : T \sim T\} \cup \bigcup\{R> \cap T \mid R : T \sim T\}$. But $R : T \sim T \;\Rightarrow\; R \vdash T$ so $\bigcup\{R> \cap T \mid R : T \sim T\} \subseteq \bigcup\{R> \cap T \mid R \vdash T\} = R\rhd$ and similarly for the left-hand term, $\bigcup\{<R \cap T \mid R : T \sim T\} \subseteq \lhd R$, so $\bigcup\{(<R \cup R>) \cap T \mid R : T \sim T\} \subseteq \lhd R \cup R\rhd$. Since each element of the family in the union is a subrelation of the union itself, $R : T \sim T \;\Rightarrow\; (<R \cup R>) \cap T \subseteq \lhd R \cup R\rhd$, so $(<R \cup R>) \cap T \subseteq (\lhd R \cup R\rhd) \cap T$. Taking unions of both sides, $\bigcup\{(<R \cup R>) \cap T \mid R : T \sim T\} \subseteq \lhd R\rhd$.

This completes the proof that $\lhd R\rhd = \bigcup\{(<R \cup R>) \cap T \mid R : T \sim T\} = \bigcup\{(\lhd R \cup R\rhd) \cap T \mid R : T \sim T\}$.

The left and right type of R is a per, because it is the l.u.b. of two pers. It is a right type of R, because being a l.u.b. it is guaranteed by the least right type, $R\rhd \vdash \lhd R\rhd$, and by the transitivity of the guaranteeing operator it follows from $R \vdash R\rhd$ that $R \vdash \lhd R\rhd$. By symmetry it must also be a left type of R. It is moreover the $\vdash$-least per which is both a left and a right type, because $R : T \sim T \;\Rightarrow\; T \dashv R \,\&\, R \vdash T \;\Rightarrow\; T \dashv \lhd R \,\&\, R\rhd \vdash T \;\Rightarrow\; \lhd R \vdash T \,\&\, R\rhd \vdash T \;\Rightarrow\; \lhd R\rhd \vdash T$. This shows that $\lhd R\rhd$ exactly determines the left-and-right types of R, because $\lhd R\rhd \vdash T \;\Leftrightarrow\; R : T \sim T$.

Unsurprisingly, we now have yet another characterisation of pers: a relation R is a per if and only if $R = \lhd R\rhd$. As before if $R = \lhd R\rhd$ it must necessarily be a per, and conversely if R is a per we know that $R = \lhd R = R\rhd$, so $\lhd R\rhd = \lhd R \perp R\rhd = R \perp R = R$.

At last, nothing much

We can now define R^0 to be $\lhd R\rhd$ and know that it is a per, and that it is the best ($\vdash$-least) type for which $R^{n+m} = R^n\,;R^m$ can be extended to allow n or m to be zero. Moreover, it has the pleasant property that $T^0 = T$ for any per T, and so that $(R^0)^0 = R^0$, and by the symmetry of its definition $(R^{-1})^0 = R^0$.

In general, $(R^n)^0$ is not the same as R^0 but it is a refinement of it, which is to say that $(R^n)^0 \vdash R^0$. Since R^0 is a per, $R^0 : R^0 \sim R^0$, but $R^0 : R^0 \sim R^0 \Leftrightarrow R : R^0 \sim R^0 \Rightarrow R^n : R^0 \sim R^0 \Rightarrow (R^n)^0 \vdash R^0$. To see that $(R^n)^0 \neq R^0$, in particular consider the relation $R = \Sigma{>}$ from above. This is the union of two overlapping full relations, $\{\langle a,a\rangle,\langle a,b\rangle,\langle b,a\rangle,\langle b,b\rangle\} \cap \{\langle b,b\rangle,\langle b,c\rangle,\langle c,b\rangle,\langle c,c\rangle\}$, so $R^0 = \{\langle a,a\rangle,\langle b,b\rangle,\langle c,c\rangle\}$ is an identity relation. However R^2 is the full relation on a, b and c, and so a per, and $(R^2)^0 = R^2 \neq R^0$.

We are finally able to justify the strange claim that $R^0 = \mathbin{;}\{T \mid R : T \sim T\}$. What it appears we mean by it is that $R^0 \in \{T \mid R : T \sim T\}$, and that since it can absorb any of the other elements composed on either side, any composition (of a reasonable number) of members of the set must be equal to R^0. We do not think we would want to press the point though.

A very brief extensive treatment

The enthusiastic reader may care to find a neat way of showing that

$$
\begin{aligned}
R\rhd &= \{x, y \mid (\exists z.\, z\,R\,x \,\&\, z\,R\,y) \,\&\, (\forall z.\, z\,R\,x \Leftrightarrow z\,R\,y)\} \\
\lhd R\rhd &= \{x, y \mid ((\exists z.\, z\,R\,x \,\&\, z\,R\,y) \vee (\exists z.\, x\,R\,z \,\&\, y\,R\,z)) \\
&\qquad \&\ ((\forall z.\, z\,R\,x \Leftrightarrow z\,R\,y) \,\&\, (\forall z.\, x\,R\,z \Leftrightarrow y\,R\,z))\}
\end{aligned}
$$

Terms of the form $\forall z.\, z\,R\,x \Leftrightarrow z\,R\,y$ can be expressed in a pointless way by using *quotients* (or *factors*), but as it stands, the only approach that we know to work is to demonstrate equal extension by pairs of pointwise arguments.

Tall and thin, short and fat

We have dealt with empty compositions. Similar concerns arise in the case of rows and columns of components that contain no components. The relation $\mathsf{row}\, R$ can be defined by

$$
\begin{aligned}
&\langle a, x\rangle\,(\mathsf{row}\,R)\,\langle y, b\rangle \\
&\quad = \exists n > 0.\, \#x = \#y = n \ \& \\
&\qquad \exists z.\, \#z = n+1 \ \&\ a = z_0 \ \&\ (\forall i.\, \langle z_i, x_i\rangle\, R\, \langle y_i, z_{i+1}\rangle) \ \&\ z_n = b
\end{aligned}
$$

and represents a row of n components connected by a horizontal 'daisy-chain', at least provided we restrict the range of n to be strictly positive. Analogously, the relation $\mathsf{col}\,R = (\mathsf{row}\,R^{-1})^{-1}$ represents a column of components.

Two properties of row used often in calculations are

$$\mathsf{snd\,map}\,R\,;\mathsf{row}\,S\,;\mathsf{fst\,map}\,T \;=\; \mathsf{row}(\mathsf{snd}\,R\,;S\,;\mathsf{fst}\,T)$$

which we call 'pushing maps into row' and

$$\mathsf{fst}\,R\,;S = T\,;\mathsf{snd}\,R \;\Rightarrow\; \mathsf{fst}\,R\,;\mathsf{row}\,S = \mathsf{row}\,T\,;\mathsf{snd}\,R$$

which is 'row induction'. There are also mutual distribution (or 'abiding', above-and-beside-ing) laws: $\mathsf{row}(R \updownarrow S) = \mathsf{row}\,R \updownarrow \mathsf{row}\,S$ and $\mathsf{col}(R \leftrightarrow S) = \mathsf{col}\,R \leftrightarrow \mathsf{col}\,S$, where $\leftrightarrow$ (beside) and $\updownarrow$ (below) are the binary analogues of rows and columns respectively; and then by induction it follows that $\mathsf{row}\,\mathsf{col}\,R =$ $\mathsf{col}\,\mathsf{row}\,R$.

Were we to extend the definitions of row and column to allow n to be zero, we would find that for any R and any x, it would be the case that $\langle x, \langle\,\rangle\rangle(\mathsf{row}\,R)$ $\langle\langle\,\rangle, x\rangle$. This is a vanishingly narrow row of components, so narrow that it has no R components in it, and connects the wire at the left – the first component of the pair in the domain – to the wire at the right – the second component of the pair in the range – as if by an identity relation. The empty row does not depend on the component R, but behaves like the relation $\mathsf{snd}\,0\,;\,\mathit{swp} =$ $\mathsf{snd}\,0\,;\,\mathit{swp}\,;\,\mathsf{fst}\,0 = \mathit{swp}\,;\,\mathsf{fst}\,0$, where 0 is the identity relation on the singleton type containing only $\langle\,\rangle$. If the universal identity relation is to be considered to be unimplementably large, then so surely is this relation.

More seriously, it is no longer the case that $\mathsf{row}(R \updownarrow S) = \mathsf{row}\,R \updownarrow \mathsf{row}\,S$. In particular $\mathsf{snd}\,0\,;\mathsf{row}(R \updownarrow S) = \mathsf{snd}\,0\,;\mathit{swp} \neq \mathsf{snd}\,0\,;(\mathit{swp} \updownarrow \mathit{swp}) = \mathsf{snd}\,0\,;(\mathsf{row}\,R \updownarrow$ $\mathsf{row}\,S)$. This is because $\langle x, \langle\,\rangle\rangle\,(\mathsf{row}\,R \updownarrow \mathsf{row}\,S)\,\langle\langle\,\rangle, x\rangle$ only if x is a pair.

The immediate reaction to this must surely be that we can deal with empty rows, and empty columns, in a way similar to that which worked for empty repeated compositions. Perhaps we should expect the empty row of Rs to be $[T, 0]\,;\mathit{swp} = [T, 0]\,;\mathit{swp}\,;[0, T] = \mathit{swp}\,;[0, T]$ for some type T which captures the essence of R, such as $T = (\pi_1^{-1}\,;R\,;\pi_2)^0$. That would indeed solve the problem with row and below, because $(\pi_1^{-1}\,;(R \updownarrow S)\,;\pi_2)^0$ is certainly a relation between pairs only.

Sadly this unexpectedly breaks the apparently much more fundamental rule for pushing maps into rows. Consider the difference between $\mathsf{snd}\,\mathsf{map}\,\mathit{empty}\,;$ $\mathsf{row}\,R$ and $\mathsf{row}(\mathsf{snd}\,\mathit{empty}\,;R)$, where empty is the entirely empty relation, which relates nothing. The component $\mathsf{snd}\,\mathit{empty}\,;R$ is the empty relation, whatever the value of R, so we conclude that $\mathsf{row}(\mathsf{snd}\,\mathit{empty}\,;R) = \mathsf{row}\,\mathit{empty}$ must be independent of R.

In contrast, $\mathsf{map}\,\mathit{empty}$ is not entirely empty. Being a map, it must be a relation between lists of equal length, and it certainly relates no lists of positive length. However it does relate $\langle\,\rangle$ to itself, for there seems no way of defining $\mathsf{map}\,R$ so that $0\,;\mathsf{map}\,R$ is any different from 0. We are forced to conclude that $\mathsf{map}\,\mathit{empty} = 0$ and so $\mathsf{snd}\,\mathsf{map}\,\mathit{empty}\,;\,\mathsf{row}\,R = \mathsf{snd}\,0\,;\,\mathsf{row}\,R$ is a vanishingly narrow row of Rs.

If the rule for pushing maps into rows is to remain true we can conclude that $\mathsf{snd}\,0\,;\mathsf{row}\,R = \mathsf{row}\,\mathit{empty}$, that is that a zero-width row of Rs must be equal in value to a particular Ruby expression which cannot depend on R.

There seems to be no simple way out of the incompatibility of the abiding properties and the pushing of maps into rows. What we have run into here is an essential limitation of the untyped language. All our 'type judgements' are assessments of the values of expressions, and all empty maps are equal, just as all empty sets are equal. From that it seems to have followed that all empty rows need to be equal as well.

Connections

It is a little unsatisfactory that so many of the few proofs in this paper are done by arguments about points. It would be nice to be able to stand a little further back from the graphs of the relations, and perform the proofs by quoting a small number of properties of the relations themselves. Those pointwise proofs that remain seem to rely on the properties of the diagonal of a relation, that is its largest intersection with an equality relation. If necessary we would be happy enough to axiomatise a diagonalisation operation, but are reluctant to add 'the identity' and take intersections, as they do in the Eindhoven school. The diagonal of a relation would also cover the provisos that every relation has at least one right (and left) type – because the diagonal of its right (or left) view would do; and that every pair of pers has a common right type – because the union of their diagonals would do.

Hutton and Voermans [2] deals with many of the ideas about types, although in a framework in which they are happy to deal explicitly with the identity relation (for them, I) and even bigger things like the total relation on everything. They are able to use the identity relation to define (the identity on) the domain, or range, of a relation. The guarantees relation appears in their work only as a partial order on pers. Although our notations are slightly different our interpretation of pers as types is identical to theirs.

Since presenting this paper we have been shown an unpublished paper by Ed Voermans and Jaap van der Woude [3] which contains some of the results, and in particular the existence of more general least upper bounds (which are greatest lower bounds with respect to their inverted order).

The conflicting requirements of the desirable properties of empty rows and empty columns is reminiscent of the essentially partial 'above' and 'beside' operations in squiggol. Alan Jeffreys [4] shows that making horizontal and vertical concatenation of arrays total operations – rather than ones that apply only to arrays of matching sizes – inevitably leads to trouble.

References

[1] Geraint Jones and Mary Sheeran, *Circuit design in Ruby* in Jørgen Staunstrup (ed.), *Formal methods for VLSI design*, North-Holland, 1990. pp. 13–70.

[2] Graham Hutton and Ed Voermans, *A calculational theory of pers as types*, University of Glasgow Computing Science Research Report 1992/R1.

[3] Ed Voermans and Jaap van der Woude, *Relational theory of datatypes: the per version*, (unpublished).

[4] Alan Jeffreys, *Soft arrays*, in *The Squiggolist*, 1.4, June 1990.