A Single Complete Relational Rule for Coalgebraic Refinement

César J. Rodrigues¹ J. N. Oliveira² Luis S. Barbosa³

Dep. Informatics & CCTC, Minho University, Portugal

Abstract

A transition system can be presented either as a binary relation or as a coalgebra for the powerset functor, each representation being obtained from the other by transposition. More generally, a coalgebra for a functor \( F \) generalises transition systems in the sense that a shape for transitions is determined by \( F \), typically encoding a signature of methods and observers. This paper explores such a duality to frame in purely relational terms coalgebraic refinement, showing that relational (data) refinement of transition relations, in its two variants, downward and upward (functional) simulations, is equivalent to coalgebraic refinement based on backward and forward morphisms, respectively. Going deeper, it is also shown that downward simulation provides a complete relational rule to prove coalgebraic refinement. With such a single rule the paper defines a pre-ordered calculus for refinement of coalgebras, with bisimilarity as the induced equivalence. The calculus is monotonic with respect to the main relational operators and arbitrary relator \( F \), therefore providing a framework for structural reasoning about refinement.

Keywords: Transition systems, coalgebraic refinement.

1 Introduction

Suppose one needs to replace part of a system by another: How safe is such a replacement? The classical answer in a process algebra context, namely after Milner landmark work in CCS [27], is well-known: they should be bisimilar. Typical concurrent implementations of non-deterministic specifications, for example, are witnessed by bisimilarity. Informally, two systems are bisimilar if they behave in such a way, that an observer cannot distinguish between them and this inability is maintained along the systems evolution. This is captured by a relation among their state spaces which maintains equality of observed effects along the system evolution, i.e., a bisimulation.

¹ Email: cjrodi.uminho.pt
² Email: jno@di.uminho.pt
³ Email: lsb@di.uminho.pt

1571-0661/$ – see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.entcs.2009.12.014
The concept of bisimulation is nowadays discussed in the broader context of coalgebras [37,2], which generalises (labelled) transition systems (LTS). Formally, given an endofunctor $F$ over $\text{Set}$ a $F$-coalgebra or $F$-system is a function $\alpha : S \xrightarrow{\alpha} S$. Set $S$ is referred to as the coalgebra carrier or set of states. Function $\alpha_S$ is the $F$-transition structure (or dynamics) of the system.

Consider, for illustrating purposes, a LTS represented by a transition relation $\text{Act} \times P \xrightarrow{\text{A}} P$ involving a set of behaviours (or processes) $P$ and a set of observations and actions $\text{Act}$. Relation $R$ can be transposed to a function $\overline{R}$ of type $(P \xleftarrow{\text{A}} P)$ so that, for all $\alpha \in \text{Act}$, $q(\overline{R} \alpha)p \equiv (\alpha, q) R p$ holds. Wherever $R$ is implicit in the context, notation $q^\alpha \overline{A} p$ (meaning $q$ is reachable from $p$ via action or observation $\alpha$) abbreviates $q(\overline{R} \alpha)p$. Clearly, the power-transpose of a given LTS $R$, $\mathcal{P}A \times P \xleftarrow{\mathcal{A}R} P$, is a $B$-coalgebra for

$$B(X) = \mathcal{P}A \times X$$

In general, a coalgebra for an arbitrary functor $F$ corresponds to a transition system of shape $F$ and the notion of bisimulation acquires a shape as well.

It is not surprising that, along the last decade, coalgebra theory emerged as a common framework to describe ‘state based’, dynamical systems. Its study along the lines of Universal Algebra, was initiated by J. Rutten in [37]. There are a number of tutorials (see, eg., [23], [17] or [2]) to which the interested reader can be referred to. The proceedings of the Coalgebraic Methods in Computer Science workshop series, initiated in 1998, document current research ranging from the study of concrete coalgebras over different base categories [38] to the development of $\text{Set}$-independent, i.e., purely categorical, presentations of coalgebra theory (see, among others [38,32]), from coalgebraic logic (eg., [28]) to applications. Application examples range from automata [36] to objects [33,21], from process semantics [39] to hybrid transition systems [20]. B. Jacobs and his group, following earlier work by H. Reichel [33,19] have coined the term coalgebraic specification [22,24,35] to denote a style of axiomatic specification involving equations up to bisimilarity acting as constraints on the observable behaviour. Alternatively, the direct use of coalgebraic structures as models of software components is discussed in [6,12].

In both cases, the quest for a suitable notion of refinement for coalgebras emerged as a main concern. The task is demanding because the classical ‘recipe’ to identify a refinement situation (look for an abstraction function to witness it), does not apply. Formally such ‘recipe’ corresponds to looking for a morphism in the relevant category, from the ‘concrete’ to the ‘abstract’ model such that the latter can be recovered from the former up to a suitable notion of equivalence. The problem is that coalgebra morphisms entail bisimilarity, whereas one would expect to have an order relation expressing some sort of behaviour simulation.

The problem was addressed previously by the second and third authors in a series of papers including [25,26] and [7]. In these references refinement is captured by the existence of some form of weak coalgebra morphism (just as bisimulation
amounts to the existence of a standard morphism) with respect to a particular refinement preorder. The latter, on its turn, exploits the structure of the coalgebra dynamics in a number of different ways (leading, correspondingly, to a number of refinement preorders). Finally, such preorders can be used in two dual ways referred to in [25] as forward or backward refinement. In broad terms, the former generalises the usual axis of non determinism reduction in a functorial way, whereas the latter corresponds to a similar functorial generalisation of definition increase.

In this context, the contribution of this paper is an attempt to frame in purely relational terms coalgebraic refinement, showing that relational (data) refinement of transition relations, in its two variants, downward and upward (functional) simulations, is equivalent to coalgebraic refinement for the structural inclusion preorder (i.e., the quotient of structural membership as explained below), based on backward and forward morphisms, respectively. Moreover, we show that downward simulation is enough as a single complete relational rule to prove this sort of coalgebraic refinement. Based on such a rule the paper introduces a pre-ordered calculus for the refinement of coalgebras, with bisimilarity as the induced equivalence. The calculus is monotonic with respect to the main relational operators and arbitrary relator $F$, therefore providing a framework for structural reasoning about refinement.

The following section contains a glimpse of the relational calculus [1,8,4] used in the paper. Then, sections 3 and 4 introduce the paper’s technical contributions as detailed above. Finally, section 5 concludes and provides a few pointers to current work.

2 Overview of the relational calculus

Relations.

Let $B \xleftarrow{R} A$ denote a binary relation on datatypes $A$ (source) and $B$ (target). We write $bRa$ to mean that pair $(b, a)$ is in $R$. The underlying partial order on relations will be written $R \subseteq S$, meaning that $S$ is either more defined or less deterministic than $R$, that is, $R \subseteq S \equiv bRa \Rightarrow bSa$ for all $a, b$. Equality on relations can be established by $\subseteq$-antisymmetry: $R = S \equiv R \subseteq S \wedge S \subseteq R$.

Relations can be combined by three basic operators: composition ($R \cdot S$), converse ($R^\circ$) and meet ($R \cap S$). $R^\circ$ is the relation such that $a(R^\circ)b$ iff $bRa$ holds. Meet corresponds to set-theoretical intersection and composition is defined in the usual way: $b(R \cdot S)c$ holds wherever there exists some mediating $a \in A$ such that $bRa \wedge aSc$. Everywhere $T = R \cdot S$ holds, the replacement of $T$ by $R \cdot S$ will be referred to as a factorization and that of $R \cdot S$ by $T$ as fusion.

Coreflexives.

Some standard terminology arises from the $id$ relation: a (endo)relation $A \xleftarrow{R} A$ (often called an order) will be referred to as reflexive iff $id_A \subseteq R$ holds and as coreflexive iff $R \subseteq id_A$ holds. As a rule, subscripts are dropped wherever types are implicit or easy to infer.
Coreflexive relations are fragments of the identity relation which can be used to model predicates or sets. The meaning of a predicate $p$ is the coreflexive $[p]$ such that $b[p]a \equiv (b = a) \land (p a)$, that is, the relation that maps every $a$ which satisfies $p$ (and only such $a$) onto itself. The meaning of a set $S \subseteq A$ is $[\lambda a.a \in S]$, that is, $b[S]a \equiv (b = a) \land a \in S$. Wherever clear from the context, we will omit the $[]$ brackets.

Orders.

Preorders are reflexive, transitive relations, where $R$ is transitive iff $R \cdot R \subseteq R$ holds. Partial orders are anti-symmetric preorders, where $R$ is anti-symmetric wherever $R \cap R^\circ \subseteq \text{id}$ holds. A preorder $R$ is an equivalence if it is symmetric, that is, if $R = R^\circ$. A total order $R$ is a connected preorder, where $R$ is connected iff $R \cup R^\circ = \top$ holds. $\cup$ is the join of two relations and $\top$ is the largest relation of its type. Its dual is $\bot$, the smallest such relation.

Taxonomy.

Converse is of paramount importance in establishing a wider taxonomy of binary relations. Let us first define two derived operators, the so-called kernel of a relation $\ker R \overset{\text{def}}{=} R^\circ \cdot R$ (1) and its image 

$\img R \overset{\text{def}}{=} \ker (R^\circ)$ (2)

An alternative to (2) is to define $\img R = R \cdot R^\circ$, since converse distributes over composition,

$(R \cdot S)^\circ = S^\circ \cdot R^\circ$ (3)

and is involutive, that is,

$(R^\circ)^\circ = R$ (4)

Kernel and image lead to the following terminology: a relation $R$ is said to be entire (or total) iff its kernel is reflexive; or simple (or functional) iff its image is coreflexive. Dually, $R$ is surjective iff $R^\circ$ is entire, and $R$ is injective iff $R^\circ$ is simple.

Functions.

A relation is a function iff it is both simple and entire. Functions will be denoted by lowercase letters ($f$, $g$, etc.) and are such that $bfa$ means $b = f a$. Function converses enjoy a number of properties of which the following is singled out because of its role in pointwise-pointfree conversion [3]:

$b(f^\circ \cdot R \cdot g)a \equiv (f \ b)R(g \ a)$ (5)

The overall taxonomy of binary relations further to the standard classification, includes representations and abstractions. These are classes of relations useful in data-refinement [29]. Because of $\subseteq$-antisymmetry, $\img S = \text{id}$ wherever $S$ is an abstraction and $\ker R = \text{id}$ wherever $R$ is a representation. This ensures that no
confusion arises in a representation and that all abstract data are reachable by an abstraction (no junk).

Isomorphisms are functions, abstractions and representations at the same time. A particular isomorphism is \( \text{id} \), which also is the smallest equivalence relation on a particular data domain. So, \( b \text{id} a \) means the same as \( b = a \). The topmost relation \( \top \) is the largest equivalence relation of its type. In fact, it is easy to show that \( \top \) is the kernel of every constant function, \( 1 \leftarrow A \) included, where function \( ! \) is the unique function of its type, where 1 denotes the singleton data domain.

Relators.

A relator [5] is a concept which extends functors to relations: \( F A \) describes a parametric type while \( F R \) is a relation from \( F A \) to \( F B \) provided \( R \) is a relation from \( A \) to \( B \). Relators are monotone and distribute over composition, converse and the identity:

\[
\begin{align*}
F (R \cdot S) &= (F R) \cdot (F S) \\
F (R^o) &= (F R)^o \\
F \text{id} &= \text{id}
\end{align*}
\]

(6) (7) (8)

The most simple relators are the identity relator \( \text{id} \), which is such that \( \text{id} A = A \) and \( \text{id} R = R \), and the constant relator \( K \) (for a particular concrete data type \( K \)) which is such that \( K A = K \) and \( K R = \text{id} K \).

Relators can also be multi-parametric. Two well-known examples of binary relators are product and sum,

\[
\begin{align*}
R \times S &\equiv \langle R \cdot \pi_1, S \cdot \pi_2 \rangle \\
R + S &\equiv [i_1 \cdot R, i_2 \cdot S]
\end{align*}
\]

(9) (10)

where \( \pi_1, \pi_2 \) denote the projection functions of a Cartesian product, \( i_1, i_2 \) denote the injection functions of a disjoint union, and the \text{split/either} relational combinators are defined by

\[
\begin{align*}
\langle R, S \rangle &\equiv \pi_1^o \cdot R \cap \pi_2^o \cdot S \\
[R, S] &\equiv (R \cdot i_1^o) \cup (S \cdot i_2^o)
\end{align*}
\]

(11) (12)

By putting these four kinds of relator (product, sum, identity and constant) together with fixpoint definition one is able to specify a large class of parametric structures — called \text{polynomial} — such as those implementable in Haskell. For instance, the \text{Maybe} datatype is an implementation of polynomial relator \( F = \text{id} + 1 \) (i.e. \( F A = A + 1 \)), where 1 denotes the \text{singleton} datatype.

Membership.

Recall the notion of membership from set theory. Wherever we write \( a \in x \), where \( x \) is a set, we mean a relation of type \( A \leftarrow P A \), where \( P A \) denotes the set of all subsets of \( A \).
Sentence \( a \in x \) (meaning that \( a \) belongs to \( x \) or \( a \) occurs in \( x \)) can be generalized to \( x \)’s other than sets. For instance, one may check whether a particular integer occurs in one or more leaves of a binary tree, or of any other collective or container type \( F \).

Such a generic membership relation will have type \( A \xleftarrow{\in} F A \), where \( F \) is a type parametric on \( A \). Technically, the parametricity of \( F \) is captured by regarding it as a functor.

There is more than one way to generalize \( A \xleftarrow{\in} PA \) to functors other than the powerset. For the purpose of this paper it will be enough to say that \( A \xleftarrow{\in_F} F A \), if it exists, is a lax natural transformation \([8]\), that is,

\[
\in_F \cdot F R \subseteq R \cdot \in_F
\]  

holds. Moreover, polynomial functors involving +, ×, Id and constants have membership defined inductively as follows:

\[
\begin{align*}
\in_K & \overset{\text{def}}{=} \bot \\
\in_{\text{Id}} & \overset{\text{def}}{=} \text{id} \\
\in_{F \times G} & \overset{\text{def}}{=} (\in_F \cdot \pi_1) \cup (\in_G \cdot \pi_2) \\
\in_{F + G} & \overset{\text{def}}{=} [\in_F, \in_G] \\
\in_{F \cdot G} & \overset{\text{def}}{=} \in_F \cdot \in_G
\end{align*}
\]  

(13)

(14)

(15)

(16)

(17)

(18)

3 Coalgebraic refinement as data refinement

As mentioned in the Introduction, the existence of a coalgebra morphism between two coalgebras makes them bisimilar. Recall that

**Definition 3.1** Let \((X, p : X \to F X)\) and \((Y, q : Y \to F Y)\) be coalgebras for functor \(F\). A morphism connecting \(p\) and \(q\) is a function \(h\) between their carriers such that the following diagram commutes:

\[
\begin{array}{ccc}
X & \xrightarrow{p} & F X \\
\downarrow{h} & & \downarrow{F h} \\
Y & \xrightarrow{q} & F Y
\end{array}
\]

i.e. \(q \cdot h = F h \cdot p\)

(19)

To base refinement a weaker notion of morphism was proposed in \([26]\) as follows

**Definition 3.2** [Forward (backward) morphism]

Let \(F\) be a polynomial functor on \(\text{Set}\) and consider two \(F\)-coalgebras \(\beta : F V \leftarrow V\) and \(\alpha : F U \leftarrow U\).

A forward morphism \(h : \alpha \leftarrow \beta\) with respect to a preorder \(\leq\), is a function from \(V\) to \(U\) such that

\[
F h \cdot \beta \leq \alpha \cdot h
\]

where \(\leq\) is the pointwise lifting of \(\leq\), i.e. \(f \leq g \iff f \subseteq \cdot g\), that is, \(f\) and \(g\) are such that \(f b \leq g b\), for all \(b\).
Dually, $h$ is said to be a backward morphism w.r.t $\leq$ if

$$\alpha \cdot h \leq Fh \cdot \beta$$

The two refinement concepts are parameterized by preorder $\leq$, which is characterized in [26,7] as belonging to interval $id \subseteq \leq \subseteq (\epsilon_F \setminus \epsilon_F)$. The refinement preorder $(\epsilon_F \setminus \epsilon_F)$ is known as the structural inclusion associated to functor $F$.

In the context of this definition, it is proved in [26] that

- a forward morphism $h$ preserves the transition relation corresponding to coalgebra $\beta$,

$$v' \xleftarrow{\beta} v \Rightarrow hv' \xleftarrow{\alpha} hv$$

- a backward morphism $h$ reflects the transition relation corresponding to coalgebra $\alpha$,

$$u' \xleftarrow{\alpha} hv \Rightarrow \langle \exists v' : v' \in V : v' \xleftarrow{\beta} v \land u' = hv' \rangle$$

Relational (data) refinement [13], on the other hand, is discussed in terms of two forms of simulation: downward and upward. Our first result relates such notions, in their functional variant, to those of coalgebraic forward and backward morphisms.

**Lemma 3.3 (Upward simulation and forward morphism)** Let $h$ be an upward simulation,

$$h \cdot \gamma \subseteq \alpha \cdot h$$

where $\gamma$ is the transition relation associated to coalgebra $\gamma$ and, in the same way, $\alpha$ is the transition relation associated to coalgebra $\alpha$. Then (20) is equivalent to forward morphism condition

$$Fh \cdot \gamma \leq \alpha \cdot h$$

where $\leq$ is the structural inclusion preorder.

**Proof.** The equivalence between upward simulation and forward morphism was proven in one way,

$$Fh \cdot \gamma \leq \alpha \cdot h \Rightarrow h \cdot \gamma \subseteq \alpha \cdot h$$

in [26], for all $\leq$ belonging to interval $id \subseteq \leq \subseteq (\epsilon_F \setminus \epsilon_F)$. The full equivalence is established as follows. Notice, however, the restriction of $\leq$ to the structural inclusion relation $\epsilon_F \setminus \epsilon_F$.

$$h \cdot \gamma \subseteq \alpha \cdot h$$

$$\equiv \{ \overset{\beta}{\epsilon_F} \}$$

$$h \cdot \epsilon_F \cdot \gamma \subseteq \epsilon_F \cdot \alpha \cdot h$$

$$\equiv \{ \epsilon_F \text{ is a natural transformation: } f \cdot \epsilon_F = \epsilon_F \cdot Ff \}$$

$$\epsilon_F \cdot Fh \cdot \gamma \subseteq \epsilon_F \cdot \alpha \cdot h$$

$$\equiv \{ \text{Galois connection for division: } R \cdot S \subseteq T \Leftrightarrow S \subseteq R \setminus T \}$$

$$Fh \cdot \gamma \subseteq \epsilon_F \setminus (\epsilon_F \cdot \alpha \cdot h)$$
\[
\begin{align*}
\equiv & \quad \{ \text{property } R \setminus (S \cdot f) = (R \setminus S) \cdot f \} \\
Fh \cdot \gamma \subseteq (\epsilon_F \setminus \epsilon_F) \cdot \alpha \cdot h \\
\equiv & \quad \{ \leq \epsilon_F \setminus \epsilon_F \} \\
Fh \cdot \gamma \subseteq \leq \cdot \alpha \cdot h \\
\equiv & \quad \{ \text{lifting of } \leq \} \\
Fh \cdot \gamma \leq \alpha \cdot h
\end{align*}
\]

\[\square\]

The dual result is established through a similar reasoning:

**Lemma 3.4 (Downward simulation and backward morphism)** In the conditions of the previous lemma, let \( h \) be a downward simulation,

\[\llhd \cdot h \subseteq h \cdot \alpha\] (21)

Then (21) is equivalent to forward morphism condition

\[\gamma \cdot h \leq Fh \cdot \alpha\]

where \( \leq \) is again the structural inclusion relation.

**Proof.** The equivalence between downward simulation and backward morphism has been proven in one way,

\[\gamma \cdot h \leq Fh \cdot \alpha \Rightarrow \llhd \cdot h \subseteq h \cdot \alpha\]

in [26], for all \( \leq \) belonging to interval \( id \subseteq \leq \subseteq (\epsilon_F \setminus \epsilon_F) \). A proof of the equivalence for the case where the refinement preorder is generic inclusion, follows.

\[\begin{align*}
\llhd \cdot h \subseteq h \cdot \alpha \\
\equiv & \quad \{ \beta = \epsilon_F \cdot \beta \} \\
\epsilon_F \cdot \gamma \cdot h \subseteq h \cdot \epsilon_F \cdot \alpha \\
\equiv & \quad \{ \epsilon_F \text{ is a natural transformation: } f \cdot \epsilon_F = \epsilon_F \cdot Ff \} \\
\epsilon_F \cdot \gamma \cdot h \subseteq \epsilon_F \cdot Fh \cdot \alpha \\
\equiv & \quad \{ \text{Galois connection for division: } R \cdot S \subseteq T \Leftrightarrow S \subseteq R \setminus T \} \\
\gamma \cdot h \subseteq \epsilon_F \setminus (\epsilon_F \cdot Fh \cdot \alpha) \\
\equiv & \quad \{ \text{property } R \setminus (S \cdot f) = (R \setminus S) \cdot f \} \\
\gamma \cdot h \subseteq (\epsilon_F \setminus \epsilon_F) \cdot Fh \cdot \alpha \\
\equiv & \quad \{ \leq \epsilon_F \setminus \epsilon_F \} \\
\gamma \cdot h \subseteq \leq \cdot Fh \cdot \alpha \\
\equiv & \quad \{ \text{lifting of } \leq \} \\
\gamma \cdot h \leq Fh \cdot \alpha
\end{align*}\]
These two lemmas show a close relationship among coalgebras and their relational counterparts. In fact, for any such coalgebra $\alpha$, its transition relation is easy to spell out:

$$\alpha \overset{\text{def}}{=} \subseteq F \cdot \alpha$$

(22)

Note also that a coalgebra morphism, defined in (3.1), admits the following alternative definition:

**Definition 3.5** Let $(X, p : X \rightarrow FX)$ and $(Y, q : Y \rightarrow FY)$ be coalgebras for functor $F$. A morphism connecting $p$ and $q$ is a function $h$ between their carriers such that

$$h \cdot p = q \cdot h$$

(23)

cf. the following diagram:

$$
\begin{array}{ccc}
X & \xrightarrow{\;h\;} & Y \\
p \downarrow & & \downarrow q \\
X & \xrightarrow{\;h\;} & Y
\end{array}
$$

This makes it possible to carry out coalgebraic refinement via relational methods. In the next section this is pursued even further.

## 4 A single complete rule for data refinement

### 4.1 The basic result

The title of this section is intentionally that of a paper [15] which proves (using predicate transformers) that a single complete method is enough, instead of the two standard methods of data (relational) refinement.

The following lemma establishes that for proving coalgebraic refinement a single complete data refinement method is enough. The rule is that of downward simulation, this time resorting to relations. Its proof makes use of the two shunting rules, witnessing well-known Galois connections,

$$f \cdot R \subseteq S \equiv R \subseteq f^\circ \cdot S$$

(24)

$$R \cdot f^\circ \subseteq S \equiv R \subseteq S \cdot f$$

(25)

**Lemma 4.1 (Downward Simulation)** For proving coalgebraic refinement downward simulation is enough. Actually, upward simulation given by the proof rule

$$\begin{array}{c}
h \cdot \gamma \subseteq \alpha \cdot h
\end{array}$$

(26)

can be reduced to

$$\begin{array}{c}
\gamma \cdot h^\circ \subseteq h^\circ \cdot \alpha
\end{array}$$

(27)

i.e. downward simulation for a relation which is the converse of a function.
Proof.

\( h \) is an upward simulation

\[
\begin{align*}
\equiv & \quad \{ \text{by definition of upward simulation} \} \\
\leftrightharpoons & \quad \text{shunting rules}
\end{align*}
\]

\( h \circ \) is a downward simulation

4.2 A refinement calculus

Lemma (4.1) can be used as a basis of a refinement calculus in which a given coalgebra \( \alpha \) is refined via its transition relation \( \xleftarrow{\alpha} \). Its laws are inequations of the form

\[
\gamma \preceq_R \alpha
\]

where \( R = h \) or \( R = h^\circ \) for some \( h \), meaning (21) or (27), respectively. Let us establish some of its properties.

\( \preceq \) is a preorder.

First note that \( \preceq \) is a preorder: clearly, it is reflexive (\( R := id \)),

\[
\alpha \preceq_{id} \alpha
\]

and transitive,

\[
\alpha \preceq_R \beta \land \beta \preceq_S \gamma \Rightarrow \alpha \preceq_{RS} \gamma
\]

\[
\begin{align*}
\equiv & \quad \{ \text{by definition of } \preceq \} \\
& \quad \text{monotonicity of } (\cdot S), \text{ because it is an adjoit in a Galois connection [34]} \\
& \quad \text{transitivity of inclusion}
\end{align*}
\]

The statement asserting transitivity is decomposed into the following two formulations, for
• functions

\[ \alpha \preceq_h \beta \land \beta \preceq_g \gamma \Rightarrow \alpha \preceq_{h \cdot g} \gamma \]

• converses of functions

\[ \alpha \preceq_{h^o} \beta \land \beta \preceq_{g^o} \gamma \Rightarrow \alpha \preceq_{(g \cdot h)^o} \gamma \]

Relation to bisimilarity is given by the following lemma

**Lemma 4.2 (Relation with bisimilarity)** \( \text{Preorder} \preceq \text{is related to bisimilarity by} \)

\[ \alpha \preceq_{f^o} \alpha \land \alpha \preceq_f \alpha \equiv \alpha \sim_f \alpha \]  \hspace{1cm} (28)

**Proof.**

\[ \alpha \preceq_{f^o} \alpha \land \alpha \preceq_f \alpha \]

\[ \equiv \{ \text{by definition of} \preceq \} \]

\[ \overset{\alpha}{\rightarrow} \cdot f^o \subseteq f^o \cdot \overset{\alpha}{\rightarrow} \land \overset{\alpha}{\rightarrow} \cdot f \subseteq f \cdot \overset{\alpha}{\rightarrow} \]

\[ \equiv \{ \text{shunting, cf. (24) and (25)} \} \]

\[ f \cdot \overset{\alpha}{\rightarrow} \subseteq \overset{\alpha}{\rightarrow} \cdot f \land \overset{\alpha}{\rightarrow} \cdot f \subseteq f \cdot \overset{\alpha}{\rightarrow} \]

\[ \equiv \{ \text{ping-pong: } R \subseteq S \land S \subseteq R \equiv R = S \} \]

\[ f \cdot \overset{\alpha}{\rightarrow} = \overset{\alpha}{\rightarrow} \cdot f \]

\[ \equiv \{ \text{homomorphism on } F\text{-coalgebra } \alpha, (23) \} \]

\[ F \cdot f \cdot \alpha = \alpha \cdot f \]

\[ \equiv \{ \text{(endo)homomorphism iff (functional) bisimulation} \} \]

\[ \alpha \sim_f \alpha \]

\[ \square \]

Since a homomorphism is a functional bisimulation \cite{37}, we arrive at the following corollary:

**Corollary 4.3**

\[ \alpha \preceq_{f^o} \gamma \land \gamma \preceq_f \alpha \equiv \alpha \sim_f \gamma \]  \hspace{1cm} (29)

Finally, let us prove the calculus is structural, in the sense that it is monotonic with respect to the main relational operators. For this we need to make transition relations explicit. So, instead of the preorder on coalgebras, we will resort, from now on, to the corresponding preorder on transition relations,

\[ \overset{\alpha}{\leftarrow} \subseteq R \overset{\alpha}{\rightarrow} \]

where \( R = h \) or \( R = h^o \) for some \( h \), meaning (21) or (27), respectively.
Lemma 4.4 (Monotonicity of converse) Converse is monotonic:
\[ \gamma \leftrightarrow_{h^\circ \leftarrow \alpha} \equiv \gamma^\circ \leftrightarrow_{h^\circ \leftarrow \alpha^\circ} \] (30)

Proof.
\[ \gamma \leftrightarrow_{h^\circ \leftarrow \alpha} \equiv \{ \text{by definition of } \sqsubseteq \} \]
\[ \gamma \cdot h^\circ \subseteq h^\circ \cdot \alpha \equiv \{ \text{converse: } R \subseteq S \equiv R^\circ \subseteq S^\circ \} \]
\[ h \cdot \gamma^\circ \subseteq \alpha^\circ \cdot h \equiv \{ \text{shunting, cf. (24) and (25)} \} \]
\[ \gamma^\circ \cdot h^\circ \subseteq h^\circ \cdot \alpha^\circ \equiv \{ \text{by definition of } \sqsubseteq \} \]
\[ \gamma \leftrightarrow_{h^\circ \leftarrow \alpha^\circ} \]
\[ \square \]

Note that monotonicity of converse cannot be proved when we replace \( h^\circ \) by \( h \).

Lemma 4.5 (Monotonicity of \( \bigcup \)) If \( \gamma \leftrightarrow_{R \leftarrow \alpha} \) and \( \beta \leftrightarrow_{R \leftarrow \delta} \) then
\[ (\gamma \cup \beta) \leftrightarrow_{R \leftarrow \alpha \cup \delta} \] (31)

Proof.
\[ \gamma \leftrightarrow_{R \leftarrow \alpha} \land \beta \leftrightarrow_{R \leftarrow \delta} \equiv \{ \text{definition of } \sqsubseteq \} \]
\[ \gamma \cdot R \subseteq R \cdot \alpha \land \beta \cdot R \subseteq R \cdot \delta \Rightarrow \{ \text{monotonicity of } \bigcup \} \]
\[ \gamma \cdot R \cup \beta \cdot R \subseteq R \cdot \alpha \cup \delta \equiv \{ (R \cdot) \text{ and } (\cdot R) \text{ are lower Galois adjoints [34], thus distribute over join} \} \]
\[ (\gamma \cup \beta) \cdot R \subseteq R \cdot (\alpha \cup \delta) \equiv \{ \text{definition of } \sqsubseteq \} \]
\[ \gamma \cup \beta \leftrightarrow_{R \leftarrow \alpha \cup \delta} \]
\[ \square \]

Lemma 4.6 (Monotonicity of \( \cap \)) If \( \alpha \cdot f \subseteq f^\circ \cdot \beta \) and \( \gamma \cdot f \subseteq f^\circ \cdot \delta \) then
\[ (\alpha \cap \gamma) \cdot f \subseteq f^\circ \cdot (\beta \cap \delta) \] (32)
Proof.
\[ \alpha \cdot f \subseteq f^\circ \cdot \beta \land \gamma \cdot f \subseteq f^\circ \cdot \delta \]
\[ \equiv \{ \text{monotonicity of } \cap \} \]
\[ \alpha \cdot f \cap \gamma \cdot f \subseteq f^\circ \cdot \beta \cap f^\circ \cdot \delta \]
\[ \equiv \{ (f^\circ \cdot) \text{ and } (\cdot f) \text{ are upper Galois adjoints [34], thus they distribute over meet} \} \]
\[ (\alpha \cap \gamma) \cdot f \subseteq f^\circ \cdot (\beta \cap \delta) \]
\[ \square \]

If \( f = f^\circ \) we have the following monotonicity result: if \( \alpha \subseteq f^\circ \beta \) and \( \gamma \subseteq f^\circ \delta \) then \( (\alpha \cap \gamma) \subseteq f (\beta \cap \delta) \)

Lemma 4.7 (Monotonicity of composition) If \( \gamma \subseteq R \alpha \) and \( \beta \subseteq R \delta \) then
\[
(\gamma \cdot \beta) \subseteq R (\alpha \cdot \delta)
\]  
(33)

Proof.
\[ \gamma \cdot \beta \cdot R \]
\[ \subseteq \{ \text{by hypothesis} \} \]
\[ \gamma \cdot R \cdot \delta \]
\[ \subseteq \{ \text{by hypothesis} \} \]
\[ R \cdot \alpha \cdot \delta \]
\[ \square \]

Finally, for an arbitrary relator \( F \),

Lemma 4.8 (F-monotonicity) If \( \gamma \subseteq R \alpha \) then
\[
F \gamma \subseteq F R \ F \alpha
\]  
(34)

Proof.
\[ \gamma \subseteq R \alpha \]
\[ \equiv \{ \text{by definition of } \subseteq \} \]
\[ \gamma \cdot R \subseteq R \cdot \alpha \]
\[ \Rightarrow \{ \text{F-monotonicity} \} \]
\[ F(\gamma \cdot R) \subseteq F(R \cdot \alpha) \]
\[ \Rightarrow \{ \text{F distributes over composition} \} \]
\[ (F \gamma) \cdot FR \subseteq FR \cdot F \alpha \]
\[ F \xrightarrow{\gamma} \sqsubseteq_{FR} F \xleftarrow{\alpha} \]

\[\Box\]

**Corollary 4.9** If \( \gamma \xleftarrow{\bullet} \sqsubseteq_{R} \alpha \xleftarrow{\bullet} \) and \( \beta \xleftarrow{\bullet} \sqsubseteq_{S} \delta \xleftarrow{\bullet} \) then
\[
(\gamma \times \beta) \sqsubseteq_{R \times S} (\alpha \times \delta) \quad (35)
\]
\[
(\gamma + \beta) \sqsubseteq_{R+S} (\alpha + \delta) \quad (36)
\]

This calculus avoids the need to resort to complicated downward simulations. Instead, we just need to know that \( \text{id} \) is a downward simulation and order the transition relations by (converse of) inclusion.

Let’s see what this means. We know that the converse of inclusion is algorithmic refinement for entire relations [34,31]. It also seems reasonable to assume that we are dealing with entire relations, the totalized (transposed [34,30]) versions of every relation. So we have found a simple method of refinement over transition relations which justifies the option of doing coalgebraic refinement indirectly by downward simulation of transition relations.

### 4.3 An example

Consider the transition system \( S \) depicted below, whose transition relation is given by \( \alpha \rightarrow \).

\[
\begin{array}{ccc}
  s_0 \xrightarrow{a} s_1 \xrightarrow{a} \ldots \\
  \downarrow \quad \quad \downarrow \\
  s'_0 \quad s'_1
\end{array}
\]

Suppose one wants to calculate an implementation \( \gamma \rightarrow \) such that:
\[
\gamma \rightarrow \subseteq \alpha \rightarrow 
\]

corresponding to downward simulation \( \text{id} \):
\[
\gamma \cdot \text{id} \subseteq \text{id} \cdot \alpha \rightarrow 
\]

that is, to the following inequation,
\[
\gamma \leq_{\text{id}} \alpha
\]

We assume that transition relations are entire (the obvious totalization is assumed), so the implementation is done by reduction of non-determinism. An obvious implementation at the level of transition relations is obtained as follows,
\[
\gamma \rightarrow = \alpha \rightarrow -\{(s_i, s'_i) | i \geq 0\}
\]

To transition relation \( \gamma \rightarrow \) one associates the obvious LTS \( R \), a completely deterministic one, depicted as follows:
The implementation coalgebra $\gamma$, the power transpose of LTS $R$, is such that
\[
\gamma \cdot id \leq F \cdot id \cdot \alpha \tag{40}
\]
according to (38), Lemma 3.4 and $F = PA \times Id$. Inequation (40) simplifies to:
\[
\gamma \leq \alpha \tag{41}
\]
since $F \cdot id = id$ as $F$, is a functor, and preserves therefore identities.

5 Conclusion and Future Work

In this paper we have shown that coalgebraic refinement, for the generic inclusion order, as witnessed by a forward morphism or a backward morphism [26,7], is equivalent to relational (data) refinement [13,9], as witnessed by an upward simulation or a downward simulation, respectively. Related work includes comparisons between data (relational) refinement [13] and process refinement [18]. The most significant examples (see e.g. [11,10,14]) arise in the context of the unification between refinement in Z and in CSP.

We have also shown that a single complete rule for data refinement is enough to prove coalgebraic refinement (cf. the title of [15]). This is because an upward simulation $h$ is equivalent to a downward simulation $h^\circ$. We defined a preorder $\preceq$ over coalgebras, which witnesses the downward simulations, and proved that the corresponding preorder $\subseteq$ over transition relations admits structural reasoning and is monotonic relative to the main relational operators. Therefore, we don’t need to define complicated downward simulations. It is enough to note that $id$ is a downward simulation and order the transition relations by (converse of) inclusion, i.e. algorithmic refinement for entire relations [34,31].

In summary, **downward simulation** is the refinement concept which unifies algebraic and coalgebraic implementation, and sequential and concurrent computation. However, our proofs are based on the generic inclusion preorder which is the greatest preorder admissible in coalgebraic refinement, $id$ being the least one, cf. [26,7]. The prospect of extending these results to an arbitrary refinement preorder $\leq$ in the interval $id \subseteq \leq \subseteq (\in \tau \setminus \in \tau)$ is a topic for future research.

On the other hand, there is no coalgebraic counterpart to the factorization of the standard algorithmic refinement partial order studied in [34,31], after the work of Groves [16]. Such a generalization is another topic for future research.

References


