Available online at www.sciencedirect.com

ScienceDirect

ELSEVIER

Electronic Notes in Theoretical Computer Science 259 (2009) 3–19

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

A Single Complete Relational Rule for Coalgebraic Refinement

César J. Rodrigues[1] J. N. Oliveira[2] Luis S. Barbosa[3]

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: cjr@di.uminho.pt

[2] Email: jno@di.uminho.pt

[3] Email: lsb@di.uminho.pt

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 *Set* a F-coalgebra or F-system is a function $\mathsf{F}\, S \xleftarrow{\alpha_S} S$. Set S is referred to as the coalgebra *carrier* or set of *states*. Function α_S is the F-transition structure (or dynamics) of the system.

Consider, for illustrating purposes, a LTS represented by a transition relation $Act \times P \xleftarrow{R} P$ involving a set of *behaviours* (or *processes*) P and a set of *observations* and *actions* Act. Relation R can be transposed to a function $\overline{R}$ of type $(P \longleftarrow P)^{Act}$ so that, for all $\alpha \in Act$,

$$q(\overline{R}\ \alpha)p \equiv (\alpha, q)\ R\ p$$

holds. Wherever R is implicit in the context, notation $q \overset{\alpha}{\leftarrow} p$ (meaning *q is reachable from p via action or observation α*) abbreviates $q(\overline{R}\ \alpha)p$. Clearly, the power-transpose of a given LTS R, $\mathcal{P} A \times P \xleftarrow{\Lambda R} P$, is a B-coalgebra for

$$\mathsf{B}\ X = \mathcal{P} A \times X$$

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 *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°) and meet $(R \cap S)$. R° 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 $b[\![p]\!]a \equiv (b = a) \wedge (p\ a)$, 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, $b[\![S]\!]a \equiv (b = a) \wedge a \in S$. 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 $R \cap R^\circ \subseteq id$ 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

$$ker\, R \stackrel{\mathrm{def}}{=} R^\circ \cdot R \tag{1}$$

and its *image*

$$img\, R \stackrel{\mathrm{def}}{=} ker\, (R^\circ) \tag{2}$$

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

$$(R \cdot S)^\circ = S^\circ \cdot R^\circ \tag{3}$$

and is involutive, that is,

$$(R^\circ)^\circ = R \tag{4}$$

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° is entire, and R is *injective* iff R° 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 rle in pointwise-pointfree conversion [3] :

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

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, $img\, S = id$ wherever S is an *abstraction* and $ker\, R = id$ 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 id, which also is the smallest equivalence relation on a particular data domain. So, $b \; id \; a$ 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, $1 \xleftarrow{!} A$ 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: $\mathsf{F}\,A$ describes a parametric type while $\mathsf{F}\,R$ is a relation from $\mathsf{F}\,A$ to $\mathsf{F}\,B$ provided R is a relation from A to B. Relators are monotone and distribute over composition, converse and the identity:

$$\mathsf{F}\,(R \cdot S) = (\mathsf{F}\,R) \cdot (\mathsf{F}\,S) \tag{6}$$

$$\mathsf{F}\,(R^\circ) = (\mathsf{F}\,R)^\circ \tag{7}$$

$$\mathsf{F}\,id = id \tag{8}$$

The most simple relators are the *identity* relator Id, which is such that $\mathsf{Id}\;A = A$ and $\mathsf{Id}\;R = R$, and the *constant* relator K (for a particular concrete data type K) which is such that $\mathsf{K}\;A = K$ and $\mathsf{K}\;R = id_K$.

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

$$R \times S \stackrel{\mathrm{def}}{=} \langle R \cdot \pi_1, S \cdot \pi_2 \rangle \tag{9}$$

$$R + S \stackrel{\mathrm{def}}{=} [i_1 \cdot R, i_2 \cdot S] \tag{10}$$

where π_1, π_2 denote the projection functions of a Cartesian product, i_1, i_2 denote the injection functions of a disjoint union, and the *split/either* relational combinators are defined by

$$\langle R, S \rangle \stackrel{\mathrm{def}}{=} \pi_1^\circ \cdot R \cap \pi_2^\circ \cdot S \tag{11}$$

$$[R, S] \stackrel{\mathrm{def}}{=} (R \cdot i_1^\circ) \cup (S \cdot i_2^\circ) \tag{12}$$

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 *polynomial* — such as those implementable in Haskell. For instance, the *Maybe* datatype is an implementation of polynomial relator $\mathsf{F} = \mathsf{Id} + 1$ (i.e. $\mathsf{F}\,A = A + 1$), where 1 denotes the *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 $A \xleftarrow{\in} \mathcal{P}A$, where $\mathcal{P}A$ 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 $A \xleftarrow{\in} \mathsf{F}\,A$, 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 $A \xleftarrow{\in} \mathcal{P}A$ to functors other than the powerset. For the purpose of this paper it will be enough to say that $A \xleftarrow{\in_\mathsf{F}} \mathsf{F}\,A$, if it exists, is a *lax natural transformation* [8], that is,

$$\in_\mathsf{F} \cdot \mathsf{F}\,R \subseteq R \cdot \in_\mathsf{F} \tag{13}$$

holds. Moreover, *polynomial* functors involving $+, \times, \mathsf{Id}$ and constants have membership defined inductively as follows:

$$\in_\mathsf{K} \stackrel{\text{def}}{=} \bot \tag{14}$$

$$\in_\mathsf{Id} \stackrel{\text{def}}{=} id \tag{15}$$

$$\in_{\mathsf{F}\times\mathsf{G}} \stackrel{\text{def}}{=} (\in_\mathsf{F} \cdot \pi_1) \cup (\in_\mathsf{G} \cdot \pi_2) \tag{16}$$

$$\in_{\mathsf{F}+\mathsf{G}} \stackrel{\text{def}}{=} [\in_\mathsf{F}, \in_\mathsf{G}] \tag{17}$$

$$\in_{\mathsf{F}\cdot\mathsf{G}} \stackrel{\text{def}}{=} \in_\mathsf{F} \cdot \in_\mathsf{G} \tag{18}$$

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 $(X, p : X \longrightarrow \mathsf{F}\,X)$ and $(Y, q : Y \longrightarrow \mathsf{F}\,Y)$ be *coalgebras* for functor F. A *morphism* connecting p and q is a function h between their carriers such that the following diagram commutes:

$$\begin{tikzcd} X \arrow[r, "p"] \arrow[d, "h"'] & \mathsf{F}\,X \arrow[d, "\mathsf{F}\,h"] \\ Y \arrow[r, "q"'] & \mathsf{F}\,Y \end{tikzcd} \qquad \text{i.e.} \qquad q \cdot h = \mathsf{F}\,h \cdot p \tag{19}$$

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 *Set* and consider two F-coalgebras $\beta : \mathsf{F}V \longleftarrow V$ and $\alpha : \mathsf{F}U \longleftarrow U$.

A *forward morphism* $h : \alpha \longleftarrow \beta$ with *respect to a preorder* $\leq$, is a function from V to U such that

$$\mathsf{F}h \cdot \beta \mathrel{\dot{\leq}} \alpha \cdot h$$

where $\dot{\leq}$ is the pointwise lifting of $\leq$, i.e. $f \mathrel{\dot{\leq}} g \Leftrightarrow f \subseteq \leq \cdot g$, 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

$$\alpha \cdot h \mathrel{\dot{\leq}} \mathsf{F}h \cdot \beta$$

The two refinement concepts are parameterized by preorder $\leq$, which is characterized in [26,7] as belonging to interval $id \subseteq \leq \subseteq (\in_{\mathsf{F}} \backslash \in_{\mathsf{F}})$. The refinement preorder $(\in_{\mathsf{F}} \backslash \in_{\mathsf{F}})$ 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 β,
$$v' \overset{\beta}{\longleftarrow} v \Rightarrow hv' \overset{\alpha}{\longleftarrow} hv$$
- a backward morphism h reflects the transition relation corresponding to coalgebra α,
$$u' \overset{\alpha}{\longleftarrow} hv \Rightarrow \langle \exists\ v'\ :\ v' \in V :\ v' \overset{\beta}{\longleftarrow} v \wedge u' = hv' \rangle$$

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 foward and backward morphisms.

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

$$h \cdot \overset{\gamma}{\leftarrow} \subseteq \overset{\alpha}{\leftarrow} \cdot h \tag{20}$$

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

$$\mathsf{F}h \cdot \gamma \mathrel{\dot{\leq}} \alpha \cdot h$$

where $\leq$ is the structural inclusion preorder.

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

$$\mathsf{F}h \cdot \gamma \mathrel{\dot{\leq}} \alpha \cdot h \Rightarrow h \cdot \overset{\gamma}{\leftarrow} \subseteq \overset{\alpha}{\leftarrow} \cdot h$$

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

$$\begin{array}{ll}
 & h \cdot \overset{\gamma}{\leftarrow} \subseteq \overset{\alpha}{\leftarrow} \cdot h \\
\equiv & \quad \{ \overset{\beta}{\leftarrow} = \epsilon_{\mathsf{F}} \cdot \beta \} \\
 & h \cdot \epsilon_{\mathsf{F}} \cdot \gamma \subseteq \epsilon_{\mathsf{F}} \cdot \alpha \cdot h \\
\equiv & \quad \{ \epsilon_{\mathsf{F}} \text{ is a natural transformation: } f \cdot \epsilon_F = \epsilon_F \cdot \mathsf{F}f \ \} \\
 & \epsilon_{\mathsf{F}} \cdot \mathsf{F}h \cdot \gamma \subseteq \epsilon_{\mathsf{F}} \cdot \alpha \cdot h \\
\equiv & \quad \{ \text{Galois connection for division: } R \cdot S \subseteq T \Leftrightarrow S \subseteq R \backslash T \} \\
 & \mathsf{F}h \cdot \gamma \subseteq \epsilon_{\mathsf{F}} \backslash (\epsilon_{\mathsf{F}} \cdot \alpha \cdot h)
\end{array}$$

$\equiv \quad \{\ \text{property } R\backslash(S \cdot f) = (R\backslash S) \cdot f\ \}$

$\mathsf{F}h \cdot \gamma \subseteq (\epsilon_{\mathsf{F}}\backslash\epsilon_{\mathsf{F}}) \cdot \alpha \cdot h$

$\equiv \quad \{ \leq = \epsilon_{\mathsf{F}}\backslash\epsilon_{\mathsf{F}}\}$

$\mathsf{F}h \cdot \gamma \subseteq \leq \cdot \alpha \cdot h$

$\equiv \quad \{\ \text{lifting of } \leq\}$

$\mathsf{F}h \cdot \gamma \mathrel{\dot{\leq}} \alpha \cdot h$

□

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,*

$$\overset{\gamma}{\leftarrow} \cdot h \subseteq h \cdot \overset{\alpha}{\leftarrow} \tag{21}$$

Then (21) is equivalent to forward morphism condition

$$\gamma \cdot h \mathrel{\dot{\leq}} \mathsf{F}h \cdot \alpha$$

where $\leq$ is again the structural inclusion relation.

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

$$\gamma \cdot h \mathrel{\dot{\leq}} \mathsf{F}h \cdot \alpha \Rightarrow \overset{\gamma}{\leftarrow} \cdot h \subseteq h \cdot \overset{\alpha}{\leftarrow}$$

in [26], for all $\leq$ belonging to interval $id \subseteq \leq \subseteq (\in_{\mathsf{F}}\backslash\in_{\mathsf{F}})$. A proof of the equivalence for the case where the refinement preoder is generic inclusion, follows.

$\overset{\gamma}{\leftarrow} \cdot h \subseteq h \cdot \overset{\alpha}{\leftarrow}$

$\equiv \quad \{ \overset{\beta}{\leftarrow} = \epsilon_{\mathsf{F}} \cdot \beta\}$

$\epsilon_{\mathsf{F}} \cdot \gamma \cdot h \subseteq h \cdot \epsilon_{\mathsf{F}} \cdot \alpha$

$\equiv \quad \{ \epsilon_{\mathsf{F}} \text{ is a natural transformation: } f \cdot \epsilon_F = \epsilon_F \cdot \mathsf{F}f\ \}$

$\epsilon_{\mathsf{F}} \cdot \gamma \cdot h \subseteq \epsilon_{\mathsf{F}} \cdot \mathsf{F}h \cdot \alpha$

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

$\gamma \cdot h \subseteq \epsilon_{\mathsf{F}}\backslash(\epsilon_{\mathsf{F}} \cdot \mathsf{F}h \cdot \alpha)$

$\equiv \quad \{\ \text{property } R\backslash(S \cdot f) = (R\backslash S) \cdot f\ \}$

$\gamma \cdot h \subseteq (\epsilon_{\mathsf{F}}\backslash\epsilon_{\mathsf{F}}) \cdot \mathsf{F}h \cdot \alpha$

$\equiv \quad \{ \leq = \epsilon_{\mathsf{F}}\backslash\epsilon_{\mathsf{F}}\}$

$\gamma \cdot h \subseteq \leq \cdot \mathsf{F}h \cdot \alpha$

$\equiv \quad \{\ \text{lifting of } \leq\}$

$\gamma \cdot h \mathrel{\dot{\leq}} \mathsf{F}h \cdot \alpha$

□

These two lemmas show a close relationship among coalgebras and their relational counterparts. In fact, for any such coalgebra α, its transition relation is easy to spell out:

$$\xleftarrow{\alpha} \overset{\text{def}}{=} \in_{\mathsf{F}} \cdot\, \alpha \tag{22}$$

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

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

$$h\cdot \xrightarrow{p} = \xrightarrow{q} \cdot h \tag{23}$$

cf. the following diagram:

$$\begin{tikzcd}
X \arrow[r, "h"] \arrow[d, "\xrightarrow{p}"'] & Y \arrow[d, "\xrightarrow{q}"] \\
X \arrow[r, "h"'] & Y
\end{tikzcd}$$

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,

$$f \cdot R \subseteq S \equiv R \subseteq f^{\circ} \cdot S \tag{24}$$

$$R \cdot f^{\circ} \subseteq S \equiv R \subseteq S \cdot f \tag{25}$$

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

$$h\cdot \xleftarrow{\gamma} \subseteq \xleftarrow{\alpha} \cdot h \tag{26}$$

can be reduced to

$$\xleftarrow{\gamma} \cdot h^{\circ} \subseteq h^{\circ} \cdot \xleftarrow{\alpha} \tag{27}$$

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

Proof.

h is an upward simulation

$\equiv$ { by definition of upward simulation}

$h \cdot \xleftarrow{\gamma} \subseteq \xleftarrow{\alpha} \cdot h$

$\equiv$ { shunting rules}

$\xleftarrow{\gamma} \cdot h^\circ \subseteq h^\circ \cdot \xleftarrow{\alpha}$

$\equiv$ { by definition of downward simulation}

h° 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 α 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,

$$\alpha \preceq_R \beta \wedge \beta \preceq_S \gamma \Rightarrow \alpha \preceq_{R \cdot S} \gamma$$

$\alpha \preceq_{R \cdot S} \gamma$

$\equiv$ { by definition of $\preceq$}

$\xleftarrow{\alpha} \cdot R \cdot S \subseteq R \cdot S \cdot \xleftarrow{\gamma}$

$\Leftarrow$ { $\beta \preceq_S \gamma$ and transitivity of inclusion}

$\xleftarrow{\alpha} \cdot R \cdot S \subseteq R \cdot \xleftarrow{\beta} \cdot S$

$\Leftarrow$ { monotonicity of $(\cdot S)$, because it is an adjoit in a Galois connection [34]}

$\xleftarrow{\alpha} \cdot R \subseteq R \cdot \xleftarrow{\beta}$

$\equiv$ { $\alpha \preceq_R \beta$}

True

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

- functions

$$\alpha \preceq_h \beta \wedge \beta \preceq_g \gamma \Rightarrow \alpha \preceq_{h \cdot g} \gamma$$

- converses of functions

$$\alpha \preceq_{h^\circ} \beta \wedge \beta \preceq_{g^\circ} \gamma \Rightarrow \alpha \preceq_{(g \cdot h)^\circ} \gamma$$

Relation to bisimilarity is given by the following lemma

Lemma 4.2 (Relation with bisimilarity) *Preorder $\preceq$ is related to bisimilarity by*

$$\alpha \preceq_{f^\circ} \alpha \wedge \alpha \preceq_f \alpha \;\equiv\; \alpha \sim_f \alpha \tag{28}$$

Proof.

$\alpha \preceq_{f^\circ} \alpha \wedge \alpha \preceq_f \alpha$

$\equiv$ { by definition of $\preceq$}

$\xrightarrow{\alpha} \cdot f^\circ \subseteq f^\circ \cdot \xrightarrow{\alpha} \wedge \xrightarrow{\alpha} \cdot f \subseteq f \cdot \xrightarrow{\alpha}$

$\equiv$ { shunting, cf. (24) and (25)}

$f \cdot \xrightarrow{\alpha} \subseteq \xrightarrow{\alpha} \cdot f \wedge \xrightarrow{\alpha} \cdot f \subseteq f \cdot \xrightarrow{\alpha}$

$\equiv$ { ping-pong: $R \subseteq S \wedge S \subseteq R \equiv R = S$}

$f \cdot \xrightarrow{\alpha} = \xrightarrow{\alpha} \cdot f$

$\equiv$ { homomorphism on F-coalgebra α, (23)}

$\mathsf{F}\ f \cdot \alpha = \alpha \cdot f$

$\equiv$ { (endo)homomorphism iff (functional) bisimulation}

$\alpha \sim_f \alpha$

$\square$

Since a homomorphism is a functional bisimulation [37], we arrive at the following corollary:

Corollary 4.3

$$\alpha \preceq_{f^\circ} \gamma \wedge \gamma \preceq_f \alpha \;\equiv\; \alpha \sim_f \gamma \tag{29}$$

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,

$\xleftarrow{\gamma} \sqsubseteq_R \xleftarrow{\alpha}$

where $R = h$ or $R = h^\circ$ for some h, meaning (21) or (27), respectively.

Lemma 4.4 (Monotonicity of converse) *Converse is monotonic:*

$$\xleftarrow{\gamma} \sqsubseteq_{h^\circ} \xleftarrow{\alpha} \equiv \xleftarrow{\gamma}^\circ \sqsubseteq_{h^\circ} \xleftarrow{\alpha}^\circ \tag{30}$$

Proof.

$\xleftarrow{\gamma} \sqsubseteq_{h^\circ} \xleftarrow{\alpha}$

$\equiv$ { by definition of $\sqsubseteq$}

$\xleftarrow{\gamma} \cdot h^\circ \subseteq h^\circ \cdot \xleftarrow{\alpha}$

$\equiv$ { converse:$R \subseteq S \equiv R^\circ \subseteq S^\circ$}

$h \cdot \xleftarrow{\gamma}^\circ \subseteq \xleftarrow{\alpha}^\circ \cdot h$

$\equiv$ { shunting, cf. (24) and (25)}

$\xleftarrow{\gamma}^\circ \cdot h^\circ \subseteq h^\circ \cdot \xleftarrow{\alpha}^\circ$

$\equiv$ { by definition of $\sqsubseteq$}

$\xleftarrow{\gamma}^\circ \sqsubseteq_{h^\circ} \xleftarrow{\alpha}^\circ$

□

Note that monotonicity of converse cannot be proved when we replace h° by h.

Lemma 4.5 (Monotonicity of $\cup$) *If* $\xleftarrow{\gamma} \sqsubseteq_R \xleftarrow{\alpha}$ *and* $\xleftarrow{\beta} \sqsubseteq_R \xleftarrow{\delta}$ *then*

$$(\xleftarrow{\gamma} \cup \xleftarrow{\beta}) \sqsubseteq_R (\xleftarrow{\alpha} \cup \xleftarrow{\delta}) \tag{31}$$

Proof.

$\xleftarrow{\gamma} \sqsubseteq_R \xleftarrow{\alpha} \wedge \xleftarrow{\beta} \sqsubseteq_R \xleftarrow{\delta}$

$\equiv$ { definition of $\sqsubseteq$}

$\xleftarrow{\gamma} \cdot R \subseteq R \cdot \xleftarrow{\alpha} \wedge \xleftarrow{\beta} \cdot R \subseteq R \cdot \xleftarrow{\delta}$

$\Rightarrow$ { monotonicity of $\cup$}

$\xleftarrow{\gamma} \cdot R \cup \xleftarrow{\beta} \cdot R \subseteq R \cdot \xleftarrow{\alpha} \cup R \cdot \xleftarrow{\delta}$

$\equiv$ { $(R\cdot)$ and $(\cdot R)$ are lower Galois adjoints [34], thus distribute over join}

$(\xleftarrow{\gamma} \cup \xleftarrow{\beta}) \cdot R \subseteq R \cdot (\xleftarrow{\alpha} \cup \xleftarrow{\delta})$

$\equiv$ { definition of $\sqsubseteq$}

$\xleftarrow{\gamma} \cup \xleftarrow{\beta} \sqsubseteq_R \xleftarrow{\alpha} \cup \xleftarrow{\delta}$

□

Lemma 4.6 (Monotonicity of $\cap$) *If* $\xleftarrow{\alpha} \cdot f \subseteq f^\circ \cdot \xleftarrow{\beta}$ *and* $\xleftarrow{\gamma} \cdot f \subseteq f^\circ \cdot \xleftarrow{\delta}$ *then*

$$(\xleftarrow{\alpha} \cap \xleftarrow{\gamma}) \cdot f \subseteq f^\circ \cdot (\xleftarrow{\beta} \cap \xleftarrow{\delta}) \tag{32}$$

Proof.

$\xleftarrow{\alpha} \cdot f \subseteq f^\circ \cdot \xleftarrow{\beta} \wedge \xleftarrow{\gamma} \cdot f \subseteq f^\circ \cdot \xleftarrow{\delta}$

$\equiv$ { monotonicity of $\cap$}

$\xleftarrow{\alpha} \cdot f \cap \xleftarrow{\gamma} \cdot f \subseteq f^\circ \cdot \xleftarrow{\beta} \cap f^\circ \cdot \xleftarrow{\delta}$

$\equiv$ {$(f^\circ \cdot)$ and $(\cdot f)$ are upper Galois adjoints [34], thus they distribute over meet}

$(\xleftarrow{\alpha} \cap \xleftarrow{\gamma}) \cdot f \subseteq f^\circ \cdot (\xleftarrow{\beta} \cap \xleftarrow{\delta})$

□

If $f = f^\circ$ we have the following monotonicity result: if $\xleftarrow{\alpha} \sqsubseteq_f \xleftarrow{\beta}$ and $\xleftarrow{\gamma} \sqsubseteq_f \xleftarrow{\delta}$ then $(\xleftarrow{\alpha} \cap \xleftarrow{\gamma}) \sqsubseteq_f (\xleftarrow{\beta} \cap \xleftarrow{\delta})$

Lemma 4.7 (Monotonicity of composition) *If* $\xleftarrow{\gamma} \sqsubseteq_R \xleftarrow{\alpha}$ *and* $\xleftarrow{\beta} \sqsubseteq_R \xleftarrow{\delta}$ *then*

$$(\xleftarrow{\gamma} \cdot \xleftarrow{\beta}) \sqsubseteq_R (\xleftarrow{\alpha} \cdot \xleftarrow{\delta}) \tag{33}$$

Proof.

$\xleftarrow{\gamma} \cdot \xleftarrow{\beta} \cdot R$

$\subseteq$ { by hypothesis}

$\xleftarrow{\gamma} \cdot R \cdot \xleftarrow{\delta}$

$\subseteq$ { by hypothesis}

$R \cdot \xleftarrow{\alpha} \cdot \xleftarrow{\delta}$

□

Finally, for an arbitrary relator F,

Lemma 4.8 (F-monotonicity) *If* $\xleftarrow{\gamma} \sqsubseteq_R \xleftarrow{\alpha}$ *then*

$$\mathsf{F} \xleftarrow{\gamma} \sqsubseteq_{\mathsf{F}R} \mathsf{F} \xleftarrow{\alpha} \tag{34}$$

Proof.

$\xleftarrow{\gamma} \sqsubseteq_R \xleftarrow{\alpha}$

$\equiv$ { by definition of $\sqsubseteq$}

$\xleftarrow{\gamma} \cdot R \subseteq R \cdot \xleftarrow{\alpha}$

$\Rightarrow$ { F-monotonicity}

$\mathsf{F}(\xleftarrow{\gamma} \cdot R) \subseteq \mathsf{F}(R \cdot \xleftarrow{\alpha})$

$\Rightarrow$ { F distributes over composition}

$(\mathsf{F} \xleftarrow{\gamma}) \cdot \mathsf{F}R \subseteq \mathsf{F}R \cdot \mathsf{F} \xleftarrow{\alpha}$

$$\equiv \qquad \{ \text{ by definition of } \sqsubseteq \}$$

$$\mathsf{F} \xleftarrow{\gamma} \sqsubseteq_{\mathsf{F}R} \mathsf{F} \xleftarrow{\alpha}$$

□

Corollary 4.9 *If* $\xleftarrow{\gamma} \sqsubseteq_R \xleftarrow{\alpha}$ *and* $\xleftarrow{\beta} \sqsubseteq_S \xleftarrow{\delta}$ *then*

$$(\xleftarrow{\gamma} \times \xleftarrow{\beta}) \sqsubseteq_{R\times S} (\xleftarrow{\alpha} \times \xleftarrow{\delta}) \tag{35}$$

$$(\xleftarrow{\gamma} + \xleftarrow{\beta}) \sqsubseteq_{R+S} (\xleftarrow{\alpha} + \xleftarrow{\delta}) \tag{36}$$

This calculus avoids the need to resort to complicated downward simulations. Instead, we just need to know that *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 $\xrightarrow{\alpha}$.

$$\begin{tikzcd} s_0 \arrow[r, "a"] \arrow[d, "a"] & s_1 \arrow[r, "a"] \arrow[d, "a"] & \cdots \\ s'_0 & s'_1 & \end{tikzcd}$$

Suppose one wants to calculate an implementation $\xrightarrow{\gamma}$ such that:

$$\xrightarrow{\gamma} \subseteq \xrightarrow{\alpha} \tag{37}$$

corresponding to downward simulation id:

$$\xrightarrow{\gamma} \cdot id \subseteq id \cdot \xrightarrow{\alpha} \tag{38}$$

that is, to the following inequation,

$$\gamma \preceq_{id} \alpha \tag{39}$$

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,

$$\xrightarrow{\gamma} = \xrightarrow{\alpha} - \{(s_i, s'_i) | i \geq 0\}$$

To transition relation $\xrightarrow{\gamma}$ one associates the obvious LTS R, a completely deterministic one, depicted as follows:

$$\begin{tikzcd}
s_0 \arrow[r, "a"] & s_1 \arrow[r, "a"] & \cdots \\
s'_0 & s'_1 &
\end{tikzcd}$$

The implementation coalgebra γ, the power transpose of LTS R, is such that

$$\gamma \cdot id \mathrel{\dot{\leq}} \mathsf{F}\ id \cdot \alpha \tag{40}$$

according to (38), Lemma 3.4 and $\mathsf{F} = \mathcal{P}A \times Id$. Inequation (40) simplifies to:

$$\gamma \mathrel{\dot{\leq}} \alpha \tag{41}$$

since $\mathsf{F}\ id = id$ 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°. We defined a preorder $\preceq$ over coalgebras, which witnesses the downward simulations, and proved that the corresponding preorder $\sqsubseteq$ 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 $id \subseteq \leq \subseteq (\in_\mathsf{T} \backslash \in_\mathsf{T})$ 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

[1] C. Aarts, R.C. Backhouse, P. Hoogendijk, E.Voermans, and J. van der Woude. A relational theory of datatypes, December 1992.

[2] J. Adamek. Introduction to coalgebra. *Theory and Applications of Categories*, 14:157–199, 2005.

[3] K. Backhouse and R.C. Backhouse. Safety of abstract interpretations for free,via logical relations and Galois connections. *Science of Computer Programming*, 15(1–2):153–196, 2004.

[4] R.C. Backhouse. Mathematics of program construction, June 2004. With help and contributions by Marcel Bijsterveld and Henk Doornbos and Rik van Geldrop and Diethard Michaelis and Jaap van der Woude.

[5] R.C. Backhouse, P. de Bruin, P. Hoogendijk, G. Malcolm, T.S. Voermans, and J. van der Woude. Polynomial relators. In *2nd Int. Conf. Algebraic Methodology and Software Technology (AMAST'91)*, pages 303–362. Springer LNCS, 1992.

[6] L. S. Barbosa. Components as processes: An exercise in coalgebraic modeling. In S. F. Smith and C. L. Talcott, editors, *FMOODS'2000 - Formal Methods for Open Object-Oriented Distributed Systems*, pages 397–417. Kluwer Academic Publishers, September 2000.

[7] L. S. Barbosa and J. N. Oliveira. Transposing partial components: an exercise on coalgebraic refinement. *Theor. Comp. Sci.*, 365(1-2):2–22, 2006.

[8] R. Bird and O. de Moor. *Algebra of Programming.* Series in Computer Science. Prentice-Hall International, 1997. C.A.R. Hoare, series editor.

[9] E. Boiten and W. de Roever. Getting to the bottom of relational refinement: Relations and correctness, partial and total. In R. Berghammer and B. Möller, editors, *7th International Seminar on Relational Methods in Computer Science (RelMiCS 7)*, pages 82–88. University of Kiel, May 2003.

[10] E. Boiten and J. Derrick. Unifying concurrent and relational refinement. *ENTCS*, 70(3), 2002. REFINE02-The BCS FACS Refinement Workshop, Denmark July 2002.

[11] C. Bolton, J. Davies, and J. Woodcock. On the refinement and simulation of data types and processes. In *IFM*, pages 273–292. Springer, 1999.

[12] A. Cruz, L. Barbosa, and J. Oliveira. From algebras to objects: Generation and composition. *Journal of Universal Computer Science*, 11(10):1580–1612, 2005.

[13] W. de Roever and K. Engelhardt. *Data Refinement Model-Oriented Proof methods and their Comparison.* Cambridge University Press, 1999. With the assistance of J. Coenen and K.-H. Buth and P. Gardiner and Y. Lakhnech and F. Stomp.

[14] J. Derrick and E. Boiten. Relational concurrent refinement. *Formal Aspects of Computing*, 15(2-3):182–214, 2003.

[15] P. Gardiner and C. Morgan. A single complete rule for data refinement. *Formal Aspects of Computing*, 5(4):367–382, 1993.

[16] L. Groves. Refinement and the Z schema calculus. *ENTCS*, 70(3), 2002. REFINE02-The BCS FACS Refinement Workshop, Denmark July 2002.

[17] H. Peter Gumm. Elements of the general theory of coalgebras. Technical report, Lecture Notes for LUTACS'99, South Africa, 1999.

[18] J. He. Process refinement. In J. McDermid, editor, *The Theory and Practice of Refinement*, pages 37–59. Butterworths, 1989.

[19] U. Hensel and H. Reichel. Defining equations in terminal coalgebras. In E. Astesiano, G. Reggio, and A. Tarlecki, editors, *Recent Trends in Data Type Specification*, pages 307–318. Springer Lect. Notes Comp. Sci. (906), 1995.

[20] B. Jacobs. Object-oriented hybrid systems of coalgebras plus monoid actions. In M. Wirsing and M. Nivat, editors, *Algebraic Methodology and Software Technology (AMAST)*, pages 520–535. Springer Lect. Notes Comp. Sci. (1101), 1996.

[21] B. Jacobs. Objects and classes, co-algebraically. In C. Lengauer B. Freitag, C.B. Jones and H.-J. Schek, editors, *Object-Orientation with Parallelism and Persistence*, pages 83–103. Kluwer Acad. Publ., 1996.

[22] B. Jacobs. Behaviour-refinement of coalgebraic specifications with coinductive correctness proofs. In *TAPSOFT'97: Theory and Practice of Software Development*, pages 787–802. Springer Lect. Notes Comp. Sci. (1214), 1997.

[23] B. Jacobs and J. Rutten. A tutorial on (co)algebras and (co)induction. *EATCS Bulletin*, 62:222–159, 1997.

[24] Bart Jacobs. Exercises in coalgebraic specification. In R. Backhouse, R. Crole, and J. Gibbons, editors, *Algebraic and Coalgebraic Methods in the Mathematics of Program Construction*, pages 237–280. Springer Lect. Notes Comp. Sci. (2297), 2002.

[25] S. Meng and L.S. Barbosa. On refinement of generic state-based software components. In C. Rettray, S. Maharaj, and C. Shankland, editors, *10th Int. Conf. Algebraic Methods and Software Technology (AMAST)*, volume 3116 of *Lecture Notes in Computer Science*, pages 506–520. Springer, Stirling, July 2004.

[26] Sun Meng and L. S. Barbosa. Components as coalgebras: The refinement dimension. *Theor. Comp. Sci.*, 351:276–294, 2005.

[27] R. Milner. A Calculus of Communicating Systems. *Lecture Notes in Computer Science*, 92, 1980.

[28] L. Moss. Coalgebraic logic. *Ann. Pure & Appl. Logic*, 96(1-3):277–317, 1999.

[29] J.N. Oliveira. Software reification using the SETS calculus. In T. Denvir, C.B. Jones, and R.C. Shaw, editors, *Proc. of the BCS FACS 5th Refinement Workshop, Theory and Practice of Formal Software Development, London, UK*, pages 140–171. Springer-Verlag, 8–10 January 1992.

[30] J.N. Oliveira and C.J. Rodrigues. Transposing relations: from *Maybe* functions to hash tables. In D. Kozen, editor, *MPC'07: Seventh International Conference on Mathematics of Program Construction, 12-14 July, 2004, Stirling, Scotland, UK*, volume 3125 of *LNCS*, pages 334–356. Springer, July 2004.

[31] J.N. Oliveira and C.J. Rodrigues. Pointfree factorization of operation refinement. In *FM'06: 14 International Symposium on Formal Methods, 21-27 August, 2006, McMaster University, Hamilton, Ontario, Canada*, volume 4085 of *Lecture Notes in Computer Science*, pages 236–251. Springer, August 2006.

[32] J. Power and H. Watanabe. An axiomatics for categories of coalgebras. In B. Jacobs, L. Moss, H. Reichel, and J. Rutten, editors, *CMCS'98, Elect. Notes in Theor. Comp. Sci.*, volume 11. Elsevier, March 1998.

[33] H. Reichel. An approach to object semantics based on terminal co-algebras. *Math. Struct. in Comp. Sci.*, 5:129–152, 1995.

[34] C.J. Rodrigues. *Foundations of Program Refinement by Calculation.* PhD thesis, University of Minho, July 2008.

[35] J. Roth, H. Tews, and B. Jacobs. The coalgebraic class specification language CCSL. *Journal of Universal Computer Science*, 7(2), 2001.

[36] J. Rutten. Automata and coinduction (an exercise in coalgebra). In *Proc. CONCUR' 98*, pages 194–218. Springer Lect. Notes Comp. Sci. (1466), 1998.

[37] J. Rutten. Universal coalgebra: A theory of systems. *Theor. Comp. Sci.*, 249(1):3–80, 2000. (Revised version of CWI Techn. Rep. CS-R9652, 1996).

[38] Daniele Turi and Jan Rutten. On the foundations of final coalgebra semantics: non-well-founded sets, partial orders, metric spaces. *Mathematical Structures in Computer Science*, 8(5):481–540, 1998.

[39] U. Wolter. A coalgebraic introduction to CSP. In *Proc. of CMCS'99*, volume 19. Elect. Notes in Theor. Comp. Sci., Elsevier, 1999.