Scientific Annals in Computer Science vol.17, 2007
"Alexandru Ioan Cuza" University of Iaşi, Romania

Homomorphism between AOMRC and Hoare Model of Deterministic Reconfiguration Processes in Reconfigurable Computing Systems

Phan Cong VINH [1]

Abstract

In this paper, the notion of aspect-oriented modular reconfigurable computing (AOMRC) is introduced, CSP-based behaviors of AOMRC are approached by developing a model of AOMRC and constructing a Hoare model of deterministic reconfiguration processes. Then, under the theory of coalgebras, we build a homomorphism between AOMRC and a Hoare model of deterministic reconfiguration processes. In other words, since AOMRC and the Hoare model of deterministic reconfiguration processes are seen as coalgebras, their homomorphic relationship results in the behavioral equivalence between AOMRC being carried out by a transformations-based aspect and a Hoare model of deterministic reconfiguration processes.

1 Introduction

Reconfigurable computing is computer processing with highly flexible computing components. Such an effective powerful paradigm joins together the flexible nature of software with hardware design and the high performance nature of hardware with software programming. This is an emerging approach in computing. The topic has seen a number of developments through various research investigations including computational models, languages,

[1]London South Bank University, Centre for Applied Formal Methods, Borough Road, London SE1 0AA, United Kingdom. Email: phanvc@ieee.org

and formalisms with the aim of providing high-level descriptions of reconfigurable computing systems [7–14,22–28]. However, devising a good programming technique, for ensuring that high flexibility of reconfigurable computing systems, is properly carried out is far from being obvious.

Apparently emerging as an orthogonal approach to modular programming methods, the programming technique based on *cross-cutting*, also referred to as *aspect* orientation [17], is seen to be a good potential practical paradigm for aspect-oriented modular reconfigurable computing as shown in [28], where we presented an approach to combining aspect-orientation and reconfigurable computing in a modular manner. Expanding on the approach in this paper, under the theory of coalgebras and *communicating sequential processes* (CSP) [15], we will examine CSP-based behaviors of aspect-oriented modular reconfigurable computing as a further development to our recent research [22–28] to embed the cross-cutting technique in modular reconfigurable computing systems.

In modular reconfigurable computing systems, an aspect is determined as transformations of configuration. Based on this aspect, a model of aspect-oriented modular reconfigurable computing (AOMRC) is developed and a Hoare model of deterministic reconfiguration processes is constructed. By this way, a homomorphism between AOMRC and a Hoare model of deterministic reconfiguration processes is investigated in detail under the theory of coalgebras to establish a behavioral equivalence between AOMRC and a Hoare model of deterministic reconfiguration processes. As a result, the homomorphism between AOMRC and a Hoare model of deterministic reconfiguration processes blurs the distinction between concepts of a configuration and process. A descriptive view of this behavioral equivalence is as follows:
Homomorphism: (AOMRC) $\longrightarrow$ (Hoare model of deterministic reconfiguration processes)

2 Outline

The paper is organized as follows: In section 3, we consider some related principal work and preliminary concepts. In section 4, we introduce the notion of aspect-oriented modular reconfigurable computing (AOMRC), including reconfiguration and its coalgebraic representations, transformations of configuration, transformation-based aspect in reconfigurable computing and inheritance property of aspect. A homomorphic relationship between

AOMRC and a Hoare model of deterministic reconfiguration processes is investigated in detail under the theory of coalgebras and CSP in section 5. A plan for future work is suggested in section 6. Finally, a short summary is given in section 7.

3 Related Major Work and Existing Basic Concepts

3.1 Related major work

Most notions and investigations of this paper are instances of a theory called *universal coalgebra* [16, 21], where some recent developments in coalgebra are presented such that

- *Bisimulation* is a categorical notion that applies to many different instances of infinite data structures, various other types of automata, and dynamic systems [16, 20, 21]. In theoretical computer science, a bisimulation is an equivalence relation between abstract machines, also called the abstract computers or state transition systems (i.e., a theoretical model of a computer hardware or software system) used in the study of computing. Abstraction of computing is usually considered as discrete time processes. Two computing systems are bisimular if, regarding their behaviors, each of the systems "simulates" the other and vice-versa. In other words, each of the systems cannot be distinguished from the other by the observation.

- *Homomorphism* is one of the fundamental concepts in categorical algebra [1, 2, 4, 18], which scrutinizes the sets of algebraic objects, relations of those algebraic objects, and functions from one set of algebraic objects to another. A function that preserves the relations of the algebraic objects is known as a homomorphism. In other words, if the source set of algebraic objects includes several relations, then all its relations must be preserved in the target set for the function to be a homomorphism between two sets of algebraic objects.

Moreover, in the considered context, we also apply the therory of CSP, which is a *formal language* for specifying interaction behaviors in reconfigurable computing systems. CSP was first described in a 1978 paper by C. A. R. Hoare [15], but has since grown substantially to become the modern process algebraic form. It is a member of the family of mathematical

theories of concurrency known as *process algebras*, or *process calculi*. CSP spreads its influence over the development of both software/hardware in traditional computing systems and flowware/configware in modern reconfigurable computing systems. The theory of CSP itself is still the subject of active research, including work to increase its range of practical applicability.

3.2 The notions of reconfigurable computing, configware and flowware

A definition of reconfigurable computing, sometimes known as configurable computing, is given by DeHon and Wawrzynek as:

> "*computing via a post-fabrication and spatially programmed connection of processing elements*" [6].

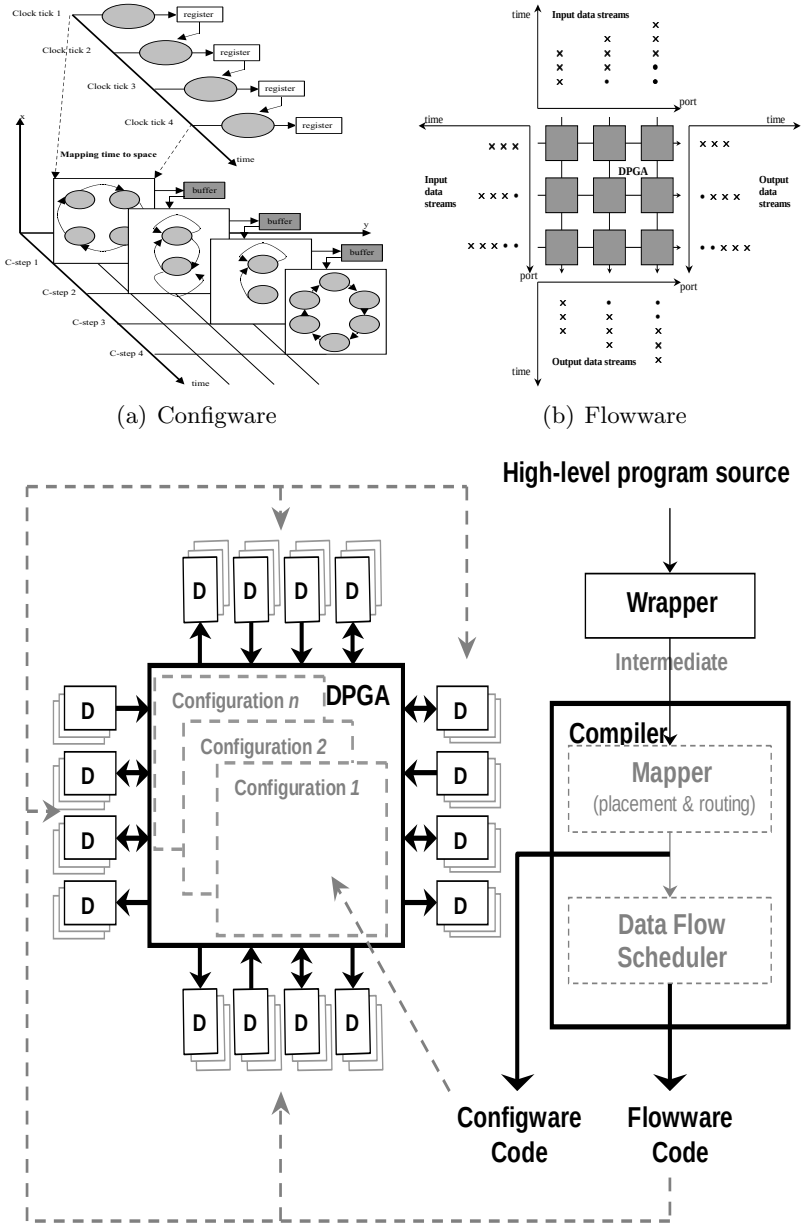
This definition is concerned with two major factors of reconfigurable computing:

- The post-fabrication property allows adjustment of the architecture for flexibility between applications due to varying data input flows and operation conditions.
- The spatial paradigm of computation, also called *configware*, contrasts with the temporal paradigm of computation using traditional microprocessors, called by the familiar name of *software*.

The implementation of reconfigurable computing has become realistic with the availability of *Field-Programmable Gate Array* (FPGA) technology [19]. A specific type of FPGA, which can be programmed on-the-fly during system operation, is *dynamically reconfigurable* FPGA, also known as a *Dynamically Programmable Gate Array* (DPGA) technology [5].

The above-mentioned definition of reconfigurable computing refers to the notion of R. Hartenstein's *configware engineering* [13], including *configware* and *flowware*, in which both configware and flowware codes are created by compiling inputs of a high-level programming language. Specifically, configware codes are produced for determining configuration before runtime, and flowware codes execute data input flows at runtime on an existing configuration [7–12, 14].

Configware means structural programming, or programming in space as presented in Figure 1(a). That is, configware executes computation in



(a) Configware (b) Flowware

(c) Configware/Flowware synthesis

Figure 1: Configware and Flowware (adapted from [13])

space. Figure 1(a) illustrates the configware approach, in which a sequence of operations is mapped from time to space and executed per *C-step* (also called a control step) in parallel on a DPGA-based platform; its intermediate results are stored in buffers for another sequence of operations in the next C-step.

Flowware means data-flow programming that schedules the data flow for output from or input to the configware using one or several data counters. Flowware determines which data item has to be connected with which port of which configuration at which time (see Figure 1(b)). The configware is vital for the flowware-based paradigm. This architecture allows the multiple data flows clocked into and out of such a pipelined network, in a similar manner to the heart and the blood streams.

Flowware may also be described by higher level flowware languages, which are similar to high level software languages like C [13]. These languages include control structures such as jumping and (nested) looping. The principal differences between software and flowware are that (sequential) software makes reference to only a single program counter, whereas flowware refers to several data counters. As a result, flowware also includes parallel looping as an important control structure, which is not supported by sequential software.

Both flowware and configware are two different complementary approaches that can be combined in programming for reconfigurable computing systems. Figure 1(c) describes a process of configware/flowware synthesis, generating both configware and flowware from a high level programming language source, in which configware is produced for establishing the configuration before runtime and flowware is used to execute data flows at runtime.

3.3 Some categorical and coalgebraic terms

In this section, we recall some concepts from the category theory [1, 2, 4, 18] and theory of coalgebras [16, 21] used in this paper.

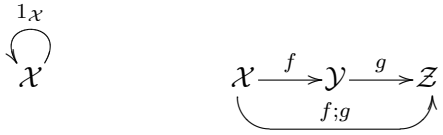
A category $\mathbf{C}$ can be viewed as a graph $(Obj(\mathbf{C}), Arc(\mathbf{C}), s, t)$, where

- $Obj(\mathbf{C})$ is the set of nodes we call *objects*,
- $Arc(\mathbf{C})$ is the set of edges we call *morphisms* and

- $s, t : Arc(\mathbf{C}) \longrightarrow Obj(\mathbf{C})$ are two maps called *source* and *target*, respectively.

We write $f : \mathcal{X} \longrightarrow \mathcal{Y}$ when f is in $Arc(\mathbf{C})$ and $s(f) = \mathcal{X}$ and $t(f) = \mathcal{Y}$.

Associated with each object $\mathcal{X}$ in $Obj(\mathbf{C})$, there is a morphism $1_{\mathcal{X}} = \mathcal{X} \longrightarrow \mathcal{X}$, called the *identity* morphism on $\mathcal{X}$, and to each pair of morphisms $f : \mathcal{X} \longrightarrow \mathcal{Y}$ and $g : \mathcal{Y} \longrightarrow \mathcal{Z}$, there is an associated morphism $f; g : \mathcal{X} \longrightarrow \mathcal{Z}$, called the *composition* of f with g.

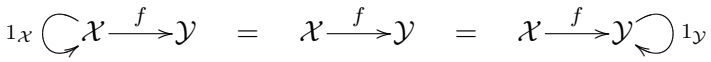


(1)

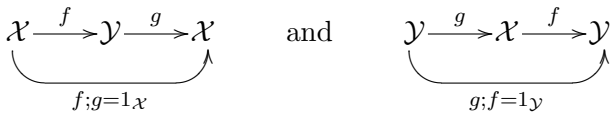
The following equations must hold for all objects $\mathcal{X}$, $\mathcal{Y}$ and $\mathcal{Z}$ and morphisms $f : \mathcal{X} \longrightarrow \mathcal{Y}$, $g : \mathcal{Y} \longrightarrow \mathcal{Z}$ and $h : \mathcal{Z} \longrightarrow \mathcal{T}$:

$$\textit{Associativity}: \qquad (f; g); h = f; (g; h) \tag{2}$$

$$\textit{Identity}: \qquad 1_{\mathcal{X}}; f = f = f; 1_{\mathcal{Y}} \tag{3}$$

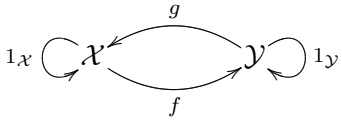


A morphism $f : \mathcal{X} \longrightarrow \mathcal{Y}$ in the category $\mathbf{C}$ is an *isomorphism* if there exists a morphism $g : \mathcal{Y} \longrightarrow \mathcal{X}$ in that category such that $f; g = 1_{\mathcal{X}}$ and $g; f = 1_{\mathcal{Y}}$.



(4)

That is, if the following diagram commutes.



(5)

For any set A, $x \in A$ iff $1 \xrightarrow{x} A$ (or $x : 1 \longrightarrow A$) where 1 denotes a singleton set. Hence, we can write $1 \xrightarrow{1} \mathbb{N}$ (or $1 : 1 \longrightarrow \mathbb{N}$) for $1 \in \mathbb{N}$, $1 \xrightarrow{i} \mathbb{N}$ (or $i : 1 \longrightarrow \mathbb{N}$) for $i \in \mathbb{N}$ and so on

Functor is a special type of mapping between categories. Functor from a category to itself is called an *endofunctor*. Note that, in this paper, when the notion of endofunctor dominates throughout in use, then we can name them as the functor, for short, without any confusion. The functors are also viewed as morphisms in a category, whose objects are smaller categories. There are two kinds of functors distinguished by the way they treat morphisms to be *covariant* and *contravariant*. A functor T is covariant if for each source morphism $\mathcal{X} \xrightarrow{f} \mathcal{Y}$ the target morphism has the form $\mathsf{T}\ \mathcal{X} \xrightarrow{\mathsf{T}\ f} \mathsf{T}\ \mathcal{Y}$. A functor T is contravariant if for each source morphism $\mathcal{X} \xrightarrow{f} \mathcal{Y}$ the target morphism has the form $\mathsf{T}\ \mathcal{X} \xleftarrow{\mathsf{T}\ f} \mathsf{T}\ \mathcal{Y}$.

Let $\mathbf{C}$ be a category and $\mathsf{T} : \mathbf{C} \longrightarrow \mathbf{C}$ an endofunctor. A T-*coalgebra* is a pair $\langle \mathcal{C}, f \rangle$, where $\mathcal{C}$ is an object in $\mathbf{C}$ and $f : \mathcal{C} \longrightarrow \mathsf{T}\ \mathcal{C}$ a morphism in $\mathbf{C}$. $\mathcal{C}$ is called a *carrier* of the coalgebra and T an *interface* of the coalgebra.

Let $\mathcal{C}$ and $\mathcal{C}'$ be two objects in the category $\mathbf{C}$. T-*homomorphism* between two T-coalgebras $\langle \mathcal{C}, f \rangle$ and $\langle \mathcal{C}', g \rangle$ is a mapping $\mathcal{C} \xrightarrow{h} \mathcal{C}'$ such that the following equation holds

$$f ; \mathsf{T}\ h = h ; g \tag{6}$$

It is equivalent to saying that the following diagram commutes

$$\begin{array}{ccc} \mathcal{C} & \xrightarrow{\quad f \quad} & \mathsf{T}\ \mathcal{C} \\ {\scriptstyle h}\downarrow & & \downarrow{\scriptstyle \mathsf{T}\ h} \\ \mathcal{C}' & \xrightarrow{\quad g \quad} & \mathsf{T}\ \mathcal{C}' \end{array} \tag{7}$$

Let $\mathbf{C}$ be a category and $\mathsf{T} : \mathbf{C} \longrightarrow \mathbf{C}$ an endofunctor. A *final* T-coalgebra is simply a final object in the associated category $\mathbf{CoAlg}(\mathsf{T})$ of T-coalgebras. Thus, it is a coalgebra $\langle \mathcal{Z}, fi : \mathcal{Z} \longrightarrow \mathsf{T}\ \mathcal{Z} \rangle$ such that for any coalgebra $\langle \mathcal{X}, f : \mathcal{X} \longrightarrow \mathsf{T}\ \mathcal{X} \rangle$ there is a *unique homomorphism* $h : \mathcal{X} \longrightarrow \mathcal{Z}$ of

coalgebras, as in:

$$\begin{array}{ccc} \mathcal{X} & \xrightarrow{\quad f \quad} & \mathsf{T}\,\mathcal{X} \\ {\scriptstyle h}\,\vdots & & \vdots\,{\scriptstyle \mathsf{T}\,h} \\ \mathcal{Z} & \xrightarrow{\quad fi \quad} & \mathsf{T}\,\mathcal{Z} \end{array} \tag{8}$$

The dashed notation in the diagram (8) is used for uniqueness.

Let $\mathbf{C}$ and $\mathbf{C}'$ be two categories. Consider a parallel pair

$$\mathbf{C} \underset{\mathsf{T}'}{\overset{\mathsf{T}}{\rightrightarrows}} \mathbf{C}' \tag{9}$$

of functors of the same variance. Two functors of T and T' are *naturally equivalent* (also called *naturally isomorphic*) if there is an inverse pair of natural isomorphisms between them. In other words, the inverse pair of natural isomorphisms between T and T' is a pair

$$\mathsf{T} \underset{\zeta_-}{\overset{\eta_-}{\rightleftarrows}} \mathsf{T}' \tag{10}$$

such that for each object $\mathcal{X}$ in $\mathbf{C}$ the morphisms

$$\mathsf{T}\,\mathcal{X} \underset{\zeta_{\mathcal{X}}}{\overset{\eta_{\mathcal{X}}}{\rightleftarrows}} \mathsf{T}'\,\mathcal{X} \tag{11}$$

are an inverse pair of isomorphisms in $\mathbf{C}'$.

4 Aspect-Oriented Modular Reconfiguration Computing (AOMRC)

4.1 Reconfiguration and its coalgebraic representations

AOMRC we want to model is intuitionally multiple partial function applications, such as

$$c \xrightarrow{t[1]} (o[1], c[1]) \xrightarrow{t[2]} (o[2], c[2]) \xrightarrow{t[3]} (o[3], c[3]) \ldots \tag{12}$$

where

- All indexes $i \in \mathbb{N}$ in square brackets refer to *control steps*,
- c is a configuration of AOMRC in the set, denoted by $\mathcal{C}$, of configurations. $c[i]$ is the configuration c at the control step i,
- o is a value in the set, denoted by $\mathcal{O}$, of observable output values. $o[i]$ is the output o at the control step i, and
- t is a transformation in the set, denoted by T, of transformations. $t[i]$ is the transformation t at the control step i.

The notation $c \xrightarrow{t[1]} (o[1], c[1])$ is viewed as, at the first control step, the initial configuration c is changed to the configuration $c[1]$ and the output $o[1]$ is produced by the transformation $t[1]$. In general, $c[i] \xrightarrow{t[i+1]} (o[i+1], c[i+1])$ states that the transformation $t[i+1]$ changes the configuration $c[i]$ to $c[i+1]$ and issues the output $o[i+1]$ at the control step $i+1$.

The diagram (12) means that *reconfiguration* with the set T of transformations is a pair $\langle \mathcal{C}, \langle o_{\mathcal{C}}, e_{\mathcal{C}} \rangle \rangle$ consisting of the set $\mathcal{C}$ of configurations, an *output function* $o_{\mathcal{C}} : \mathcal{C} \longrightarrow \mathcal{O}^T$ and an *evolution function* $e_{\mathcal{C}} : \mathcal{C} \longrightarrow \mathcal{C}^T$, where $\mathcal{O}^T$ and $\mathcal{C}^T$ are read as the sets of partial functions $\{f : T \rightharpoonup \mathcal{O}\}$ and $\{g : T \rightharpoonup \mathcal{C}\}$, respectively. $o_{\mathcal{C}}$ assigns, to a configuration c, a function $o_{\mathcal{C}}(c) : T \rightharpoonup \mathcal{O}$, which specifies the value $o_{\mathcal{C}}(c)(t)$ that is reached after a transformation t has been executed. Similarly, $e_{\mathcal{C}}$ assigns, to a configuration c, a function $e_{\mathcal{C}}(c) : T \rightharpoonup \mathcal{C}$, which specifies the configuration $e_{\mathcal{C}}(c)(t)$ that is reached after a transformation t has been executed. Sometimes $c \xrightarrow{t} c'$ is used to denote $e_{\mathcal{C}}(c)(t) = c'$. Generally, both the set $\mathcal{C}$ of configurations and the set T of transformations may be infinite. If both $\mathcal{C}$ and T are finite, then we have a finite reconfiguration, otherwise we have an infinite reconfiguration.

We define a bisimulation between two reconfigurations $\langle \mathcal{C}, \langle o_{\mathcal{C}}, e_{\mathcal{C}} \rangle \rangle$ and $\langle \mathcal{C}', \langle o_{\mathcal{C}'}, e_{\mathcal{C}'} \rangle \rangle$ to be a relation $R \subseteq \mathcal{C} \times \mathcal{C}'$ with, for all c in $\mathcal{C}$, c' in $\mathcal{C}'$ and t in T

$$\text{if } \; c \; R \; c' \; \text{ then } \; \begin{cases} o_{\mathcal{C}}(c)(t) = o_{\mathcal{C}'}(c')(t) & \text{and} \\ e_{\mathcal{C}}(c)(t) \; R \; e_{\mathcal{C}'}(c')(t). \end{cases} \tag{13}$$

In other words,

$$\text{if } \; \langle c, c' \rangle \in R \; \text{ then } \; \begin{cases} o_{\mathcal{C}}(c)(t) = o_{\mathcal{C}'}(c')(t) & \text{and} \\ \langle e_{\mathcal{C}}(c)(t), e_{\mathcal{C}'}(c')(t) \rangle \in R. \end{cases} \tag{14}$$

This bisimulation relation is read as: two reconfigurations $\langle \mathcal{C}, \langle o_{\mathcal{C}}, e_{\mathcal{C}} \rangle \rangle$ and $\langle \mathcal{C}', \langle o_{\mathcal{C}'}, e_{\mathcal{C}'} \rangle \rangle$ issue the same observations in $\mathcal{O}$, and then the same transformation t is carried out on both configurations c in $\mathcal{C}$ and c' in $\mathcal{C}'$ that will lead to two new configurations $e_{\mathcal{C}}(c)(t)$ in $\mathcal{C}$ and $e_{\mathcal{C}'}(c')(t)$ in $\mathcal{C}'$ that are observationally indistinguishable again. Bisimulation between $\langle \mathcal{C}, \langle o_{\mathcal{C}}, e_{\mathcal{C}} \rangle \rangle$ and itself is called a bisimulation *on* $\langle \mathcal{C}, \langle o_{\mathcal{C}}, e_{\mathcal{C}} \rangle \rangle$. We write $c \sim c'$ whenever there exists a bisimulation R with $c \; R \; c'$.

For the reconfiguration $\langle \mathcal{C}, \langle o_{\mathcal{C}}, e_{\mathcal{C}} \rangle \rangle$, the output function $o_{\mathcal{C}} : \mathcal{C} \longrightarrow \mathcal{O}^T$ and evolution function $e_{\mathcal{C}} : \mathcal{C} \longrightarrow \mathcal{C}^T$ can be now combined into a *reconfiguration function* $\langle o_{\mathcal{C}}, e_{\mathcal{C}} \rangle : \mathcal{C} \longrightarrow (\mathcal{O} \times \mathcal{C})^T$, which maps c in $\mathcal{C}$ into the pair $\langle o_{\mathcal{C}}(c), e_{\mathcal{C}}(c) \rangle$. Thus the reconfiguration $\langle \mathcal{C}, \langle o_{\mathcal{C}}, e_{\mathcal{C}} \rangle \rangle$ is considered as the structure observed through the following *interface*.

$$\odot\frown\odot\,(\_) \;=\; (\mathcal{O} \times \_)^T \tag{15}$$

because applying this interface to the observable set $\mathcal{C}$ of configurations, we have

$$\odot\frown\odot\,(\mathcal{C}) \;=\; (\mathcal{O} \times \mathcal{C})^T \tag{16}$$

This interface of $\odot\frown\odot\,(\_)$ can be regarded as a mapping to decompose the observable set $\mathcal{C}$ of configurations into an observation context $(\mathcal{O} \times \_)^T$ of $\mathcal{C}$.

Let $\odot\frown\odot\,(\_) : \mathbf{C} \longrightarrow \mathbf{C}$ be a functor on the category $\mathbf{C}$ of sets. A $\odot\frown\odot\,(\_)$-*coalgebra* is defined by a pair $\langle \mathcal{C}, \langle o_{\mathcal{C}}, e_{\mathcal{C}} \rangle \rangle$ consisting of the set $\mathcal{C}$ of configurations and the reconfiguration function $\langle o_{\mathcal{C}}, e_{\mathcal{C}} \rangle : \mathcal{C} \longrightarrow \odot\frown\odot\,(\mathcal{C})$. In this way, the reconfiguration $\langle \mathcal{C}, \langle o_{\mathcal{C}}, e_{\mathcal{C}} \rangle \rangle$ is represented as a coalgebra of the functor $\odot\frown\odot\,(\_)$, which is defined on the category $\mathbf{C}$ by $\odot\frown\odot\,(\mathcal{C}) \;=\; (\mathcal{O} \times \mathcal{C})^T$. Through this coalgebraic representation of reconfiguration, there exist a number of notions and results on coalgebras in general, which can be applied to reconfiguration.

As a result, a reconfiguration process is defined by the reconfiguration function $\langle o_{\mathcal{C}}, e_{\mathcal{C}} \rangle$ such that

$$\langle o_{\mathcal{C}}, e_{\mathcal{C}} \rangle : \mathcal{C} \longrightarrow (\mathcal{O} \times \mathcal{C})^T \tag{17}$$

Thus, the set $\mathcal{C}$ of configurations equipped by the structure (17) defines a category $\mathbf{Conf}(\odot\frown\odot\,(\_))$, whose objects are configurations of the set $\mathcal{C}$ and morphisms can be explicitly drawn as follows: For each $c \in \mathcal{C}$,

$$c \xrightarrow{\langle o_{\mathcal{C}}, e_{\mathcal{C}} \rangle} \odot\frown\odot\,(c) \xrightarrow{\odot\frown\odot(\langle o_{\mathcal{C}}, e_{\mathcal{C}} \rangle)} \odot\frown\odot^2\,(c) \xrightarrow{\odot\frown\odot^2(\langle o_{\mathcal{C}}, e_{\mathcal{C}} \rangle)} \odot\frown\odot^3\,(c) \xrightarrow{\odot\frown\odot^3(\langle o_{\mathcal{C}}, e_{\mathcal{C}} \rangle)} \ldots \tag{18}$$

The reconfiguration function $\langle o_{\mathcal{C}}, e_{\mathcal{C}}\rangle$ in (17) can be represented in the following different way

$$\langle o_{\mathcal{C}}, e_{\mathcal{C}}\rangle_{\bullet} : \mathcal{C} \longrightarrow ((T \times \mathcal{O}) \rightarrowtail \mathcal{C}) \tag{19}$$

Similarly, with this result, the set $\mathcal{C}$ equipped by the structure (19) defines a category $\mathbf{Conf}(\odot(\_))$, whose objects are configurations of the set $\mathcal{C}$. Let

$$\odot(\_) \;=\; (T \times \mathcal{O}) \rightarrowtail \_ \tag{20}$$

then morphisms of the category $\mathbf{Conf}(\odot(\_))$ can be explicitly viewed as

$$c \xrightarrow{\langle o_{\mathcal{C}}, e_{\mathcal{C}}\rangle_{\bullet}} \odot\,(c) \xrightarrow{\odot(\langle o_{\mathcal{C}}, e_{\mathcal{C}}\rangle_{\bullet})} \odot^2\,(c) \xrightarrow{\odot^2(\langle o_{\mathcal{C}}, e_{\mathcal{C}}\rangle_{\bullet})} \odot^3\,(c) \xrightarrow{\odot^3(\langle o_{\mathcal{C}}, e_{\mathcal{C}}\rangle_{\bullet})} \ldots \tag{21}$$

for each $c \in \mathcal{C}$.

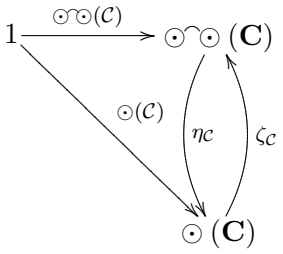
A bijective relationship between the representation (17) and (19) of the reconfiguration function is justified by proving that there exists a *natural equivalence* (also called *natural isomorphism*) between two functors $\odot(\_) = (\mathcal{O} \times \_)^T$ and $\odot\frown\odot\,(\_) \;=\; ((T \times \mathcal{O}) \rightarrowtail \_)$. In fact,

Proposition 1. *Consider a parallel pair*

$$\mathbf{C} \underset{\odot(\_)}{\overset{\odot\frown\odot(\_)}{\rightrightarrows}} \mathbf{C} \tag{22}$$

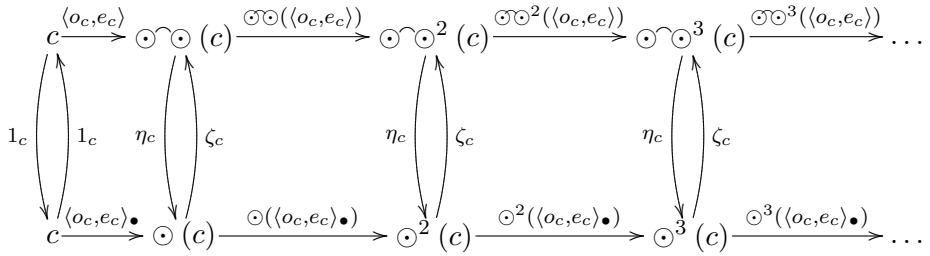
of two functors of $\odot\frown\odot\,(\_)$ *and* $\odot(\_)$. $\odot\frown\odot\,(\_)$ *and* $\odot(\_)$ *are naturally equivalent.*

Proof. For morphisms $1 \xrightarrow{\odot\frown\odot(\mathcal{C})} \odot\frown\odot\,(\mathbf{C})$ and $1 \xrightarrow{\odot(\mathcal{C})} \odot\,(\mathbf{C})$, there exists a unique morphism $\odot\frown\odot\,(\mathbf{C}) \xrightarrow{\eta_{\mathcal{C}}} \odot\,(\mathbf{C})$ such that the equation $\odot\frown\odot\,(\mathcal{C}); \eta_{\mathcal{C}} = \odot(\mathcal{C})$ holds. Inversely, there exists a unique morphism $\odot\,(\mathbf{C}) \xrightarrow{\zeta_{\mathcal{C}}} \odot\frown\odot\,(\mathbf{C})$ such that the equation $\odot(\mathcal{C}); \zeta_{\mathcal{C}} = \odot\frown\odot(\mathcal{C})$ holds. This is described by the following commutative diagram.



(23)

By (23), for each c in $\mathcal{C}$, the following diagram (constructed by (18) and (21)) commutes.



With this result, $\eta_{\mathcal{C}}$ and $\zeta_{\mathcal{C}}$ are an inverse pair of natural isomorphisms in $\mathbf{C}$ between two functors of $\odot\frown\odot\,(\_)$ and $\odot(\_)$. Hence, $\odot\frown\odot\,(\_)$ and $\odot(\_)$ are naturally equivalent. □

Note that using either of the above-mentioned representations (17) and (19) is completely dependent on aspect that we want to treat. Thus, for a closely structural relationship between AOMRC and CSP, in the remainder of the paper, the representation of reconfiguration function in (19) is chosen to investigate and analyze CSP-based behaviors of AOMRC. In section 5, we will justify that (19) and CSP deterministic process match well each other on their structural representations.

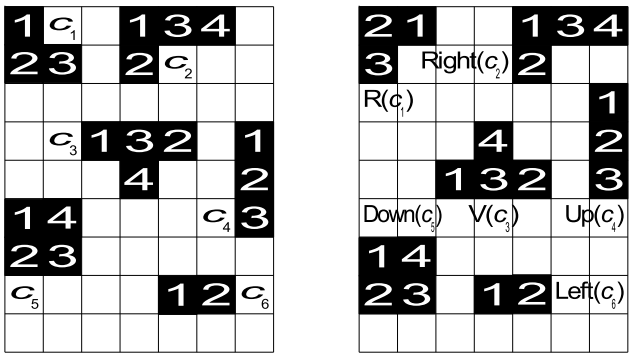
4.2 Transformations of configuration

For reconfigurable computing, in [22] we have shown that the set of transformations $T = \{\mathbf{R}, \mathbf{V}, \mathbf{Up}, \mathbf{Down}, \mathbf{Left}, \mathbf{Right}\}$ is sufficient to change configurations as modules in a $2D$ affine space where the elements of T are understood as follows (see Figure 2): $\mathbf{R}$ rotates a configuration $c \in \mathcal{C}$ by 90^0 clockwise. $\mathbf{V}$ vertically flips a configuration $c \in \mathcal{C}$. **Up** moves a configuration vertically up. **Down** moves a configuration vertically down. **Left** moves a configuration horizontally in the left direction. **Right** moves a configuration horizontally in the right direction. Moreover, in a specific way [22], we have considered the output set $\mathcal{O}$ as $\mathbf{2} = \{0, 1\}$, where output is 1 when a change of module $c \in \mathcal{C}$ is successfully executed and 0 otherwise.

As a reasoning in section 4.1, we choose the model of reconfiguration such that

$$\langle o_{\mathcal{C}}, e_{\mathcal{C}}\rangle_\bullet : \mathcal{C} \longrightarrow ((T \times \mathbf{2}) \rightarrowtail \mathcal{C}) \tag{24}$$

in which the set T of transformations is extended to $(T \times \mathbf{2})$ that consists of twelve transformations (see an illustration in Figure 3).



(a) Current configurations $c_1 - c_5$ (b) Changed configurations

Figure 2: Changing configurations by transformations in T

Each extended transformation in $(T \times \mathbf{2})$ is a unary operation. Formally, each $(t, o) \in (T \times \mathbf{2})$ is recognized as an operation

$$(t, o) : \mathcal{C} \ \rightarrowtail \ \mathcal{C} \tag{25}$$

taking a configuration input of type $\mathcal{C}$ and producing a configuration output of type $\mathcal{C}$. Moreover, these transformations are defined such that

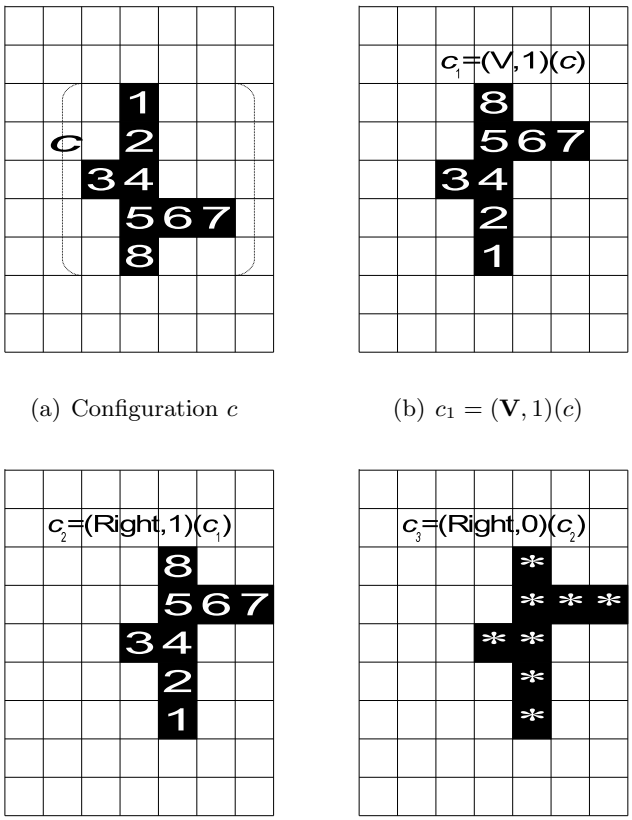
$$(t, o)(c) = \begin{cases} * & \text{when } o = 0 \text{ and} \\ (t, 1)(c) & \text{otherwise} \end{cases} \tag{26}$$

where $\{*\}$ is an extra configuration in $\mathcal{C}$, in this case no successor configuration in $\mathcal{C}$ is produced. In other words, it corresponds to stop of reconfiguration.

4.3 A transformations-based aspect in reconfigurable computing

At a control step $i \in \mathbb{N}$, a modular reconfigurable computing system $\mathcal{M}$ is composed by n *configuration modules* denoted by $c^i_\Diamond \in \mathcal{C}, \forall \Diamond \in \{1, \ldots, n\}$, whose subscript and superscript relate to module number and control step number, respectively.

We see that aspect-oriented development of the modular system $\mathcal{M}$ is really a refactoring process (or decomposing process) to discover so-called



(a) Configuration c (b) $c_1 = (\mathbf{V}, 1)(c)$

(c) $c_2 = (\mathbf{Right}, 1)(c_1)$ (d) $c_3 = (\mathbf{Right}, 0)(c_2)$

Figure 3: Changing configuration by transformations in $(T \times \mathbf{2})$

common factors of $\mathcal{M}$. Those common factors are called *aspects* [17]. In fact, for $t^i_\Diamond \in T$ and $o^i_\Diamond \in \mathbf{2}$, an aspect $\mathcal{A} = \langle (t^i_1, o^i_1), (t^i_2, o^i_2), \ldots, (t^i_n, o^i_n) \rangle$ is viewed as a transformation on the modular reconfigurable computing system $\mathcal{M} = \langle c^{i-1}_1, c^{i-1}_2, \ldots, c^{i-1}_n \rangle$ including n configuration modules $c^{i-1}_\Diamond, \forall \Diamond \in \{1, \ldots, n\}$.

We define an application of $\mathcal{A}$ to $\mathcal{M}$ (denoted by $\mathcal{A}(\mathcal{M})$) to determine the modular resulting reconfigurable computing system $\mathcal{M}' = \langle c^i_1, c^i_2, \ldots, c^i_n \rangle$ $= \langle (t^i_1, o^i_1)(c^{i-1}_1), (t^i_2, o^i_2)(c^{i-1}_2), \ldots, (t^i_n, o^i_n)(c^{i-1}_n) \rangle$. That is,

$$\mathcal{A}(\mathcal{M}) = \mathcal{M}' = \langle (t^i_1, o^i_1)(c^{i-1}_1), (t^i_2, o^i_2)(c^{i-1}_2), \ldots, (t^i_n, o^i_n)(c^{i-1}_n) \rangle$$

$$= \begin{cases} \star \quad \text{when } \exists o^i_\Diamond = 0 \text{ for } \Diamond \in \{1, \ldots, n\} \text{ and} \\ (t^i_1, 1)(c^{i-1}_1), (t^i_2, 1)(c^{i-1}_2), \ldots, (t^i_n, 1)(c^{i-1}_n) \quad \text{otherwise} \end{cases}$$

where $\{\star\}$ is an extra reconfigurable computing system, in this context no next reconfigurable computing system is produced and the reconfiguration process of whole system stops.

4.3.1 An example of aspect and its applications

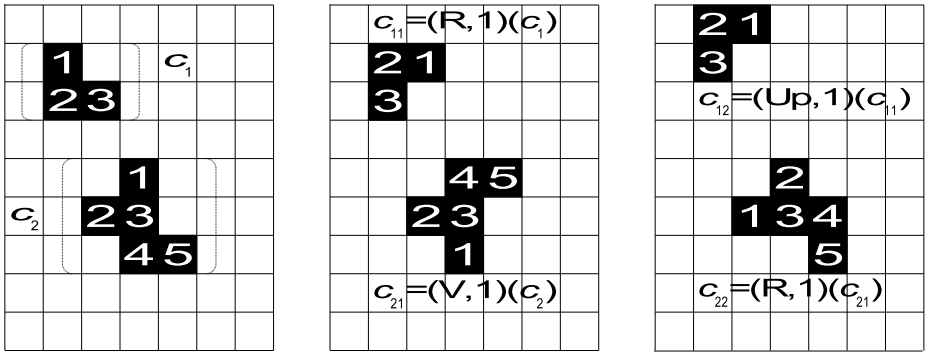
From now on, we omit the superscripts (relating to control step number) in all notations for short, as the example is considered at an arbitrary control step. Let $\mathcal{M}$ be a modular reconfigurable computing system consisting of two modules of configuration $c_1 \in \mathcal{C}$ and $c_2 \in \mathcal{C}$ as in Figure 4(a). From the transformations in $(T \times \mathbf{2})$ considered in subsection 4.2, we construct the aspect $\mathcal{A} = \langle (t_1, o_1), (t_2, o_2) \rangle$, where $\forall (t_1, o_1), (t_2, o_2) \in (T \times \mathbf{2})$, to be applied on $\mathcal{M} = \langle c_1, c_2 \rangle$. In this case, the application of $\mathcal{A} = \langle (t_1, o_1), (t_2, o_2) \rangle$ to $\mathcal{M}$ is written as $\langle (t_1, o_1), (t_2, o_2) \rangle (\langle c_1, c_2 \rangle) = \langle (t_1, o_1)(c_1), (t_2, o_2)(c_2) \rangle$. As an exemplification, here we illustrate only six instances of the modular resulting system as in Figure 4. Specifically, $\langle (\mathbf{R}, 1)(c_1), (\mathbf{V}, 1)(c_2) \rangle$ produces the resulting system in Figure 4(b), $\langle (\mathbf{Up}, 1)(c_1), (\mathbf{R}, 1)(c_2) \rangle$ in Figure 4(c), $\langle (\mathbf{R}, 1)(c_1), (\mathbf{Down}, 1)(c_2) \rangle$ in Figure 4(d), $\langle (\mathbf{Left}, 1)(c_1), (\mathbf{Right}, 1)(c_2) \rangle$ in Figure 4(e), $\langle (\mathbf{Up}, 0)(c_1), (\mathbf{Down}, 1)(c_2) \rangle$ and $\langle (\mathbf{Left}, 0)(c_1), (\mathbf{Right}, 1)(c_2) \rangle$ in Figure 4(f).

Note that, in Figure 4, the notation $\vdash_{\mathcal{A}}$ denotes a transition when applying the aspect $\mathcal{A}$ on a modular system. In this way, when we write $\mathcal{M} \vdash_{\mathcal{A}} \mathcal{M}_1$, for example, which means that a change from $\mathcal{M}$ to $\mathcal{M}_1$ occurs after applying the aspect $\mathcal{A}$ on $\mathcal{M}$. Moreover, when we write $\mathcal{M} \vdash_{\mathcal{A}} \mathcal{M}_1 \vdash_{\mathcal{A}} \mathcal{M}_2 \vdash_{\mathcal{A}} \mathcal{M}_3 \vdash_{\mathcal{A}} \mathcal{M}_4 \vdash_{\mathcal{A}} STOP$ that is understood as an application process of the aspect $\mathcal{A}$ by which the change happens.

4.3.2 Bisimulation between two modular reconfigurable computing systems as inheritance property of aspect

A bisimulation between two modular reconfigurable computing systems $\mathcal{M} = \langle c_1, c_2, \ldots, c_n \rangle$ and $\mathcal{M}' = \langle c'_1, c'_2, \ldots, c'_n \rangle$ is a relation $L \subseteq \mathcal{M} \times \mathcal{M}'$ with, for all $\langle c_1, c_2, \ldots, c_n \rangle$ in $\mathcal{M}$, $\langle c'_1, c'_2, \ldots, c'_n \rangle$ in $\mathcal{M}'$ and $\langle (t_1, o_1), (t_2, o_2), \ldots, (t_n, o_n) \rangle$ in $\mathcal{A}$.

If $\langle c_1, c_2, \ldots, c_n \rangle \; L \; \langle c'_1, c'_2, \ldots, c'_n \rangle$ then $c_\Diamond \overset{(t_\Diamond, o_\Diamond)}{\sim} c'_\Diamond, \forall \; \Diamond \in \{1, \ldots, n\}$. We write $c_\Diamond \overset{(t_\Diamond, o_\Diamond)}{\sim} c'_\Diamond$ when $c_\Diamond \sim c'_\Diamond$ under transformations $(t_\Diamond, o_\Diamond) \in (T \times \mathbf{2})$.



(a) A modular system ($\mathcal{M} = \langle c_1, c_2 \rangle$)

(b) $\mathcal{M} \vdash_{\mathcal{A}} \mathcal{M}_1 = \langle (\mathbf{R}, 1), (\mathbf{V}, 1) \rangle (\mathcal{M})$

(c) $\mathcal{M}_1 \vdash_{\mathcal{A}} \mathcal{M}_2 = \langle (\mathbf{Up}, 1), (\mathbf{R}, 1) \rangle (\mathcal{M}_1)$

(d) $\mathcal{M}_2 \vdash_{\mathcal{A}} \mathcal{M}_3 = \langle (\mathbf{R}, 1), (\mathbf{Down}, 1) \rangle (\mathcal{M}_2)$

(e) $\mathcal{M}_3 \vdash_{\mathcal{A}} \mathcal{M}_4 = \langle (\mathbf{Left}, 1), (\mathbf{Right}, 1) \rangle (\mathcal{M}_3)$

(f) $\mathcal{M}_4 \vdash_{\mathcal{A}} STOP$

Figure 4: An application process of aspect $\mathcal{A}$ on the modular systems

In other words, If $\langle\langle c_1, c_2, \ldots, c_n\rangle, \langle c'_1, c'_2, \ldots, c'_n\rangle\rangle \in L$ then $\langle c_\Diamond, c'_\Diamond\rangle_{(t_\Diamond, o_\Diamond)} \in R$, where $\langle c_\Diamond, c'_\Diamond\rangle_{(t_\Diamond, o_\Diamond)}$ in R denotes $\langle c_\Diamond, c'_\Diamond\rangle$ in R under transformations $(t_\Diamond, o_\Diamond)$ in $(T \times \mathbf{2})$.

We also write $\mathcal{M} \sim \mathcal{M}'$ whenever there exists a bisimulation L with $\langle c_1, c_2, \ldots, c_n\rangle \; L \; \langle c'_1, c'_2, \ldots, c'_n\rangle$.

Note that the superscripts (relating to control step number) in all notations can be omitted for short, as the considered context is at an arbitrary control step.

Two modular reconfigurable computing systems $\mathcal{M}$ and $\mathcal{M}'$ that are related by a bisimulation relation are observationally indistinguishable in that:

- Each member module in $\mathcal{M}$ and its partner module in $\mathcal{M}'$ cause the same observations, and
- Applying the same aspect to both systems will lead to two resulting systems that are indistinguishable again.

For further well-founded investigation, we justify some following statements.

Proposition 2. *Let $\mathcal{M}'$ and $\mathcal{M}''$ be applications of aspect $\mathcal{A}(\mathcal{M})$. If $\mathcal{M} \sim \mathcal{M}'$ and $\mathcal{M}' \sim \mathcal{M}''$ then $(\mathcal{M} \sim \mathcal{M}') \circ (\mathcal{M}' \sim \mathcal{M}'') = \mathcal{M} \sim \mathcal{M}''$, where the symbol $\circ$ denotes a relational composition. For more descriptive notation, we can write this in the form*

$$\frac{\mathcal{M} \sim \mathcal{M}', \mathcal{M}' \sim \mathcal{M}''}{(\mathcal{M} \sim \mathcal{M}') \circ (\mathcal{M}' \sim \mathcal{M}'') = \mathcal{M} \sim \mathcal{M}''} \tag{27}$$

and conversely, if $\mathcal{M} \sim \mathcal{M}''$ then there exists $\mathcal{M}'$ such that $\mathcal{M} \sim \mathcal{M}'$ and $\mathcal{M}' \sim \mathcal{M}''$. This can be written as

$$\frac{\mathcal{M} \sim \mathcal{M}''}{\exists \mathcal{M}' : \mathcal{M} \sim \mathcal{M}' \quad \text{and} \quad \mathcal{M}' \sim \mathcal{M}''} \tag{28}$$

Proof. Proving (27) originates as the result of the truth that the relational composition of two bisimulations $L_1 \subseteq \mathcal{M} \times \mathcal{M}'$ and $L_2 \subseteq \mathcal{M}' \times \mathcal{M}''$ is a bisimulation obtained by $L_1 \circ L_2 = \{\langle x, y\rangle \mid x \; L_1 \; z \text{ and } z \; L_2 \; y \text{ for some } z \in \mathcal{M}'\}$, where $x = \langle c_1, c_2, \ldots, c_n\rangle \in \mathcal{M}$, $z = \langle c'_1, c'_2, \ldots, c'_n\rangle \in \mathcal{M}'$ and $y = \langle c''_1, c''_2, \ldots, c''_n\rangle \in \mathcal{M}''$.

Proving (28) comes from the fact that there are always $\mathcal{M}' = \mathcal{M}$ or $\mathcal{M}' = \mathcal{M}''$ as simply as they can. Hence, (28) is always true in general. □

Proposition 3. *Let* $\mathcal{M}_i, \forall i \in \mathbb{N}$, *be applications of aspect* $\mathcal{A}(\mathcal{M})$ *and* $\bigcup_{i\in\mathbb{N}}$ *be union of a family of sets. Then*

$$\frac{\mathcal{M} \sim \mathcal{M}_i \quad \text{with } i \in \mathbb{N}}{\bigcup_{i\in\mathbb{N}}(\mathcal{M} \sim \mathcal{M}_i) = \mathcal{M} \sim \bigcup_{i\in\mathbb{N}} \mathcal{M}_i} \tag{29}$$

and conversely,

$$\frac{\mathcal{M} \sim \bigcup_{i\in\mathbb{N}} \mathcal{M}_i}{\exists i \in \mathbb{N} : \mathcal{M} \sim \mathcal{M}_i} \tag{30}$$

Proof. Proving (29) stems straightforwardly from the fact that $\mathcal{M}$ bisimulates $\mathcal{M}_i$ (i.e., $\mathcal{M} \sim \mathcal{M}_i$) then, $\mathcal{M}$ bisimulates each system in $\bigcup_{i\in\mathbb{N}} \mathcal{M}_i$.

Conversely, proving (30) develops as the result of the fact that for each $\langle x, y\rangle \in \bigcup_{i\in\mathbb{N}}(\mathcal{M} \times \mathcal{M}_i)$, there exists $i \in \mathbb{N}$ such that $\langle x, y\rangle \in \mathcal{M} \times \mathcal{M}_i$. In other words, it is formally denoted by $\bigcup_{i\in\mathbb{N}}(\mathcal{M} \times \mathcal{M}_i) = \{\langle x, y\rangle \mid \exists i \in \mathbb{N} : x \in \mathcal{M} \quad \text{and} \quad y \in \mathcal{M}_i\}$, where $x = \langle c_1, c_2, \ldots, c_n\rangle$ and $y = \langle c'_1, c'_2, \ldots, c'_n\rangle$. □

The union of all bisimulations between $\mathcal{M}$ and applications of aspect $\mathcal{A}(\mathcal{M})$ (i.e., $\bigcup\ (\mathcal{M} \sim \mathcal{A}(\mathcal{M}))$) is the greatest bisimulation. The greatest bisimulation is called the *bisimulation equivalence* or *bisimilarity* [16, 21] (again denoted by the notation $\sim$).

Proposition 4. *The bisimilarity* $\sim$ *on* $\bigcup\ (\mathcal{M} \sim \mathcal{A}(\mathcal{M}))$ *is an equivalence relation.*

Proof. In fact, a bisimilarity $\sim$ on $\bigcup\ (\mathcal{M} \sim \mathcal{A}(\mathcal{M}))$ is a binary relation $\sim$ on $\bigcup\ (\mathcal{M} \sim \mathcal{A}(\mathcal{M}))$, which is reflexive, symmetric and transitive. In other words, the following properties hold for $\sim$

- Reflexivity:
$$\frac{\forall(a \sim b) \in \bigcup\ (\mathcal{M} \sim \mathcal{A}(\mathcal{M}))}{(a \sim b) \sim (a \sim b)}$$
- Symmetry:
$$\frac{\forall(a \sim b), (c \sim d) \in \bigcup\ (\mathcal{M} \sim \mathcal{A}(\mathcal{M})), \quad (a \sim b) \sim (c \sim d)}{(c \sim d) \sim (a \sim b)}$$
- Transitivity:
$$\frac{\forall(a \sim b), (c \sim d), (e \sim f) \in \bigcup\ (\mathcal{M} \sim \mathcal{A}(\mathcal{M})), \quad ((a \sim b) \sim (c \sim d)) \bigwedge ((c \sim d) \sim (e \sim f))}{(a \sim b) \sim (e \sim f)}$$

to be an equivalence relation on $\bigcup (\mathcal{M} \sim \mathcal{A}(\mathcal{M}))$. □

The bisimulation relation between $\mathcal{M}$ and $\mathcal{A}(\mathcal{M})$ is now shown as a semantic basis of *inheritance property* of aspect. In other words, for some condition α, we can formally represent the inheritance property as the following:

$$\frac{\mathcal{M} \models \alpha}{\mathcal{A}(\mathcal{M}) \models \alpha}$$

That is, if modular reconfigurable computing system $\mathcal{M}$ satisfies condition α then this constraint is still preserved on an application of aspect $\mathcal{A}(\mathcal{M})$. Thus it is read as $\mathcal{M} \sim \mathcal{A}(\mathcal{M})$ in the constraint of α (and denoted by $\mathcal{M} \sim_\alpha \mathcal{A}(\mathcal{M})$).

5 Homomorphism between AOMRC and Hoare Model of Deterministic Reconfiguration Processes

5.1 Coalgebraic behaviors of AOMRC

In the coalgebraic approach, the reconfiguration $\langle \mathcal{C}, \langle o_\mathcal{C}, e_\mathcal{C} \rangle_\bullet \rangle$ is considered as the structure observed through the *interface* $\odot(\_) = (T \times \mathbf{2}) \rightarrowtail \_$ because applying this interface to the observable set $\mathcal{C}$ of configurations, we have $\odot(\mathcal{C}) = (T \times \mathbf{2}) \rightarrowtail \mathcal{C}$. This interface of $\odot(\_)$ can be regarded as a mapping to decompose the observable set $\mathcal{C}$ of configurations into an observation context $(T \times \mathbf{2}) \rightarrowtail \_$ of the set $\mathcal{C}$.

Proposition 5. *$\langle \mathcal{C}, \langle o_\mathcal{C}, e_\mathcal{C} \rangle_\bullet \rangle$, whose function $\langle o_\mathcal{C}, e_\mathcal{C} \rangle_\bullet : \mathcal{C} \longrightarrow \odot(\mathcal{C})$ defines a deterministic reconfiguration process, is really an $\odot(\_)$-coalgebra of a covariant functor $\odot(\_)$.*

Proof. In fact, by the definition of coalgebra in subsection 3.3, we only need to check that the function $\mathcal{C} \longrightarrow ((T \times \mathbf{2}) \rightarrowtail \mathcal{C})$ extends to a functor $\odot(\_) : \mathbf{C} \longrightarrow \mathbf{C}$. For this we assign to any mapping $h : \mathcal{C} \longrightarrow \mathcal{C}'$ the mapping $\odot(h) : ((T \times \mathbf{2}) \rightarrowtail \mathcal{C}) \longrightarrow ((T \times \mathbf{2}) \rightarrowtail \mathcal{C}')$ with $\odot(h)(f) \stackrel{def}{=} h; f$ for all functions $f \in ((T \times \mathbf{2}) \rightarrowtail \mathcal{C})$. This defines a covariant functor $\odot(\_) : \mathbf{C} \longrightarrow \mathbf{C}$. □

An $\odot(\_)$-*coalgebra* is a pair $\langle \mathcal{C}, \langle o_\mathcal{C}, e_\mathcal{C} \rangle_\bullet \rangle$ consisting of a set $\mathcal{C}$ and a function $\langle o_\mathcal{C}, e_\mathcal{C} \rangle_\bullet : \mathcal{C} \longrightarrow \odot(\mathcal{C})$. Thus reconfiguration $\langle \mathcal{C}, \langle o_\mathcal{C}, e_\mathcal{C} \rangle_\bullet \rangle$ is a coalgebra of the functor $\odot(\_)$, which is defined on the category $\mathbf{C}$ by $\odot(\mathcal{C}) = (T \times \mathbf{2}) \rightarrowtail \mathcal{C}$.

To see what $\mathcal{C}$ is, we have to determine the reconfiguration process that starts with a configuration $c \in \mathcal{C}$. That is, we have to analyze step-by-step which configurations can be reached from c by which transformations. To this end, we unfold the reconfiguration function $\langle o_{\mathcal{C}}, e_{\mathcal{C}} \rangle_\bullet : \mathcal{C} \longrightarrow \odot(\mathcal{C})$ to obtain the following sequence:

$$c \xrightarrow{\langle o_{\mathcal{C}}, e_{\mathcal{C}} \rangle_\bullet} \odot\,(c) \xrightarrow{\odot(\langle o_{\mathcal{C}}, e_{\mathcal{C}} \rangle_\bullet)} \odot^2\,(c) \xrightarrow{\odot^2(\langle o_{\mathcal{C}}, e_{\mathcal{C}} \rangle_\bullet)} \odot^3\,(c) \xrightarrow{\odot^3(\langle o_{\mathcal{C}}, e_{\mathcal{C}} \rangle_\bullet)} \ldots \tag{31}$$

where, for any configuration $c \in \mathcal{C}$, $\langle o_{\mathcal{C}}, e_{\mathcal{C}} \rangle_\bullet : \mathcal{C} \longrightarrow \odot(\mathcal{C})$ refers to which configurations can be reached from c in one control step by which transformation.

$\odot^i(\langle o_{\mathcal{C}}, e_{\mathcal{C}} \rangle_\bullet) : \odot^i(\mathcal{C}) \longrightarrow \odot^{i+1}(\mathcal{C})$ describes how arbitrary transformations sequences of length i can be continued according to $\langle o_{\mathcal{C}}, e_{\mathcal{C}} \rangle_\bullet$ in the next control step. Starting with a configuration $c \in \mathcal{C}$ we obtain in such a way an infinite sequence

$$\mathit{reconf\_process}(c) \stackrel{def}{=} (c, \langle o_{\mathcal{C}}, e_{\mathcal{C}} \rangle^1_\bullet, \langle o_{\mathcal{C}}, e_{\mathcal{C}} \rangle^2_\bullet, \ldots) \tag{32}$$

with, for all $i \in \mathbb{N}$,

$$\begin{array}{rcl} \langle o_{\mathcal{C}}, e_{\mathcal{C}} \rangle^1_\bullet & = & \langle o_{\mathcal{C}}, e_{\mathcal{C}} \rangle_\bullet(c) \quad \in \odot(\mathcal{C}) \\ \langle o_{\mathcal{C}}, e_{\mathcal{C}} \rangle^2_\bullet & = & \odot(\langle o_{\mathcal{C}}, e_{\mathcal{C}} \rangle_\bullet)(\langle o_{\mathcal{C}}, e_{\mathcal{C}} \rangle_\bullet(c)) \quad \in \odot^2(\mathcal{C}) \\ \ldots & & \\ \langle o_{\mathcal{C}}, e_{\mathcal{C}} \rangle^{i+1}_\bullet & = & \odot^i(\langle o_{\mathcal{C}}, e_{\mathcal{C}} \rangle_\bullet)(\langle o_{\mathcal{C}}, e_{\mathcal{C}} \rangle^i_\bullet) \quad \in \odot^{i+1}(\mathcal{C}) \end{array} \tag{33}$$

where $\langle o_{\mathcal{C}}, e_{\mathcal{C}} \rangle^i_\bullet$ represents all transformations sequences of length at most i starting with c. In addition, $\langle o_{\mathcal{C}}, e_{\mathcal{C}} \rangle^i_\bullet$ also points out which configurations are reached by transformations sequences of length exactly i.

5.2 Hoare model of deterministic reconfiguration processes

5.2.1 Deterministic reconfiguration process

Based on the coalgebraic behaviors of AOMRC in subsection 5.1, the function $\langle o_{\mathcal{C}}, e_{\mathcal{C}} \rangle_\bullet : \mathcal{C} \longrightarrow \odot(\mathcal{C})$ defines a *deterministic reconfiguration process*. Formally, the CSP-based definition of deterministic reconfiguration processes is regarded as follows:

A *trace* is a sequence of transformations, separated by commas and enclosed in angular brackets, recording the events in which a process P has engaged up to some moment in time. Before the process starts, it is

not known which of its possible traces will actually be recorded: the choice will depend on environmental factors beyond the control of the process. However the complete set of all possible traces of the process P can be known in advance, and a function $traces(P)$ is defined to yield that set.

The set $(T \times \mathbf{2})^*$ is the set of all finite traces (including $\langle\rangle$) which are formed from transformations in the set $(T \times \mathbf{2})$, also called an *alphabet* of the process. When such traces are restricted to $(T \times \mathbf{2})$, they remain unchanged. This fact permits a simple definition

$$(T \times \mathbf{2})^* = \{s \in traces(P) \mid s \upharpoonright (T \times \mathbf{2}) = s\} \tag{34}$$

Note that the expression $s \upharpoonright (T \times \mathbf{2})$ denotes the trace s when restricted to transformations in the alphabet $(T \times \mathbf{2})$; it is formed from s simply by omitting all transformations outside $(T \times \mathbf{2})$.

A deterministic reconfiguration process P with alphabet $(T \times \mathbf{2})$ is defined to be a pair

$$((T \times \mathbf{2}), traces(P)) \tag{35}$$

where $traces(P)) \subseteq (T \times \mathbf{2})^*$ (read as every event that occurs must be in the alphabet of the process) which satisfies the two following conditions:

$$\langle\rangle \in traces(P) \tag{36}$$

(read as $\langle\rangle$ is a trace of every process up to the moment in which it engages in its very first event)
and

$$\forall s, t \in traces(P) : s \frown t \in traces(P) \Rightarrow s \in traces(P) \tag{37}$$

(read as if $s \frown t$ is a trace of a process up to some moment, then s must have been a trace of that process up to some earlier moment. The notation of $\frown$ is a catenation)

The simplest process which meets this definition is the one that does nothing

$$STOP_{(T \times \mathbf{2})} = ((T \times \mathbf{2}), \{\langle\rangle\}) \tag{38}$$

At the other extreme there is the process that will do anything at any time

$$RUN_{(T \times \mathbf{2})} = ((T \times \mathbf{2}), (T \times \mathbf{2})^*) \tag{39}$$

5.2.2 Hoare model of deterministic reconfiguration processes

To see what and why Hoare model of deterministic reconfiguration processes is, we consider the basic notion of deterministic reconfiguration process based on CSP.

The reconfiguration process which first engages in the transformation $(t, o) \in (T \times \mathbf{2})$ and then behaves exactly as the reconfiguration process P is specified by the Hoare's CSP syntax as

$$(t, o) \to P \tag{40}$$

Note that the notation of $\to$ is a *prefix* operator that combines a transformation and a reconfiguration process to produce a new reconfiguration process.

In general, the specification (40) is thought of as every deterministic reconfiguration process P with alphabet $(T \times \mathbf{2})$ can be considered as a function F with a domain $B \subseteq (T \times \mathbf{2})$, defining the set of transformations in which the reconfiguration process P is initially prepared to engage; and for each (t, o) in B, the deterministic reconfiguration process $F((t, o))$ defines the future behavior of the reconfiguration process P if the first transformation is (t, o). In other words, let $D\mathcal{C}_{(T\times\mathbf{2})}$ be the set of all deterministic reconfiguration processes with alphabet $(T \times \mathbf{2})$, every deterministic reconfiguration process $P \in D\mathcal{C}_{(T\times\mathbf{2})}$ can be uniquely described by a partial function

$$F : (T \times \mathbf{2}) \rightarrowtail D\mathcal{C}_{(T\times\mathbf{2})} \tag{41}$$

with domain $\mathsf{dom}(F) = B$.

A model, called *Hoare model*, of deterministic reconfiguration processes is built by considering the mathematical model (41) of deterministic reconfiguration processes represented in (40).

As mentioned above, $D\mathcal{C}_{(T\times\mathbf{2})}$ stands for the set of all prefix closed subsets of $(T \times \mathbf{2})^*$ thus the *Hoare model of deterministic reconfiguration processes* with alphabet $(T \times \mathbf{2})$ is given by the following bijective mapping

$$next_{(T\times\mathbf{2})} : D\mathcal{C}_{(T\times\mathbf{2})} \longrightarrow ((T \times \mathbf{2}) \rightarrowtail D\mathcal{C}_{(T\times\mathbf{2})}) \tag{42}$$

where $((T \times \mathbf{2}) \rightarrowtail D\mathcal{C}_{(T\times\mathbf{2})})$ denotes the set of all partial functions from $(T \times \mathbf{2})$ into $D\mathcal{C}_{(T\times\mathbf{2})}$.

For instance, $STOP_{(T\times\mathbf{2})}$ is the reconfiguration process uniquely determined by the condition $\mathsf{dom}(next_{(T\times\mathbf{2})}(STOP_{(T\times\mathbf{2})})) = \emptyset$. $RUN_{(T\times\mathbf{2})}$, i.e., the deterministic reconfiguration process which at all times can engage in any transformation of $(T \times \mathbf{2})$, can be described uniquely by the conditions $\mathsf{dom}(next_{(T\times\mathbf{2})}(RUN_{(T\times\mathbf{2})})) = (T \times \mathbf{2})$ and $next_{(T\times\mathbf{2})}(RUN_{(T\times\mathbf{2})})((t, o)) = RUN_{(T\times\mathbf{2})}$ for all $(t, o) \in (T \times \mathbf{2})$.

Moreover, $next_{(T\times\mathbf{2})}$ is bijective since we can assign to any partial function $F : (T \times \mathbf{2}) \rightarrowtail DC_{(T\times\mathbf{2})}$ the prefix closed set $next^{-1}_{(T\times\mathbf{2})}(F) = \{\langle\rangle\} \cup \{\langle(t, o)\rangle \frown (t', o')|(t, o) \in \mathsf{dom}(F) \wedge (t', o') \in F((t, o))\}$. The domain of $next_{(T\times\mathbf{2})}(P)$ is defined by $\mathsf{dom}(next_{(T\times\mathbf{2})}(P)) = \{(t, o)|\langle(t, o)\rangle \in P\}$. For any $(t, o) \in \mathsf{dom}(next_{(T\times\mathbf{2})}(P))$, $next_{(T\times\mathbf{2})}(P)((t, o))$ is defined by $next_{(T\times\mathbf{2})}(P)((t, o)) = \{(t', o')|\langle(t, o)\rangle \frown (t', o') \in P\}$.

In the next subsection, we will have an in-depth consideration on behaviors of the model (42) that is seen as a final object in the category of all deterministic reconfigurations with alphabet $(T \times \mathbf{2})$.

5.3 Coalgebraic behaviors of a Hoare model of deterministic reconfiguration processes

The Hoare model of deterministic reconfiguration processes in (42) is an $\odot(\_)$-coalgebra $\langle DC_{(T\times\mathbf{2})}, next_{(T\times\mathbf{2})}\rangle$. In fact, we have

Proposition 6. *$\langle DC_{(T\times\mathbf{2})}, next_{(T\times\mathbf{2})}\rangle$, whose function $next_{(T\times\mathbf{2})} : DC_{(T\times\mathbf{2})} \longrightarrow \odot(DC_{(T\times\mathbf{2})})$ defines the Hoare model of deterministic reconfiguration process, is really an $\odot(\_)$-coalgebra of a contravariant functor $\odot(\_)$.*

Proof. It is similar to the proposition 31, we only need to check that the function $next_{(T\times\mathbf{2})}$ extends to a functor $\odot(\_) : \mathbf{C} \longrightarrow \mathbf{C}$. For this we assign to any mapping $h : DC_{(T\times\mathbf{2})} \longrightarrow DC'_{(T\times\mathbf{2})}$ the mapping $\odot(h) : ((T \times \mathbf{2}) \rightarrowtail DC_{(T\times\mathbf{2})}) \longrightarrow ((T \times \mathbf{2}) \rightarrowtail DC'_{(T\times\mathbf{2})})$ with $\odot(h)(f) \stackrel{def}{=} f; h$ for all functions $f \in ((T \times \mathbf{2}) \rightarrowtail DC_{(T\times\mathbf{2})})$. This defines a contravariant functor $\odot(\_) : \mathbf{C} \longrightarrow \mathbf{C}$. □

To see what $DC_{(T\times\mathbf{2})}$ is, we unfold the function $next_{(T\times\mathbf{2})} : DC_{(T\times\mathbf{2})} \longrightarrow \odot(DC_{(T\times\mathbf{2})})$ to obtain the following sequence, also called ω^{op}-*chain*.

$$1_X \xleftarrow{!} \odot(1_X) \xleftarrow{\odot(!)} \odot^2(1_X) \xleftarrow{\odot^2(!)} \odot^3(1_X) \xleftarrow{\odot^3(!)} \cdots \tag{43}$$

This diagram means that the ω^{op}-chain is constructed by applying successively the functor $\odot(\_) = ((T \times \mathbf{2}) \rightarrowtail \_)$ to the unique mapping

$! : \odot(1_X) \longrightarrow 1_X$ (note that $1_X = \{X\}$ is a singleton set including only a process variable X).

Moreover, for all $i \in \mathbb{N}$, $\odot^i(1_X)$ can be considered as nested functions of depth less than or equal to i such that

$$\begin{array}{rcl} \odot(1_X) & = & ((T \times \mathbf{2}) \rightarrowtail 1_X) \\ \odot^2(1_X) & = & ((T \times \mathbf{2}) \rightarrowtail ((T \times \mathbf{2}) \rightarrowtail 1_X)) \\ \ldots & & \\ \odot^i(1_X) & = & ((T \times \mathbf{2}) \rightarrowtail \odot^{i-1}(1_X)) \end{array}$$

$! : \odot(1_X) \longrightarrow 1_X$ maps each function in $\odot(1_X) = ((T \times \mathbf{2}) \rightarrowtail 1_X)$ to X. Thus $\odot(!) = \_ ; ! : ((T \times \mathbf{2}) \rightarrowtail ((T \times \mathbf{2}) \rightarrowtail 1_X)) \longrightarrow ((T \times \mathbf{2}) \rightarrowtail 1_X)$ maps each $g \in ((T \times \mathbf{2}) \rightarrowtail ((T \times \mathbf{2}) \rightarrowtail 1_X))$ to $g; ! \in ((T \times \mathbf{2}) \rightarrowtail 1_X)$ with $\mathsf{dom}(g; !) = \mathsf{dom}(g) \subseteq (T \times \mathbf{2})$ and $(g; !)((t, o)) = !(g((t, o))) = X$ for all $(t, o) \in \mathsf{dom}(g; !)$. In general, we have $\odot^i(!) : \odot^{i+1}(1_X) \longrightarrow \odot^i(1_X)$.

The elements of $D\mathcal{C}_{(T \times \mathbf{2})}$ are infinite sequences $(X, g_1, g_2, \ldots)$ of nested functions (i.e., $g_i \in \odot^i(1_X)$ for all $i \in \mathbb{N}$) such that $\odot^i(!)(g_{i+1}) = g_i$. In such a way $(X, g_1, g_2, \ldots)$ is considered as approximation of a possibly infinite process. $(X, g_1, g_2, \ldots)$ will become $(g_1, g_2, \ldots)$ to represent a finite process (i.e., no process variable X in the process expression) if $g_i = g_{i+1}$ for some $i \in \mathbb{N}$ and thus $g_i = g_j$ for all $i \leq j$.

A significant property of the $\odot(\_)$-coalgebra $\langle D\mathcal{C}_{(T \times \mathbf{2})}, next_{(T \times \mathbf{2})} \rangle$ is that it is a *final* $\odot(\_)$-coalgebra of the contravariant functor $\odot(\_) = ((T \times \mathbf{2}) \rightarrowtail \_)$. Before proving this, we start with the definition of final coalgebra in subsection 3.3 to see what a final coalgebra is.

Proposition 7. *$\langle D\mathcal{C}_{(T \times \mathbf{2})}, next_{(T \times \mathbf{2})} \rangle$ is a final coalgebra in the category* $\mathbf{CoAlg}(\odot(\_))$ *of $\odot(\_)$-coalgebras.*

Proof. A complete proof of this proposition is the content of subsection 5.4. □

5.4 Unique homomorphism between AOMRC and a Hoare model of deterministic reconfiguration processes

To prove the unique homomorphism between AOMRC and the Hoare model of deterministic reconfiguration processes in the category $\mathbf{CoAlg}(\odot(\_))$ of $\odot(\_)$-coalgebras, we have to justify two themes, namely *existence* and *uniqueness* of a homomorphism $h : \mathcal{C} \longrightarrow 1_X$. These two themes give us two following principles:

- *Coinductive definition principle*, by which the existence theme tells us how to obtain the homomorphism $h : \mathcal{C} \longrightarrow 1_X$.
- *Conductive proof principle*, by which the uniqueness theme tells us how to prove that two homomorphisms $h, h' : \mathcal{C} \longrightarrow 1_X$ are equal.

In other words, we must prove that the following diagram (44) commutes (by the coinductive definition principle), and then the commutativity is unique (by the conductive proof principle).

$$\begin{array}{ccccccc}
\mathcal{C} & \xrightarrow{\langle o_{\mathcal{C}}, e_{\mathcal{C}}\rangle_\bullet} & \odot(\mathcal{C}) & \xrightarrow{\odot(\langle o_{\mathcal{C}}, e_{\mathcal{C}}\rangle_\bullet)} & \odot^2(\mathcal{C}) & \xrightarrow{\odot^2(\langle o_{\mathcal{C}}, e_{\mathcal{C}}\rangle_\bullet)} & \cdots \\
\downarrow{\scriptstyle h} & & \downarrow{\scriptstyle \odot(h)} & & \downarrow{\scriptstyle \odot^2(h)} & & \\
1_X & \xleftarrow{\;!\;} & \odot(1_X) & \xleftarrow{\odot(!)} & \odot^2(1_X) & \xleftarrow{\odot^2(!)} & \odot^3(1_X)\cdots
\end{array} \tag{44}$$

Note that diagram (44) is composed by two unfolding diagrams of the coalgebras $\langle \mathcal{C}, \langle o_{\mathcal{C}}, e_{\mathcal{C}}\rangle_\bullet\rangle$ (in top line) and $\langle D\mathcal{C}_{(T\times\mathbf{2})}, next_{(T\times\mathbf{2})}\rangle$ (in bottom line) under the mappings $\odot^i(h) : \odot^i(\mathcal{C}) \longrightarrow \odot^i(1_X)$ for every i in $\mathbb{N}_0$ ($= \mathbb{N}\cup\{0\}$).

To prove the commutativity of diagram (44), we just need to prove that the following diagram (45) commutes for every i in $\mathbb{N}_0$.

$$\begin{array}{ccccc}
\cdots\odot^i(\mathcal{C}) & \xrightarrow{\odot^i(\langle o_{\mathcal{C}}, e_{\mathcal{C}}\rangle_\bullet)} & \odot^{i+1}(\mathcal{C})\cdots \\
\downarrow{\scriptstyle \odot^i(h)} & & \downarrow{\scriptstyle \odot^{i+1}(h)} \\
\cdots\odot^i(1_X) & \xleftarrow{\odot^i(!)} & \odot^{i+1}(1_X)\cdots
\end{array} \tag{45}$$

By $\langle o_{\mathcal{C}}, e_{\mathcal{C}}\rangle_\bullet^i \in \odot^i(\mathcal{C})$ and $g_i \in \odot^i(1_X)$, diagram (45) can be more specifically represented by

$$\begin{array}{ccccc}
\cdots\langle o_{\mathcal{C}}, e_{\mathcal{C}}\rangle_\bullet^i \in \odot^i(\mathcal{C}) & \xrightarrow{\odot^i(\langle o_{\mathcal{C}}, e_{\mathcal{C}}\rangle_\bullet)} & \langle o_{\mathcal{C}}, e_{\mathcal{C}}\rangle_\bullet^{i+1} \in \odot^{i+1}(\mathcal{C})\cdots \\
\downarrow{\scriptstyle \odot^i(h)} & & \downarrow{\scriptstyle \odot^{i+1}(h)} \\
\cdots g_i \in \odot^i(1_X) & \xleftarrow{\odot^i(!)} & g_{i+1} \in \odot^{i+1}(1_X)\cdots
\end{array} \tag{46}$$

For every $i \in \mathbb{N}_0$ and $g_i \in \odot^i(1_X)$, we define $g_i \stackrel{def}{=} \odot^i(h)(\langle o_{\mathcal{C}}, e_{\mathcal{C}}\rangle^i_\bullet)$. Hence,

$$\begin{array}{rcl} g_i & = & \odot^i(!)(g_{i+1}), \qquad \text{by (43)} \\ & = & \odot^i(!)(\odot^{i+1}(h)(\langle o_{\mathcal{C}}, e_{\mathcal{C}}\rangle^{i+1}_\bullet)), \quad \text{by the definition of } g_i \\ & = & \odot^i(!)(\odot^{i+1}(h)(\odot^i(\langle o_{\mathcal{C}}, e_{\mathcal{C}}\rangle_\bullet)(\langle o_{\mathcal{C}}, e_{\mathcal{C}}\rangle^i_\bullet))), \qquad \text{by (33)} \end{array}$$

With this result, it yields $\odot^i(!)(\odot^{i+1}(h)(\odot^i(\langle o_{\mathcal{C}}, e_{\mathcal{C}}\rangle_\bullet)(\langle o_{\mathcal{C}}, e_{\mathcal{C}}\rangle^i_\bullet))) = \odot^i(h)$ $(\langle o_{\mathcal{C}}, e_{\mathcal{C}}\rangle^i_\bullet)$. Generally, for every i in $\mathbb{N}_0$, $\odot^i(\langle o_{\mathcal{C}}, e_{\mathcal{C}}\rangle_\bullet); \odot^{i+1}(h); \odot^i(!) = \odot^i(h)$. This means that diagram (45) commutes. That is, the mapping h is an $\odot(\_)$-homomorphism.

The uniqueness of h to be a unique $\odot(\_)$-homomorphism stems straightforwardly from the *coinductive proof principle* [16,21]. For validating whether two $\odot(\_)$-homomorphisms h and h' are identical, i.e. $h = h'$, a powerful method is so-called proof principle of coinduction that states as follows:

$$\frac{h \sim h'}{h = h'} \tag{47}$$

It means that if a bisimulation $\sim$ between h and h' is established then h and h' are identical. Hence, in order to prove the identicalness between two $\odot(\_)$-homomorphisms h and h', it is sufficient to establish the existence of a bisimulation relation $h \sim h'$.

Suppose that there are two homomorphisms $h(\langle o_{\mathcal{C}}, e_{\mathcal{C}}\rangle^i_\bullet)$ and $h'(\langle o_{\mathcal{C}}, e_{\mathcal{C}}\rangle^i_\bullet)$. Let $\sim = \{\langle h(\langle o_{\mathcal{C}}, e_{\mathcal{C}}\rangle^i_\bullet), h'(\langle o_{\mathcal{C}}, e_{\mathcal{C}}\rangle^i_\bullet)\rangle \mid \langle o_{\mathcal{C}}, e_{\mathcal{C}}\rangle^i_\bullet \text{ is in } \odot^i(\mathcal{C})\}$. We prove that $\sim$ is a bisimulation on $\odot^i(1_X)$. Consider $\langle h(\langle o_{\mathcal{C}}, e_{\mathcal{C}}\rangle^i_\bullet), h'(\langle o_{\mathcal{C}}, e_{\mathcal{C}}\rangle^i_\bullet)\rangle \in \sim$, then

$$h(\langle o_{\mathcal{C}}, e_{\mathcal{C}}\rangle^i_\bullet) \text{ and } h'(\langle o_{\mathcal{C}}, e_{\mathcal{C}}\rangle^i_\bullet) \text{ are homomorphims} \Longrightarrow \left(\begin{array}{c} h(\langle o_{\mathcal{C}}, e_{\mathcal{C}}\rangle^0_\bullet) = h(c) \\ = \\ h'(\langle o_{\mathcal{C}}, e_{\mathcal{C}}\rangle^0_\bullet) = h'(c) \\ = \\ X \end{array}\right)$$

$$\left(\begin{array}{c} \langle h(\langle o_{\mathcal{C}}, e_{\mathcal{C}}\rangle^{i\geqslant 1}_\bullet), h'(\langle o_{\mathcal{C}}, e_{\mathcal{C}}\rangle^{i\geqslant 1}_\bullet)\rangle \in \sim \\ \text{and} \\ h(\langle o_{\mathcal{C}}, e_{\mathcal{C}}\rangle^0_\bullet) = h'(\langle o_{\mathcal{C}}, e_{\mathcal{C}}\rangle^0_\bullet) \end{array}\right) \Longrightarrow \sim \text{ is a bisimulation}$$

Note that, notation $\Longrightarrow$ stands for a logical implication. For each $\langle o_{\mathcal{C}}, e_{\mathcal{C}}\rangle^i_\bullet$ in $\odot^i(\mathcal{C})$, if $\langle h(\langle o_{\mathcal{C}}, e_{\mathcal{C}}\rangle^i_\bullet), h'(\langle o_{\mathcal{C}}, e_{\mathcal{C}}\rangle^i_\bullet)\rangle \in \sim$ we can write $h(\langle o_{\mathcal{C}}, e_{\mathcal{C}}\rangle^i_\bullet) \sim h'(\langle o_{\mathcal{C}}, e_{\mathcal{C}}\rangle^i_\bullet)$. Thus, by the coinductive proof principle in (47) then $h = h'$.

For each $c \in \mathcal{C}$, we define a CSP process $CSPprocess(c)$ to be an infinite sequence

$$CSPprocess(c) \stackrel{def}{=} (X, g_1, g_2, \ldots) \tag{48}$$

thus, obviously, $CSPprocess(c)$ becomes a process. In other words, it is an element of $D\mathcal{C}_{(T\times\mathbf{2})}$. This means that $CSPprocess(c)$ is constructed as the process starting with configuration $c \in \mathcal{C}$. Generally, this provides a mapping $CSPprocess : \mathcal{C} \longrightarrow D\mathcal{C}_{(T\times\mathbf{2})}$. This mapping forms a unique $\odot(\_)$-homomorphism $CSPprocess : \langle \mathcal{C}, \langle o_{\mathcal{C}}, e_{\mathcal{C}} \rangle_{\bullet} \rangle \longrightarrow \langle D\mathcal{C}_{(T\times\mathbf{2})}, next_{(T\times\mathbf{2})} \rangle$.

6 Future Work

The reconfigurations we have studied here are specified by determining their behaviors. This is done not only by an external input specification, but also by the internal context in the modular system, to which there is no direct access in general. Such reconfigurations are inherently dynamic and have observable behaviors, but their internal configurations remain hidden and therefore have to be identified if not distinguishable by observation. Our CSP-based approach of behaviors throughout this paper has aimed at developing a foundation of aspect-oriented modular reconfigurable computing that is characterized by:

- *Configuration space*: In this space, configurations evolve and persist by adapting to improve suitability over time;
- *Interaction*: This specifies the exchangeability with the environment during the reconfiguration process;
- *Aspect/Component-Oriented model*: In this model, configware is oriented towards "aspect" development and flowware is oriented towards a "component" approach. Aspect-Oriented Configware relies on "separation of concerns" in all phases of the configware development lifecycle, while Component-Oriented Flowware greatly depends on prefabricated "components" for building the data processing needs of flowware. Incorporating both the emerging programming approaches based on aspect [17] and component [3] oriented paradigms could unify the two specialized concepts of "modularity" and "separation of concerns" in the model (see Figure 5).

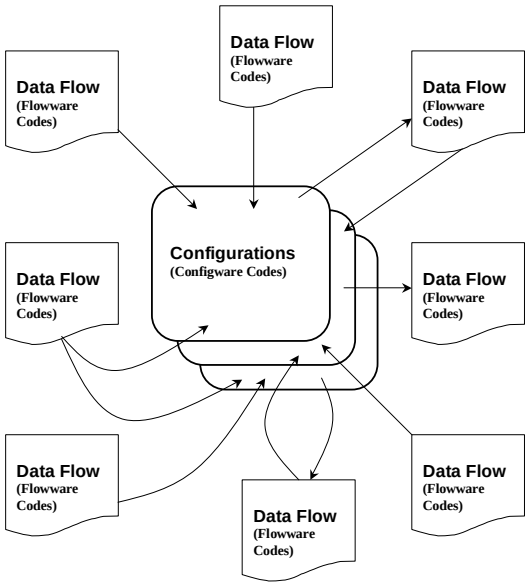


Figure 5: A diagrammatic illustration of Aspect/Component-Oriented reconfigurable computing

7 Summary

As an approach to CSP-based behaviors of aspect-oriented modular reconfigurable computing (AOMRC), using categorical and coalgebraic languages we have firstly introduced the notion of AOMRC and developed a coalgebraic model $\langle \mathcal{C}, \langle o_{\mathcal{C}}, e_{\mathcal{C}} \rangle_{\bullet} \rangle$ of AOMRC. Secondly, a Hoare model $\langle D\mathcal{C}_{(T\times\mathbf{2})}, next_{(T\times\mathbf{2})} \rangle$ of deterministic reconfiguration processes $((T \times \mathbf{2}), traces(P))$ has been constructed by developing coalgebraic model of a CSP process, which is proven to be a final coalgebra in the category $\mathbf{CoAlg}(\odot(\_))$ of $\odot(\_)$-coalgebras. Thus, we have established a homomorphic relation between $\langle \mathcal{C}, \langle o_{\mathcal{C}}, e_{\mathcal{C}} \rangle_{\bullet} \rangle$ and $\langle D\mathcal{C}_{(T\times\mathbf{2})}, next_{(T\times\mathbf{2})} \rangle$ as a unique homomorphism of coalgebras in $\mathbf{CoAlg}(\odot(\_))$. In such a way, the approach results in the behavioral equivalence between $\langle \mathcal{C}, \langle o_{\mathcal{C}}, e_{\mathcal{C}} \rangle_{\bullet} \rangle$ and $\langle D\mathcal{C}_{(T\times\mathbf{2})}, next_{(T\times\mathbf{2})} \rangle$. This behavioral equivalence means that the separation between the concepts of process in CSP and configuration in AOMRC is completely blurred. In other words, behaviors of any AOMRC configuration change being performed by a transformations-based aspect are not distinct from deterministic processes in Hoare's CSP.

Acknowledgements: Thank to the anonymous reviewers for their helpful comments and valuable suggestions which have contributed to the final preparation of the paper.

References

[1] J. Adamek, H. Herrlich, and G. Strecker. *Abstract and Concrete Categories*. John Wiley and Sons, 1990.

[2] A. Asperti and G. Longo. *Categories, Types and Structures*. M.I.T. Press, 1991.

[3] L.S. Barbosa and Z. Liu, editors. *Proceedings of the 2nd International Workshop on Formal Aspects of Component Software (FACS)*, volume 160 (August 2006) of *ENTCS*, Macao, 24–25 October 2005. UNU/IIST, Elsevier Science Publishers Ltd.

[4] G. M. Bergman. *An Invitation to General Algebra and Universal Constructions*. Henry Helson, 15 the Crescent, Berkeley CA 94708, US, 1998.

[5] A. DeHon. DPGA Utilization and Application. In *Proceedings of 4th International Symposium on Field Programmable Gate Arrays (FPGA'96)*, pages 115–121, Monterey, CA, US, 11–13 February 1996. ACM.

[6] A. DeHon and J. Wawrzynek. Reconfigurable Computing: What, Why, and Implications for Design Automation. In *Proceedings of 36th Design Automation Conference (DAC)*, pages 610–615, New Orleans, LA, USA, 21–25 June 1999. ACM.

[7] R. Hartenstein. The Microprocessor is No Longer General Purpose: Why Future Reconfigurable Platforms Will Win. In *Proceedings of 2nd Annual International Conference on Innovative Systems in Silicon*, pages 2–12, Austin, TX, USA, 8–10 October 1997. IEEE.

[8] R. Hartenstein. A Decade of Reconfigurable Computing: A Visionary Retrospective. In *Proceedings of Design, Automation, and Test in Europe Conference (DATE)*, pages 642–649, Munich, Germany, 13–16 March 2001. IEEE.

[9] R. Hartenstein. Coarse Grain Reconfigurable Architectures. In *Proceedings of Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 564–569, Yokohama, Japan, 30 January – 2 February 2001. IEEE.

[10] R. Hartenstein. Reconfigurable Computing: A New Business Model and its Impact on SoC Design. In *Proceedings of Euromicro Symposium on Digital Systems, Design (DSD)*, pages 103–110, Warzaw, Poland, 4–6 September 2001. IEEE.

[11] R. Hartenstein. Trends in Reconfigurable Logic and Reconfigurable Computing. In *Proceedings of 9th International Conference on Electronics, Circuits and Systems (ICECS)*, volume 2, pages 801–808, Dubrovnik, Croatia, 15–18 September 2002. IEEE.

[12] R. Hartenstein. Are We Really Ready for the Breakthrough? [morphware]. In *Proceedings of International Parallel and Distributed Processing Symposium (IPDPS)*, Nice, France, 22–26 April 2003. IEEE. CD-ROM.

[13] R. Hartenstein. Morphware and configware. In Albert Y. Zomaya, editor, *Handbook of Nature-Inspired and Innovative Computing. Integrating Classical Models with Emerging Technologies*. Springer, 2005. 780 pages.

[14] M. Herz, R. Hartenstein, M. Miranda, E. Brockmeyer, and F. Catthoor. Memory Addressing Organization for Stream-based Reconfigurable Computing. In *Proceedings of 9th International Conference on Electronics, Circuits and Systems (ICECS)*, volume 2, pages 813–817, Dubrovnik, Croatia, 15–18 September 2002. IEEE.

[15] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice Hall, 2004. This book has been updated by Jim Davies at the Oxford University Computing Laboratory.

[16] B. Jacobs and J. Rutten. A Tutorial on (Co)Algebras and (Co)Induction. *Bulletin of EATCS*, 62:222–259, 1997.

[17] G. Kiczales, J. Lamping, A. Menhdhekar, C. Maeda, C. Lopes, J.M. Loingtier, and J. Irwin. Aspect-Oriented Programming. In M. Akşit and S. Matsuoka, editors, *Proceedings of 11th European Conference on*

Object-Oriented Programming (ECOOP), volume 1241 of *Lecture Notes in Computer Science*, pages 220–242, Jyväskylä, Finland, 10 June 1997. Springer-Verlag. The paper originating AOP.

[18] M. Levine. Categorical Algebra. In G. Benkart, T.S. Ratiu, H.A. Masur, and M. Renardy, editors, *Mixed Motives*, volume 57 of *Mathematical Surveys and Monographs*, chapter I, II, II of Part II, pages 373–499. American Mathematical Society, USA, 1998.

[19] P. Lysaght. Aspects of Dynamically Reconfigurable Logic. In *Proceedings of Colloquium on Reconfigurable Systems*, volume 61, pages 1/1–1/5, Glasgow, Scotland, UK, 10 March 1999. IEE.

[20] J.J.M.M. Rutten. Automata and Coinduction (an Exercise in Coalgebra). In *Proceedings of 9th International Conference on Concurrency Theory (CONCUR)*, volume 1466 of *Lecture Notes in Computer Science*, pages 194–218, Nice, France, 8–11 September 1998. Springer-Verlag.

[21] J.J.M.M. Rutten. Universal Coalgebra: A Theory of Systems. *Theoretical Computer Science*, 249(1):3–80, 17 October 2000.

[22] P.C. Vinh. *Formal Aspects of Dynamic Reconfigurability in Reconfigurable Computing Systems*. PhD thesis, London South Bank University, 103 Borough Road, London SE1 0AA, UK, 4 May 2006.

[23] P.C. Vinh and J.P. Bowen. Formalising Configuration Relocation Behaviours for Reconfigurable Computing. In *Proceedings of SFP Workshop, FDL'02: Forum on Specification & Design Languages*, Marseille, France, 24–27 September 2002. CD-ROM.

[24] P.C. Vinh and J.P. Bowen. An Algorithmic Approach by Heuristics to Dynamical Reconfiguration of Logic Resources on Reconfigurable FPGAs. In *Proceedings of 12th International Symposium on Field Programmable Gate Arrays*, page 254, Monterey, CA, USA, 22–24 February 2004. ACM/SIGDA.

[25] P.C. Vinh and J.P. Bowen. A Provable Algorithm for Reconfiguration in Embedded Reconfigurable Computing. In M.G. Hinchey, editor, *Proceedings of 29th Annual IEEE/NASA Software Engineering Workshop (SEW)*, pages 245–252, Greenbelt, MD, USA, 6–7 April 2005. IEEE Computer Society Press.

[26] P.C. Vinh and J.P. Bowen. Continuity Aspects of Embedded Reconfigurable Computing. *Innovations in Systems and Software Engineering: A NASA journal*, 1(1):41–53, April 2005. Springer.

[27] P.C. Vinh and J.P. Bowen. POM Based Semantics of RTL and a Validation Method for the Synthesis Results in Dynamically Reconfigurable Computing Systems. In J. Rozenblit, T. O'Neill, and J. Peng, editors, *Proceedings of 12th Annual International Conference and Workshop on the Engineering of Computer Based Systems (ECBS)*, pages 247–254, Greenbelt, MD, USA, 4–5 April 2005. IEEE Computer Society Press.

[28] P.C. Vinh and J.P. Bowen. A Formal Approach to Aspect-Oriented Modular Reconfigurable Computing. In *Proceedings of 1st IEEE & IFIP International Symposium on Theoretical Aspects of Software Engineering (TASE)*, pages 369–378, Shanghai, China, 6–8 June 2007. IEEE Computer Society Press.