Coinductively Combinational Context-Awareness

Phan Cong Vinh(✉) and Nguyen Kim Quoc

Faculty of IT, Nguyen Tat Thanh University (NTTU), 300A Nguyen Tat Thanh St., Ward 13, District 4, HCM City, Vietnam
{pcvinh,nkquoc}@ntt.edu.vn

Abstract. This paper presents the notion of coinductively combinational context-awareness from practical perspective of P2P networks. Through the combinational features of context-awareness, we use the stream calculus and coinduction to discover the solution for arrangements of actions of context-aware systems in a uniform way.

Keywords: Automata · Coinduction · Context-awareness · Context-aware systems · Stream calculus

1 Introduction

One of the combinatorial features of context-awareness relates to the regularly repeated arrangements of the actions satisfying specified rules [12]. In fact, if a specified arrangement of actions is possible, there may be several ways of achieving it. If so, we want to count their number or to classify them into types. This is an important point when each action can be mapped into a suitable type of processing units for performing context-awareness. Moreover, counting hardware resources (i.e., processing units) can be also evaluated and, as a result, finding an allocation way of processing units becomes available to react upon a specific arrangement of the actions.

Although existence problems have been considered extensively in combinatorics, classification problems have been found to be more difficult [1,2]. Regarding the arrangements of actions, however, if the existence problems for a specified arrangement of actions can be tackled by a reasonable method, it is possible to count the number of ways of achieving the arrangement of actions. When approaching any solution of the combinatorial context-aware problems, we see that there is a differentiation between listing all arrangements of actions and determining their number. When the arrangements of actions are listed, they can be mapped one-to-one from a set of actions onto the set of natural numbers $\mathbb{N}$. By contrast, when the number of arrangements of actions become so large that a complete listing becomes impossible, the techniques for determining the number of arrangements of actions become more important. In general, the arrangement of actions is connected with the classification of discrete structures.

P.C. Vinh et al. (Eds.): ICCASA 2013, LNICST 128, pp. 238–249, 2014.
DOI: 10.1007/978-3-319-05939-6_24, © Springer International Publishing Switzerland 2014

One of the traditional techniques for solving these problems is mathematical induction [1,2]. In this paper, a dual method called *coinduction* is used to discover the solution for arrangements of actions in a uniform way:

- Actions are classified by an infinite weighted automaton.
- The automaton is reduced by the quantitative notion of stream bisimulation.
- A reduced automaton is used to compute a rule representing the stream determining the number of arrangements of actions.

The rest of this paper is organized as follows. Section 2 covers related work. Section 3 considers context-awareness from practical perspective. Section 4 focuses on the arrangements of actions of context-aware systems using coinduction. A short conclusion is given in Sect. 5.

2 Related Work

Rutten [3,4] has developed the notion of stream calculus to build a playground for the use of coinduction definition and proof principles. The important elements of stream calculus are the useful and powerful stream operations. Some basic stream operations such as *sum* ($+$), *convolution product* ($\times$) and *inverse* ($^{-1}$), which are used in this paper. Further details related to stream calculus and other operations can be referred to the standard text in [3]. The notion of weighted stream automata is also introduced by Rutten in [3,5], in which the transition diagrams are graphically represented for the successive derivatives of streams. In our previous research results, the stream calculus and weighted stream automata are widely applied in reconfigurable computing [7] and context-aware computing [6,8–12].

3 Context-Awareness from Practical Perspective

A network, which consists of the set of peers (considered as nodes) together with morphisms $\_ \parallel \_$ in the set of parallel compositions (considered as edges), generates P2P structure [8]. The P2P structure is dynamic in nature because peers can be dynamically added to or dropped from the network. For such every action, *context-awareness* for the P2P structure occurs.

Let PEER be the set of peers and $\mathsf{SYS} = \{\|_{i \in \mathbb{N}_0} a_i \text{ with } a_i \in \mathsf{PEER}\}$ be the set of parallel compositions on the P2P network.

Let $T = \{add, drop\}$ be the set of actions making a P2P structure on the network change, in which *add* and *drop* are defined as follows:

add is a binary operation

$$add : \mathsf{SYS} \times \mathsf{PEER} \longrightarrow \mathsf{SYS} \qquad (1)$$

(sometimes specified as $\mathsf{SYS} \xrightarrow{add(\mathsf{PEER})} \mathsf{SYS}$ or $add(\mathsf{PEER}) : \mathsf{SYS} \longrightarrow \mathsf{SYS}$)

obeying the following axioms: For all $i \in \mathbb{N}_0$,

$$add(\|_i\ a_i, b) = \begin{cases} (\|_{1\leqslant i\leqslant n}\ a_i) \parallel b & \text{for } i \geqslant 1 \\ (\|_0) \parallel b = skip \parallel b = b & \text{when } i = 0 \end{cases} \tag{2}$$

or, also written as

$$\begin{cases} \|_{1\leqslant i\leqslant n}\ a_i \xrightarrow{add(b)} (\|_{1\leqslant i\leqslant n}\ a_i) \parallel b & \text{for } i \geqslant 1 \\ \|_0 \xrightarrow{add(b)} (\|_0) \parallel b = skip \parallel b = b & \text{when } i = 0 \end{cases}$$

or

$$\begin{cases} add(b)\ :\ \|_{1\leqslant i\leqslant n}\ a_i \longrightarrow (\|_{1\leqslant i\leqslant n}\ a_i) \parallel b & \text{for } i \geqslant 1 \\ add(b)\ :\ \|_0 \longrightarrow (\|_0) \parallel b = skip \parallel b = b & \text{when } i = 0 \end{cases}$$

Example:

$$\begin{aligned} add(\|_0, a) &= a \\ add(a, b) &= a \parallel b \\ add(a \parallel b, c) &= a \parallel b \parallel c \end{aligned}$$

drop is also a binary operation

$$drop : \mathsf{SYS} \times \mathsf{PEER} \longrightarrow \mathsf{SYS} \tag{3}$$

(sometimes specified as $\mathsf{SYS} \xrightarrow{drop(\mathsf{PEER})} \mathsf{SYS}$ or $drop(\mathsf{PEER}) : \mathsf{SYS} \longrightarrow \mathsf{SYS}$)

obeying the following axioms: For all $i \in \mathbb{N}_0$,

$$drop(\|_i\ a_i, b) = \begin{cases} \|_{1\leqslant i\leqslant (n-1)}\ a_i & \text{when there exists } a_i = b \\ \|_{1\leqslant i\leqslant n}\ a_i & \text{for all } a_i \neq b \end{cases} \tag{4}$$

or, also written as

$$\begin{cases} \|_{1\leqslant i\leqslant n}\ a_i \xrightarrow{drop(b)} \|_{1\leqslant i\leqslant (n-1)}\ a_i & \text{when there exists } a_i = b \\ \|_{1\leqslant i\leqslant n}\ a_i \xrightarrow{drop(b)} \|_{1\leqslant i\leqslant n}\ a_i & \text{for all } a_i \neq b \end{cases}$$

or

$$\begin{cases} drop(b)\ :\ \|_{1\leqslant i\leqslant n}\ a_i \longrightarrow \|_{1\leqslant i\leqslant (n-1)}\ a_i & \text{when there exists } a_i = b \\ drop(b)\ :\ \|_{1\leqslant i\leqslant n}\ a_i \longrightarrow \|_{1\leqslant i\leqslant n}\ a_i & \text{for all } a_i \neq b \end{cases}$$

It follows that $drop(\|_0, b) = \|_0 = skip$.

Example:

$$\begin{aligned} drop(a, a) &= \|_0 \\ drop(a \parallel b \parallel c, b) &= a \parallel c \\ drop(a \parallel b \parallel c, d) &= a \parallel b \parallel c \end{aligned}$$

A context-awareness process is completely defined when actions *add* and *drop* are executed on a P2P network as illustrated in automaton (5):

$$\begin{tikzcd}
(\|_0) \arrow[loop right, "drop"] \arrow[r, bend left, "add"] & (a_1) \arrow[loop right, "\varepsilon"] \arrow[l, bend left, "drop"] \arrow[r, bend left, "add"] & (a_1 \parallel a_2) \arrow[loop right, "\varepsilon"] \arrow[l, bend left, "drop"] \arrow[r, bend left, "add"] & (a_1 \parallel a_2 \parallel a_3) \arrow[loop right, "\varepsilon"] \arrow[l, bend left, "drop"] \arrow[r, bend left, "add"] & \cdots \arrow[l, bend left, "drop"]
\end{tikzcd} \tag{5}$$

In consideration of P2P networks, context-awareness are known as *homomorphisms* from a P2P network to another P2P network to preserve the P2P structure. In other words, context-awareness is a map from a set of parallel compositions to another set of parallel compositions of the same type that preserves all the P2P structures.

Definition 1 (Context-Awareness). *Let $T = \{add, drop\}$ be a set of actions. A context-awareness with set of actions T is a pair $\langle \mathsf{SYS}, \langle o_{\mathsf{SYS}}, e_{\mathsf{SYS}} \rangle \rangle$ consisting of*

- *a set SYS of* P2P networks*,*
- *an* output function $o_{\mathsf{SYS}} : \ \mathsf{SYS} \longrightarrow (T \longrightarrow \mathit{2})$*, and*
- *an* evolution function $e_{\mathsf{SYS}} : \ \mathsf{SYS} \longrightarrow (T \longrightarrow \mathsf{SYS})$*.*

where

- $\mathbf{2} = \{0, 1\}$,
- o_{SYS} assigns, to a network c, a function $o_{\mathsf{SYS}}(c) : \ T \longrightarrow \mathbf{2}$, which specifies the value $o_{\mathsf{SYS}}(c)(t)$ that is reached after an action t has been executed. In other words,
$$o_{\mathsf{SYS}}(c)(t) = \begin{cases} 1 \text{ when } t \text{ becomes fully available, or} \\ 0 \text{ otherwise} \end{cases}$$
- Similarly, e_{SYS} assigns, to a network c, a function $e_{\mathsf{SYS}}(c) : \ T \longrightarrow \mathsf{SYS}$, which specifies the network $e_{\mathsf{SYS}}(c)(t)$ that is reached after an action t has been executed. Sometimes $c \xrightarrow{t} c'$ is used to denote $e_{\mathsf{SYS}}(c)(t) = c'$.

Generally, both the network space SYS and the set T of actions may be infinite. If both SYS and T are finite, then we have a finite context-awareness, otherwise we have an infinite context-awareness.

4 Coinductively Combinational Context-Awareness

Regarding regularly repeated arrangements of the actions during the changing of a context, in this paper we concentrate to deal with counting the actions, which are introduced in Sect. 3. We apply the *method of coinductive counting* [5] for problem solving. There are three phases in this very general and flexible procedure of coinductive counting, which enables the number of our combinational actions to be counted in a uniform way:

- The number of actions are counted to classify in an infinite weighted automaton that plays a crucial role as the basis for a representation of the infinite stream.
- The automaton is reduced by the notion of stream bisimulation. This is heart of the method, using bisimulation relations to identify the states for reducing infinite weighted automaton of the previous phase to much better structured (and often finite) automaton.
- A reduced automaton is used to compute an expression in closed form representing the stream behavior of all arrangements of actions as a behavioral function.

Consider the set of basic actions $T = \{\mathbf{A}, \mathbf{D}\}$ consisting of two actions, *add* ($\mathbf{A}$) and *drop* ($\mathbf{D}$), which can be combined to create other actions. The question arises, for any natural number $k \geqslant 0$, when applying action $\mathbf{A}$ or $\mathbf{D}$ on a system k times, what is the count s_k of the k-length sequence of $\mathbf{A}$s or $\mathbf{D}$s? In other words, what is the stream of all counts $\sigma = (s_0, s_1, s_2, ...)$?

Figure 1 is an automaton informally describing all possible combinational actions. The states are numbered to identify according to which action occurs. In other words, 1 when $\mathbf{D}$ occurs and 0 otherwise. On the other hand, let a sequence of actions, called w, be an arrangement of $\mathbf{A}$s or $\mathbf{D}$s following in order and the set of all action sequences denote T^*. The automaton $\langle Q, \langle o, t\rangle\rangle$ is formally described by defining

- a state set $Q = \{w | w \in T^*\}$,
- an output function $o : Q \longrightarrow \mathbb{A}$ given by $o(w) = \begin{cases} 0 & \text{if } \ddagger(w,1) = \mathbf{A} \\ 1 & \text{otherwise} \end{cases}$, and
- a transition function $t : Q \longrightarrow \mathbb{R}^Q$ given by $t(v)(w) = 1$ denoted by $v \longrightarrow w$.

All states $w \in Q$ labeled by the sequences of $\mathbf{A}$ and $\mathbf{D}$ are output states and $\ddagger(w, i)$ is an output of the function $\ddagger : w \in Q \times i \in \mathbb{N} \longrightarrow \ddagger(w, i) \in T^*$ that outputs the last i actions contained in w. The stream $\sigma = (s_0, s_1, s_2, ...)$ of all counts is represented by the initial state ε, that is, $\sigma = (s_0, s_1, s_2, ...) = S(\varepsilon)$. After numbering we obtain the state numbers; the automaton can be simplified by identifying all state numbers in Fig. 1. In other words, the streams represented by the i-numbered states can be mapped into the streams represented by the states q_i of the simplified automaton in Fig. 2. Formally, this relation suggests the following statement.

Proposition 1. *The relation* $R \subseteq \mathbb{R}^\omega \times \mathbb{R}^\omega$ *is established by* $R = \{\langle S(\varepsilon), S(q_0)\rangle\} \cup \{\langle S(w), S(q_0)\rangle | w \in T^*, \ddagger(w,1) = \mathbf{A}\} \cup \{\langle S(w), S(q_1)\rangle | w \in T^*, \ddagger(w,1) = \mathbf{D}\}$ *is a bisimulation-up-to.*

Proof. It is straightforward to check that the streams $S(w)$ represented by the i-numbered states of the automaton in Fig. 1 are matched with the streams represented by the state q_i of the automaton in Fig. 2.

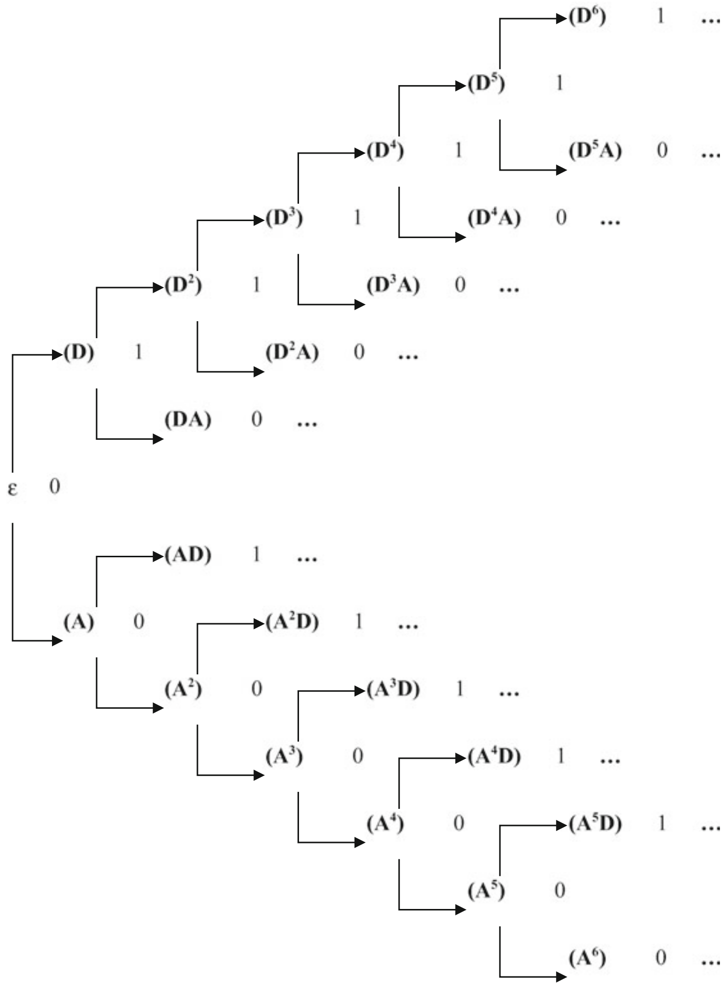

Fig. 1. Automaton based on actions of **A** and **D**

Thus,

Corollary 1.

$$\begin{array}{ll} S(\varepsilon) = S(q_0) & \textit{when } w = \varepsilon \\ S(w) = S(q_0) & \textit{when } \ddagger(w,1) = \mathbf{A} \\ S(w) = S(q_1) & \textit{when } \ddagger(w,1) = \mathbf{D} \end{array}$$

Proof. From the coinductive proof principle, named coinduction-up-to, it is easy to recognize that the relation R in Proposition 1 is a bisimulation-up-to. In addition, $\langle S(\varepsilon), S(q_0)\rangle \in R$, this yields $S(\varepsilon) = S(q_0)$ when $w = \varepsilon$. $\{\langle S(w), S(q_0)\rangle | w \in T^*, \ddagger(w,1) = \mathbf{A}\} \subset R$, this yields $S(w) = S(q_0)$ when $\ddagger(w,1) = \mathbf{A}$. $\{\langle S(w), S(q_1)\rangle | w \in T^*, \ddagger(w,1) = \mathbf{D}\} \subset R$, this yields $S(w) = S(q_1)$ when $\ddagger(w,1) = \mathbf{D}$.

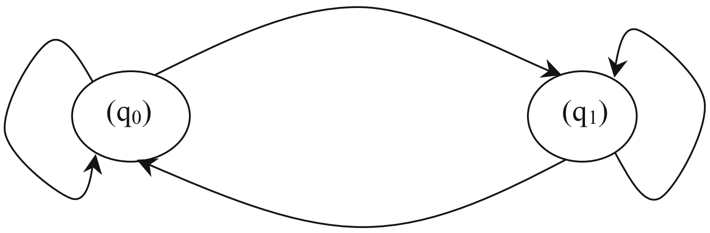

Fig. 2. Automaton of states $q_i,\ i = 0, 1$

We have the following equations for the behaviors of q_0 and q_1:

System of differential equations	Initial values
$S(q_0)' = S(q_1)' = S(q_0) + S(q_1)$	$S(q_0)(0) = S(q_1)(0) = 1$

To solve this systems of equations, following the fundamental theorem in [3] we obtain:

$$S(q_0) = 1 + XS(q_0)'$$

and

$$S(q_1) = 1 + XS(q_1)'$$

$\therefore$

$$S(q_0)' = 2 + 2XS(q_0)'$$

$\therefore$ [property of inverse operation]

$$S(q_0)' = \frac{2}{1-2X}$$

and the stream behaviors of $S(q_0)$ and $S(q_1)$ are computed by $S(q_0) = S(q_1) = \frac{1}{1-2X}$, which yield the solution streams of $S(q_0)$ and $S(q_1)$ by reasoning as follows:

$$1 - 2X = (1, -2, 0, 0, ...)$$

Let $\frac{1}{1-2X} = (a_0, a_1, a_2, ...)$

$\therefore$ [inverse operation]

$$\frac{1}{1-2X} \times (1 - 2X) = 1$$

$\therefore$ [product operation]

$$(a_0, a_1 - 2a_0, a_2 - 2a_1, a_3 - 2a_2, ...) = (1, 0, 0, 0, ...)$$

yielding

$$S(q_0) = S(q_1) = \frac{1}{1-2X} = (1, 2, 2^2, 2^3, 2^4, ..., 2^k, ...) \tag{6}$$

Another type of issue is, for instance, finding the count s_k of the k-length sequence of **A**s and **D**s ending with **DD**. Figure 3 is an automaton informally describing all possible combinational actions. All states labeled by the sequences of **A** and **D** ending with **DD** are output states and have no further transitions. On the other hand, the automaton $\langle Q, \langle o, t\rangle\rangle$ is a formal description defined by:

- a state set $Q = \{w | w \in T^*\}$,
- an output function $o : Q \longrightarrow \mathbb{R}$ given by $o(w) = \begin{cases} 1 & \text{if } \ddagger(w, 2) = \mathbf{DD} \\ 0 & \text{otherwise} \end{cases}$, and
- a transition function $t : Q \longrightarrow \mathbb{R}^Q$ given by $t(v)(w) = 1$ denoted by $v \longrightarrow w$.

The state numbers are used to identify the final **DD**s. 0 is numbered for the state when **A** occurs, 1 for the state when **D** occurs and 2 for the state including **DD** at the end. The stream $\sigma = (s_0, s_1, s_2, ...)$ of all counts is represented by the initial state ε, that is, $\sigma = S(\varepsilon)$. After numbering states as described above, the automaton can be simplified by identifying all state numbers in Fig. 3. In other words, the streams represented by the i-numbered states can be mapped into the streams represented by the states q_i of the automaton in Fig. 4. Formally, this relation suggests the following statement.

Proposition 2. *The relation $R \subseteq \mathbb{R}^\omega \times \mathbb{R}^\omega$ is established by $R = \{\langle S(\varepsilon), S(q_0)\rangle\} \cup \{\langle S(w), S(q_0)\rangle \mid w \in T^*,\ \ddagger(w, 1) = \mathbf{A}\} \cup \{\langle S(w), S(q_1)\rangle \mid w \in T^*,\ \ddagger(w, 1) = \mathbf{D}\} \cup \{\langle S(w), S(q_2)\rangle \mid w \in T^*,\ \ddagger(w, 2) = \mathbf{DD}\}$ is a bisimulation-up-to.*

Proof. The streams $S(w)$ represented by the i-numbered states of the automaton in Fig. 3 are matched with the streams represented by the state q_i of the automaton in Fig. 4.

Thus,

Corollary 2.

$$\begin{array}{ll} S(\varepsilon) = S(q_0) & \textit{when } w = \varepsilon \\ S(w) = S(q_0) & \textit{when } \ddagger(w, 1) = \mathbf{A} \\ S(w) = S(q_1) & \textit{when } \ddagger(w, 1) = \mathbf{D} \\ S(w) = S(q_2) & \textit{when } \ddagger(w, 2) = \mathbf{DD} \end{array}$$

Proof. It happens as a result of coinduction-up-to. In fact, it is easy to recognize that the relation R in Proposition 2 is a bisimulation-up-to. In addition, $\langle S(\varepsilon), S(q_0)\rangle \in R$, this yields $S(\varepsilon) = S(q_0)$ when $w = \varepsilon$. $\{\langle S(w), S(q_0)\rangle \mid w \in T^*,\ \ddagger(w, 1) = \mathbf{A}\} \subset R$, this yields $S(w) = S(q_0)$ when $\ddagger(w, 1) = \mathbf{A}$. $\{\langle S(w), S(q_1)\rangle \mid w \in T^*,\ \ddagger(w, 1) = \mathbf{D}\} \subset R$, this yields $S(w) = S(q_1)$ when $\ddagger(w, 1) = \mathbf{D}$. $\{\langle S(w), S(q_2)\rangle \mid w \in T^*,\ \ddagger(w, 2) = \mathbf{DD}\} \subset R$, this yields $S(w) = S(q_2)$ when $\ddagger(w, 2) = \mathbf{DD}$.

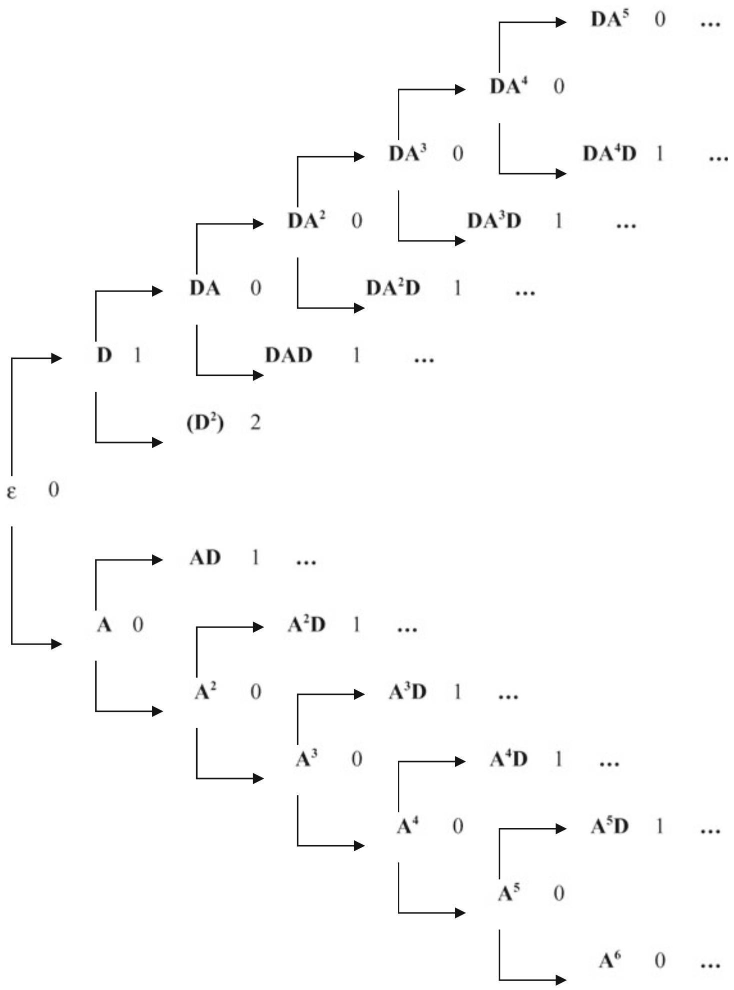

Fig. 3. Automaton based on actions of **A** and **D** ending at **DD**

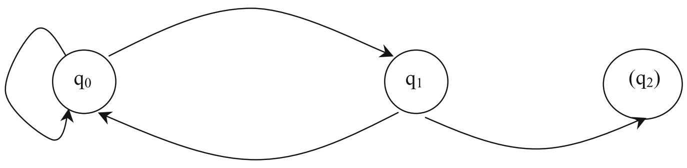

Fig. 4. Automaton of states $q_i,\ i = 0, 1, 2$

We have the following equations for the behaviors of q_0, q_1 and q_2:

System of differential equations	Initial values
$S(q_0)' = S(q_0) + S(q_1)$	$S(q_0)(0) = 0$
$S(q_1)' = S(q_0) + S(q_2)$	$S(q_1)(0) = 0$
$S(q_2)(0)' = 0$	$S(q_2)(0) = 1$

To solve this systems of equations, following the fundamental theorem in [3] we obtain:

$$\begin{aligned} S(q_0) &= XS(q_0)' \\ S(q_1) &= XS(q_1)' \\ S(q_2) &= 1 \end{aligned}$$

$\therefore$ [properties of addition and product operations]

$$S(q_0)' = XS(q_0)' + XS(q_1)'$$

$\therefore$

$$S(q_1)' = XS(q_0)' + 1$$

$\therefore$

$$S(q_0)' = XS(q_0)' + XS(q_0)' + 1$$

$\therefore$ [property of inverse operation]

$$S(q_0)' = \frac{1}{1-2X}$$

$\therefore$

$$S(q_1)' = \frac{1-X}{1-2X}$$

and the stream behaviors of $S(q_0)$, $S(q_1)$ and $S(q_2)$ are computed by

$$\begin{aligned} S(q_0) &= \frac{X}{1-2X} \\ S(q_1) &= \frac{X-X^2}{1-2X} \\ S(q_2) &= 1 \end{aligned}$$

which yield the solution stream of $S(q_2) = (1, 0, 0, 0, ...)$. In addition, the solution streams of $S(q_0)$ and $S(q_1)$ are calculated by reasoning as follows:
$\therefore$ [from the result in (6) and property of the constant stream X]

$$S(q_0) = \frac{X}{1-2X} = (0, 1, 2, 2^2, 2^3, ..., 2^k, ...) \tag{7}$$

$\therefore$ [from the result in (7) and property of the constant stream X]

$$\begin{aligned} S(q_1) = \frac{X - X^2}{1 - 2X} = \frac{X}{1 - 2X} - \frac{X^2}{1 - 2X} \\ = (0, 1, 2, 2^2, 2^3, 2^4, ..., 2^k, ...) - (0, 0, 1, 2, 2^2, 2^3, ..., 2^k, ...) \\ = (0, 1, 1, 2, 2^2, 2^3, ..., 2^k, ...) \end{aligned} \tag{8}$$

5 Conclusions

In this paper, from the combinational features of context-awareness, we use the stream calculus and coinduction to evaluate the counting of actions in a uniform way consisting of three steps:

- Actions are counted to classify in an infinite weighted automaton that is the basis for a representation of the infinite stream.
- Using stream bisimulation relations, the infinite weighted automaton is reduced to much better structured (often finite) automaton.
- A reduced automaton is used to compute a closed form expression representing the stream behavior (or behavioral function) of all arrangements of actions.

Acknowledgements. Thank you to NTTU (Nguyen Tat Thanh University, Vietnam) for the constant support of our work which culminated in the publication of this paper. As always, we are deeply indebted to the anonymous reviewers for their helpful comments and valuable suggestions which have contributed to the final preparation of the paper.

References

1. Brualdi, R.A.: Introductory Combinatorics, 4th edn. Prentice Hall, Upper Saddle River (2004). 6 April 2004
2. Cameron, P.J.: Combinatorics: Topics, Techniques, Algorithms. Cambridge University Press, New York (2001)
3. Rutten, J.J.M.M.: Elements of stream calculus (an extensive exercise in coinduction). In: Brooks, S., Mislove, M. (eds.) Proceedings of the MFPS 2001: 7th Conference on the Mathematical Foundations of Programming Semantics. Electronic Notes in Theoretical Computer Science, vol. 45, pp. 1–66. Elsevier Science Publishers, Amsterdam (2001)
4. Rutten, J.J.M.M.: An application of stream calculus to signal flow graphs. In: de Boer, F.S., Bonsangue, M.M., Graf, S., de Roever, W.-P. (eds.) FMCO 2003. LNCS, vol. 3188, pp. 276–291. Springer, Heidelberg (2004)
5. Rutten, J.J.M.M.: Coinductive counting with weighted automata. J. Autom. Lang. Comb. **8**(2), 319–352 (2003)
6. Vinh, P.C.: Formal aspects of self-* in autonomic networked computing systems. In: Zhang, Y., Yang, L.T., Denko, M.K. (eds.) Autonomic Computing and Networking, pp. 381–410. Springer, New York (2009)
7. Vinh, P.C.: Dynamic Reconfigurability in Reconfigurable Computing Systems: Formal Aspects of Computing, 1st edn., 236 pp. VDM Verlag, Saarbrucken (2009)

8. Vinh, P.C.: Formal specification and verification of self-configuring P2P networking: a case study in mobile environments. Formal and Practical Aspects of Autonomic Computing and Networking: Specification, Development, and Verification, 1st edn, pp. 170–188. IGI Global, Hershey (2011)
9. Phan, C.-V.: Data intensive distributed computing in data aware self-organizing networks. In: Gavrilova, M.L., Tan, C.J.K., Phan, C.-V. (eds.) Transactions on Computational Science XV. LNCS, vol. 7050, pp. 74–107. Springer, Heidelberg (2012)
10. Behan, M., Krejcar, O.: Concept of the personal devices content management using modular architecture and evaluation based design. In: Vinh, P.C., Hung, N.M., Tung, N.T., Suzuki, J. (eds.) ICCASA 2012. LNICST, vol. 109, pp. 151–159. Springer, Heidelberg (2013)
11. Vinh, P.C., Tung, N.T.: Coalgebraic aspects of context-awareness. Mob. Netw. Appl., August 2012. doi:10.1007/s11036-012-0404-0
12. Vinh, P.C., Tung, N.T., Van Phuc, N., Thanh, N.H.: Functional stream derivatives of context-awareness on P2P networks. In: Vinh, P.C., Hung, N.M., Tung, N.T., Suzuki, J. (eds.) ICCASA 2012. LNICST, vol. 109, pp. 160–167. Springer, Heidelberg (2013)