Theoretical Informatics and Applications
Theoret. Informatics Appl. **33** (1999) 393–400

A NOTE ON COINDUCTION AND WEAK BISIMILARITY FOR WHILE PROGRAMS

J.J.M.M. RUTTEN[1]

Abstract. An illustration of coinduction in terms of a notion of weak bisimilarity is presented. First, an operational semantics $\mathcal{O}$ for while programs is defined in terms of a final automaton. It identifies any two programs that are weakly bisimilar, and induces in a canonical manner a compositional model $\mathcal{D}$. Next $\mathcal{O} = \mathcal{D}$ is proved by coinduction.

AMS Subject Classification. 68Q10, 68Q55.

1. AUTOMATA

Let O be a (possibly infinite) set of output symbols. An *automaton* with outputs in O is a pair $S = (S, \alpha)$ consisting of a set S of *states* and a *transition function* $\alpha : S \to O + S$. The transition function α specifies for a state s in S either an output o in O or a next state s' in S. The intuition is that in the first case, the computation is terminating, with observable output o; in the second case, the computation takes one step and will continue from the new state s'. We shall sometimes write $s \downarrow o$ if $\alpha(s) = o \in O$, and $s \longrightarrow_S s'$ if $\alpha(s) = s' \in S$. If S is clear from the context, we shall simply write $s \longrightarrow s'$.

This type of automaton is sometimes referred to as *Elgot machine*, because of the prominent role similar such structures play in the work of Elgot (*cf.* [3]).

Keywords and phrases: Coalgebra, automaton, weak bisimulation, coinduction, while program.

[1] CWI, P.O. Box 94079, 1090 GB Amsterdam, The Netherlands; e-mail: janr@cwi.nl URL: www.cwi.nl/~janr

A *homomorphism* between automata $S = (S, \alpha)$ and $T = (T, \beta)$ is a function $f : S \to T$ for which the following diagram commutes:

$$\begin{tikzcd}
S \arrow[r, "f"] \arrow[d, "\alpha"'] & T \arrow[d, "\beta"] \\
O + S \arrow[r, "id_O + f"'] & O + T.
\end{tikzcd}$$

Here $id_O + f$ acts as the identity on O, and maps S to T by f. A function f is a homomorphism precisely when $s \longrightarrow_S s'$ implies $f(s) \longrightarrow_T f(s')$ and $s \downarrow o$ implies $f(s) \downarrow o$, for all s in S.

A *bisimulation* between two automata S and T is a relation $R \subseteq S \times T$ with, for all s in S and t in T: if $s\,R\,t$ then

1. if $s \longrightarrow_S s'$ then $t \longrightarrow_T t'$ and $s'\ R\ t'$;
2. if $s \downarrow o$ then $t \downarrow o$;
3. if $t \longrightarrow_T t'$ then $s \longrightarrow_S s'$ and $s'\ R\ t'$;
4. if $t \downarrow o$ then $s \downarrow o$.

A bisimulation between S and itself is called a bisimulation *on* S. Unions and (relational) compositions of bisimulations are bisimulations again. We write $s \sim s'$ whenever there exists a bisimulation R with sRs'. This relation $\sim$ is the union of all bisimulations and, therewith, the greatest bisimulation. The greatest bisimulation on one and the same automaton, again denoted by $\sim$, is called the *bisimilarity* relation. It is an equivalence relation.

Let $\mathcal{N} = \{0, 1, \ldots\}$ and let $C = (O \times \mathcal{N}) \cup \{\infty\}$. The set C can be supplied with a transition function $\gamma : C \to O + C$ by defining $\gamma(\langle o, 0\rangle) = o$, $\gamma(\langle o, n+1\rangle) = \langle o, n\rangle$, and $\gamma(\infty) = \infty$. The automaton $C = (C, \gamma)$ is of special interest because it is *final* in the sense that for any automaton $S = (S, \alpha)$, there exists a unique homomorphism $\alpha^\sharp : S \to C$:

$$\begin{tikzcd}
S \arrow[r, dashed, "\exists!\,\alpha^\sharp"] \arrow[d, "\alpha"'] & C \arrow[d, "\gamma"] \\
O + S \arrow[r, dashed, "id_O + \alpha^\sharp"'] & O + C,
\end{tikzcd}$$

defined, for s in S, by

$$\alpha^\sharp(s) = \begin{cases} \langle o, n\rangle & \text{if } s = s_0 \longrightarrow s_1 \longrightarrow \cdots \longrightarrow s_n \downarrow o \\ \infty & \text{if } s = s_0 \longrightarrow s_1 \longrightarrow s_2 \longrightarrow \cdots \end{cases}$$

If $\alpha^\sharp(s) = \infty$ then we say that (the computation starting in) s *diverges*. If s does not diverge we say it *converges*.

There is the following principle:

$$\text{COINDUCTION:} \quad \forall s, s' \in S,\ s \sim s' \iff \alpha^\sharp(s) = \alpha^\sharp(s'). \tag{1}$$

Coinduction can be used as a proof principle: in order to prove $\alpha^\sharp(s) = \alpha^\sharp(s')$, it is sufficient to establish the existence of a bisimulation relation R on S with $s\,R\,s'$.

2. While programs

Let Σ be an abstract set of *program states* and let the set *Prog* of *while programs*, be given by the following syntax:

$$P ::= \ \underline{a} \mid P;Q \mid \text{if } \underline{c} \text{ then } P \text{ else } Q \mid \text{while } \underline{c} \text{ do } P.$$

Here $\underline{a}$ is in *Act*, the set of *actions*, and $\underline{c}$ is in *Cond*, the set of *conditions*, with

$$Act = \{\underline{a} \mid a : \Sigma \rightharpoonup \Sigma\} \quad \text{and} \quad Cond = \{\underline{c} \mid c \subseteq \Sigma\},$$

where $\Sigma \rightharpoonup \Sigma$ is the set of all *partial* functions on Σ. Clearly, more concrete definitions can be given for either of these sets. Skip statements and assignments would be typical atomic actions, Boolean expressions could be used as a syntax for conditions, and program states are usually defined as functions from variables to values. We are not interested in such details here. Although not needed for a standard interpretation of while programs, atomic actions are allowed to be partial, which will be convenient later.

Next the behaviour of programs is defined by specifying, in the usual manner, a transition function on pairs $\langle P, \sigma\rangle$ of programs and program states, as follows:

$\langle \underline{a}, \sigma\rangle \downarrow a(\sigma)$, if $a(\sigma)$ is defined; $\langle \underline{a}, \sigma\rangle \longrightarrow \langle \underline{a}, \sigma\rangle$, otherwise;
$\langle P;Q, \sigma\rangle \longrightarrow \langle P';Q, \sigma'\rangle$, if $\langle P, \sigma\rangle \longrightarrow \langle P', \sigma'\rangle$;
$\langle P;Q, \sigma\rangle \longrightarrow \langle Q, \sigma'\rangle$, if $\langle P, \sigma\rangle \downarrow \sigma'$;
$\langle \text{if } \underline{c} \text{ then } P \text{ else } Q,\ \sigma\rangle \longrightarrow \langle P, \sigma\rangle$, if $\sigma \in c$;
$\langle \text{if } \underline{c} \text{ then } P \text{ else } Q,\ \sigma\rangle \longrightarrow \langle Q, \sigma\rangle$, if $\sigma \notin c$;
$\langle \text{while } \underline{c} \text{ do } P,\ \sigma\rangle \longrightarrow \langle P; (\text{while } \underline{c} \text{ do } P), \sigma\rangle$, if $\sigma \in c$;
$\langle \text{while } \underline{c} \text{ do } P,\ \sigma\rangle \downarrow \sigma$, if $\sigma \notin c$.

The above determines a transition function $\alpha : (Prog \times \Sigma) \to \Sigma + (Prog \times \Sigma)$. Taking $O = \Sigma$ in Section 1 then yields a function

$$\alpha^\sharp : (Prog \times \Sigma) \to (\Sigma \times \mathcal{N}) \cup \{\infty\},$$

which can be viewed as a first operational semantics for while programs. The function $\alpha^\sharp$ assigns to a pair $\langle P, \sigma\rangle$ either ∞, corresponding to the fact that the computation when started in $\langle P, \sigma\rangle$ is diverging, or $\alpha^\sharp$ yields a pair $\langle \sigma', n\rangle$, consisting of an end result σ' together with a natural number indicating the number of computation steps that were needed to obtain it.

Coinduction may now be applied to establish some familiar identities. Let us write $P \sim Q$ whenever $\langle P, \sigma\rangle \sim \langle Q, \sigma\rangle$, for all σ. Writing c' for the complement of $c \subseteq \Sigma$, we have, for instance,

$$\text{if } \underline{c} \text{ then } P \text{ else } Q \ \sim \ \text{if } \underline{c'} \text{ then } Q \text{ else } P,$$

since for any σ, the following relation obviously is a bisimulation:

$$\{(\langle \text{if } \underline{c} \text{ then } P \text{ else } Q,\ \sigma\rangle,\ (\langle \text{if } \underline{c'} \text{ then } Q \text{ else } P,\ \sigma\rangle)\} \cup \Delta,$$

where Δ is the identity relation on $Prog \times \Sigma$. It follows that, for any $\sigma \in \Sigma$,

$$\alpha^\sharp(\langle \text{if } \underline{c} \text{ then } P \text{ else } Q,\ \sigma\rangle) \;=\; \alpha^\sharp(\langle \text{if } \underline{c'} \text{ then } Q \text{ else } P,\ \sigma\rangle).$$

Because $\alpha^\sharp$ keeps track of the number of computation steps, it is clearly not very abstract. For instance, letting 1 be the identity on Σ (corresponding to a skip statement), P and $\underline{1}; P$ are generally not bisimilar (unless the program P will diverge for any σ). Consequently, the two programs will not be identified by the operational semantics $\alpha^\sharp$. A more abstract semantics is needed.

3. Weak bisimilarity

Recall from Section 1 that $C = (O \times \mathcal{N}) \cup \{\infty\}$. Let $p : C \to O$ map $\langle o, n\rangle$ to o and let p be undefined in ∞. Let $\alpha^\dagger = p \circ \alpha^\sharp$:

$$\begin{tikzcd}
S \arrow[r, "\alpha^\sharp"'] \arrow[d, "\alpha"'] \arrow[rr, bend left, "\alpha^\dagger"] & C \arrow[r, "p"'] \arrow[d, "\gamma"] & O \\
O+S \arrow[r, "id_O+\alpha^\sharp"'] & O+C \arrow[ur, "{[id_O, p]}"'] &
\end{tikzcd}$$

The partial function $\alpha^\dagger$ is a more abstract version of $\alpha^\sharp$ in that it no longer registers the number of computation steps. It can be characterized as follows. Let $\Longrightarrow_S$ denote the reflexive and transitive closure of the transition relation $\longrightarrow_S$ of an automaton $S = (S, \alpha)$. A *weak bisimulation* between automata S and T is a relation $R \subseteq S \times T$ with, for all s in S and t in T: if $s\,R\,t$ then

1. if $s \longrightarrow_S s'$ then $t \Longrightarrow_T t'$ and $s'\ R\ t'$;
2. if $s \downarrow o$ then $t \Longrightarrow_T t' \downarrow o$;
3. if $t \longrightarrow_T t'$ then $s \Longrightarrow_S s'$ and $s'\ R\ t'$;
4. if $t \downarrow o$ then $s \Longrightarrow_S s' \downarrow o$.

Unions and compositions of weak bisimulations are weak bisimulations again. Two elements s and s' in S are called *weakly bisimilar*, denoted by $s \approx s'$, if there exists a weak bisimulation R on S with $s\,R\ s'$.

Based on the notion of weak bisimulation, there is the following principle:

$$\approx\text{-COINDUCTION:}\quad \forall s, s' \in S,\ s \approx s' \iff \alpha^\dagger(s) = \alpha^\dagger(s'). \tag{2}$$

Again, the implication from left to right may serve as a proof principle: in order to prove $\alpha^\dagger(s) = \alpha^\dagger(s')$, it is sufficient to show that s and s' are weakly bisimilar.

Applying all this to while programs by taking $O = \Sigma$, we obtain a (partial) function $\alpha^\dagger : \mathit{Prog} \times \Sigma \to \Sigma$. Equivalently, there is a function

$$\mathcal{O} : \mathit{Prog} \to (\Sigma \rightharpoonup \Sigma), \quad \mathcal{O}(P)(\sigma) = \alpha^\dagger(\langle P, \sigma \rangle),$$

which is the classical operational semantics of while programs. Writing $P \approx Q$ whenever $\langle P, \sigma \rangle \approx \langle Q, \sigma \rangle$ for all σ, coinduction takes the following form:

$$\approx\text{-COINDUCTION:} \quad \forall P, Q \in \mathit{Prog}, \; P \approx Q \iff \mathcal{O}(P) = \mathcal{O}(Q).$$

Many semantic equalities are now immediate by coinduction from the fact that there exist a suitable weak bisimulation, such as for the following pair of programs:

$$\text{while } \underline{c} \text{ do } P \quad \approx \quad \text{if } \underline{c} \text{ then } (P; \text{while } \underline{c} \text{ do } P) \text{ else } \underline{1}.$$

Note that these statements are not bisimilar in the sense of Section 2.

4. A COMPOSITIONAL SEMANTICS

An operational semantics for while programs is usually followed by a compositional semantics (also called denotational), which is typically obtained as a least fixed point of a monotone or continuous operator on a complete lattice or complete partial order (*cf.* [2]). Here we show that such a compositional semantics can be directly obtained from the automaton $(\mathit{Prog} \times \Sigma, \alpha)$ or, equivalently, from the operational semantics $\mathcal{O}$. As a consequence, the equivalence of both semantics will be immediate by coinduction.

Recalling that for any partial function $a : \Sigma \rightharpoonup \Sigma$, we have an element $\underline{a}$ in *Prog*, we can define semantic operators of the following types

$$\begin{aligned} (-)\,;(-) \quad &: \quad (\Sigma \rightharpoonup \Sigma)^2 \to (\Sigma \rightharpoonup \Sigma) \\ \mathbf{if}\ \underline{c}\ \mathbf{then}\ (-)\ \mathbf{else}\ (-) \quad &: \quad (\Sigma \rightharpoonup \Sigma)^2 \to (\Sigma \rightharpoonup \Sigma) \\ \mathbf{while}\ \underline{c}\ \mathbf{do}\ (-) \quad &: \quad (\Sigma \rightharpoonup \Sigma) \to (\Sigma \rightharpoonup \Sigma) \end{aligned}$$

by simply putting, for partial functions $a, b : \Sigma \rightharpoonup \Sigma$,

$$\begin{aligned} a; b \quad &= \quad \mathcal{O}(\underline{a}; \underline{b}) \\ \mathbf{if}\ \underline{c}\ \mathbf{then}\ a\ \mathbf{else}\ b \quad &= \quad \mathcal{O}(\text{if } \underline{c} \text{ then } \underline{a} \text{ else } \underline{b}) \\ \mathbf{while}\ \underline{c}\ \mathbf{do}\ a \quad &= \quad \mathcal{O}(\text{while } \underline{c} \text{ do } \underline{a}). \end{aligned}$$

Next a compositional semantics $\mathcal{D} : \mathit{Prog} \to (\Sigma \rightharpoonup \Sigma)$ can be defined as usual:

$$\begin{aligned} \mathcal{D}(\underline{a}) \quad &= \quad a \\ \mathcal{D}(P; Q) \quad &= \quad \mathcal{D}(P); \mathcal{D}(Q) \\ \mathcal{D}(\text{if } \underline{c} \text{ then } P \text{ else } Q) \quad &= \quad \mathbf{if}\ \underline{c}\ \mathbf{then}\ \mathcal{D}(P)\ \mathbf{else}\ \mathcal{D}(Q) \\ \mathcal{D}(\text{while } \underline{c} \text{ do } P) \quad &= \quad \mathbf{while}\ \underline{c}\ \mathbf{do}\ \mathcal{D}(P). \end{aligned}$$

In order to prove the equivalence of $\mathcal{O}$ and $\mathcal{D}$, we first observe that

$$P \approx \underline{\mathcal{O}(P)}, \tag{3}$$

for all P in $Prog$. Secondly, weak bisimilarity is a congruence relation; that is, for instance,

$$\text{if } \; P \approx P' \text{ then } \; (\text{while } \underline{c} \text{ do } P) \approx (\text{while } \underline{c} \text{ do } P'), \tag{4}$$

and similarly for the other constructs. It is now straightforward to prove

$$\mathcal{O}(P) = \mathcal{D}(P)$$

for all P, by induction on the structure of P. Supposing, for instance, that $\mathcal{O}(P) = \mathcal{D}(P)$, it follows that

$$\begin{aligned} & \mathcal{D}(\text{while } \underline{c} \text{ do } P) \\ = \;& \textbf{while } \underline{c} \textbf{ do } \mathcal{D}(P) \\ = \;& \textbf{while } \underline{c} \textbf{ do } \mathcal{O}(P) \quad [\text{by the inductive hypothesis}] \\ = \;& \mathcal{O}(\text{while } \underline{c} \text{ do } \underline{\mathcal{O}(P)}) \quad [\text{by definition}] \\ = \;& \mathcal{O}(\text{while } \underline{c} \text{ do } P) \end{aligned}$$

since $P \approx \underline{\mathcal{O}(P)}$ implies $(\text{while } \underline{c} \text{ do } P) \approx (\text{while } \underline{c} \text{ do } \underline{\mathcal{O}(P)})$, from which the last equality follows by coinduction.

5. Notes and discussion

Since the automata we have been dealing with are coalgebras of the functor $O + (-)$ on the category of sets and functions, the present note can be considered as yet another exercise in coalgebra, similar to that of [6], which deals with deterministic automata. Thus a further illustration is given of the fact that many apparently different manifestations of circular behaviour–such as modelled by automata and while programs but furthermore including various kinds of transition systems, infinite data types and many other examples–can be described in a conceptually uniform and simple way, the only ingredients of the theory being the notions of coalgebra (= automaton), bisimulation, and homomorphism. This uniformity regards also the definitions of both operational and denotational semantics in one and the same framework, where the operational automaton does the work, so to speak, of defining the semantic operators, without the need of introducing sets carrying additional structure (such as partial orders).

The theory of ordinary bisimulation is a by now rather well developed part of (universal) coalgebra. This is not at all the case for weak bisimulation. The present definition has been inspired by Milner's canonical example of weak bisimulation for concurrent processes (*cf.* [4]). A general coalgebraic theory of weak bisimulation remains still to be formulated.

The present treatment of while programs can be related to the discipline of iteration theories (see [1] for a recent overview) as follows. From the diagram in Section 3, it follows that $\alpha^\dagger = [id_O, \alpha^\dagger] \circ \alpha$, which we recognize as one of the fundamental laws of iteration theories. The coinduction principle of (2) can be viewed as a coalgebraic counterpart of this algebraic law.

6. Proofs

The proofs of the statements in Section 1, including (1), all follow from more general observations on universal coalgebra (*cf.* [5]). Direct proofs are not very difficult either.

For (2), from left to right, consider a weak bisimulation R with $s\,R\,t$. It follows from the weak bisimulation property that s converges iff t converges. If $\alpha^\dagger(s) = o$ then $s \Longrightarrow_S s' \downarrow o$. Because R is a weak bisimulation, this implies $t \Longrightarrow_S t' \downarrow o'$ with $o = o'$. Thus $\alpha^\dagger(t) = o$.

For (2), from right to left, suppose $\alpha^\dagger(s) = \alpha^\dagger(t)$. If both are undefined then there are s_i and t_i with $s = s_0 \longrightarrow s_1 \longrightarrow s_2 \longrightarrow \cdots$ and $t = t_0 \longrightarrow t_1 \longrightarrow t_2 \longrightarrow \cdots$. Now $\{\langle s_i, t_i\rangle\}_i$ is a (weak) bisimulation. If both are defined then there exist n, m, s_i, and t_i with $s = s_0 \longrightarrow s_1 \longrightarrow \cdots \longrightarrow s_n \downarrow o$ and $t = t_0 \longrightarrow t_1 \longrightarrow \cdots \longrightarrow t_m \downarrow o$. In this case, $\{\langle s_i, t_j\rangle \mid 0 \le i \le n,\ 0 \le j \le m\}$ is a weak bisimulation.

For (3), consider a program P and a program state σ. The following relation can be readily shown to be a weak bisimulation:

$$\left\{\left(\langle P', \sigma'\rangle, \left\langle \underline{\mathcal{O}(P)}, \sigma\right\rangle\right) \mid \langle P, \sigma\rangle \Longrightarrow \langle P', \sigma'\rangle\right\},$$

using the fact that $\langle \underline{\mathcal{O}(P)}, \sigma\rangle \downarrow \tau$ if $\mathcal{O}(P)(\sigma) = \tau$, and $\langle \underline{\mathcal{O}(P)}, \sigma\rangle \to \langle \underline{\mathcal{O}(P)}, \sigma\rangle$, if $\mathcal{O}(P)(\sigma)$ is undefined.

For (4), let R be a weak bisimulation with $\langle P, \sigma\rangle\, R\, \langle P', \sigma\rangle$, for any σ in Σ. Then

$$\{(\langle \text{while } \underline{c} \text{ do } P, \sigma\rangle\,,\ \langle \text{while } \underline{c} \text{ do } P', \sigma\rangle\,) \mid \sigma \in \Sigma\} \ \cup$$

$$\{(\langle Q; (\text{while } \underline{c} \text{ do } P),\ \tau\rangle\,,\ \langle Q'; (\text{while } \underline{c} \text{ do } P'),\ \tau'\rangle\,) \mid \langle Q, \tau\rangle\ R\ \langle Q', \tau'\rangle\}$$

is a weak bisimulation, showing that (while $\underline{c}$ do P) $\approx$ (while $\underline{c}$ do P'). Similarly for the other constructs.

References

[1] S.L. Bloom and Z. Ésik, The equational logic of fixed points. *Theoret. Comput. Sci.* **179** (1997) 1–60.

[2] J.W. de Bakker, *Mathematical theory of program correctness*. Prentice-Hall International (1980).

[3] C.C. Elgot, Monadic computation and iterative algebraic theories, H.E. Rose and J.C. Shepherdson, Eds., *Logic Colloquium '73*. North-Holland, *Stud. Log. Found. Math.* **80** (1975) 175–230.

[4] R. Milner, *Communication and Concurrency*. Prentice Hall International, New York, *Prentice Hall Internat. Ser. Comput. Sci.* (1989).

[5] J.J.M.M. Rutten, Universal coalgebra: A theory of systems. Report CS-R9652, CWI, 1996. FTP-available at `ftp.cwi.nl` as pub/CWIreports/AP/CS-R9652.ps.Z. *Theoret. Comput. Sci.*, to appear.

[6] J.J.M.M. Rutten, Automata and coinduction (an exercise in coalgebra). Report SEN-R9803, CWI, 1998. FTP-available at `ftp.cwi.nl` as pub/CWIreports/SEN/SEN-R9803.ps.Z. Also in the proceedings of CONCUR '98, *Lecture Notes in Comput. Sci.* **1466** (1998) 194-218.

Received October 20, 1998. Revised April 26, 1999.