Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 162 (2006) 305–309
www.elsevier.com/locate/entcs

Confluence Thanks to Extensional Determinism

A.W. Roscoe[1,2]

Oxford University Computing Laboratory
Wolfson Building, Parks Road
Oxford OX1 3QD, UK

Abstract

A process is extensionally deterministic if, after any trace s and given any event a, it is either certain to accept or certain to refuse a (stably) after s. We show how several process algebras are capable of expressing this property and how they agree on the equivalence of deterministic processes. A number of important properties of processes P, including confluence, can be captured in terms of the determinism of some context $C[P]$.

Keywords: determinism, confluence, process algebra, CCS, CSP

1 Introduction

The first reaction of those used to thinking operationally about processes will naturally be to try to understand questions about them in that way. Operational semantics explain naturally how nondeterminism arises in process algebra: either through the uncertainty caused by the availability of τ actions or through ambiguous branching on actions. It is therefore natural to come up with an operational characterisation of determinism, examples being the banning of τ actions and ambiguous branching, and Milner's concepts of *confluent* and *weakly determinate* processes.

Process algebraists are familiar with the issue of deciding just when two nondeterministic processes are equivalent: one of our problems has been the tremendous range of congruences that make sense for that purpose. This short paper shows that we can agree about the rest of the processes, namely the deterministic ones, and gain insight in one formalism from the results and definitions known about another.

[1] The work reported in this paper was supported by a grant from US ONR
[2] Email: bill.roscoe@comlab.ox.ac.uk

doi:10.1016/j.entcs.2005.12.098

CSP has long (e.g., [1],[2]) provided what we can term an *extensional* definition of a deterministic process: one that is divergence-free, and never has both the trace $s^\frown\langle a\rangle$ and the failure $(s, \{a\})$. Under a standard interpretation of what it means to interact with an LTS, this precisely corresponds to the statement that, after any trace s and for any event a, if $\{a\}$ is offered, that either it is certain a will occur or it is certain it will be refused stably. It is most naturally decided in terms of the failures-divergences representation of a process. There are several algorithms for deciding whether or not a finite-state process is deterministic: see [6],[7]. The failures-divergences model is [8] *fully abstract* with respect to the question of whether a process is deterministic.

This definition of determinism transfers to any language with an LTS-based semantics since the sets of failures and divergences of a process are easily calculated from the LTS. In particular it makes sense for CCS, or more comfortably CCS in which unguarded recursions are banned, or are (following Walker [11]) labelled $\perp$ and treated as divergent. It also translates into the language of *testing* [3]: a process P is deterministic if and only if

$$P \; may \; t \Leftrightarrow P \; must \; t$$

if t is of the form $s^\frown\langle\omega\rangle$, for s a finite trace and ω the "success" flag. It is straightforward to verify that, over LTS's, these two definitions are equivalent. Both support the informal description of determinism as the property of being reliably testable: the same test on different occasions will yield the same result.

They are a little different from Milner's concept of *weak determinacy* [4],[5], which is that if $P \stackrel{s}{\Longrightarrow} Q$ and $P \stackrel{s}{\Longrightarrow} Q'$ then Q and Q' are weakly bisimilar. But not much different: if P is divergence-free then this is equivalent too.

It is well known in both process algebras that two deterministic/weakly determinate processes are equivalent/weakly bisimilar if and only if they have the same set of traces. We can conclude that in CSP and CCS:

- The sets of extensionally deterministic processes (namely the deterministic/divergence-free weakly determinate ones) are in effect the same.
- Both process algebras are capable of identifying them, and have (perhaps modulo an initial τ) the same equality theory for them.

We exploit this "confluence" of CCS and CSP in the rest of this paper.

2 Security

[10],[6] identified the provable lack of information flow from H to L (sets that partition P's alphabet) with the determinism of the lazy abstraction $\mathcal{L}_H(P)$ (where the events of H are concealed but made available to P nondeterministically rather than eagerly as they are in conventional hiding). A natural way of describing $\mathcal{L}_H(P)$ is

as $(P \underset{H}{\parallel} Chaos_H) \setminus H$, where

$$Chaos_H = STOP \sqcap ?x : H \to Chaos_H$$

is the most nondeterministic divergence-free process over H. We always assume P is divergence free in this section.

The strict treatment of divergence in CSP causes a problem: if an infinite sequence of H events occur without an L event it throws the value of the CSP term above to bottom. The solution to this in CSP has been to postulate the divergence to be absent, for example by using the stable failures model to calculate the above value. There is an alternative arising from the confluence of process algebras.

Proposition 2.1 *The lazy abstraction $\mathcal{L}_H(P)$ is deterministic if and only if the process $(P \underset{H}{\parallel} Chaos_H) \setminus H$ (interpreted in the standard operational semantics of CSP) is weakly determinate.*

For various reasons $Chaos_H$ cannot, in CCS, be said to be the *most* nondeterministic process. It would be interesting to investigate whether the abstraction definition $(P \underset{H}{\parallel} Chaos_H) \setminus H$ (either with the above or some CSP-equivalent but CCS-inequivalent definition of $Chaos_H$) has properties in CCS-style equivalences which are analogous to the other uses lazy abstraction has in CSP (see Chapter 12 of [6]).

3 Confluence and functionality

In [4],[5], Milner introduced the idea of a *confluent* process: P such that if $P \stackrel{s}{\Longrightarrow} Q_1$ and $P \stackrel{t}{\Longrightarrow} Q_2$ then there exists R with $Q_1 \stackrel{t-s}{\Longrightarrow} R$ and $Q_2 \stackrel{s-t}{\Longrightarrow} R$ where $s - t$ is the trace consisting of s with the events of t deleted according to multiplicity from the beginning. For example

$$\langle a, b, c, c, b, a\rangle - \langle d, c, b, a, c\rangle = \langle b, a\rangle$$

We may clearly broaden this to encompass the two R's being different but weakly bisimilar. Confluence is easily seen to imply weak determinacy. This means

- Two confluent processes are weakly bisimilar if and only if they have the same traces.
- A process is confluent if and only if it is weakly determinate and has a confluent trace set (namely one which has $s\hat{}(t - s)$ if it has s and t).

Confluent processes have many attractive properties. In [9] the author established that they are useful tools in the area of *buffer tolerance* (the study of when we can establish properties of buffered systems by checking their buffer-less analogues). The following proposition is taken from there.

Proposition 3.1 *The process P is confluent and divergence-free if and only the process $C^*[P]$, in which a one-place inwards-pointing buffer is placed on every individual event of P, is extensionally deterministic.*

The "only if" part of this result is a straightforward consequence of standard properties of confluent processes (in fact, if P is confluent, then so is $C^*[P]$). The "if" part consists of showing first that P itself is deterministic, and then showing that its trace set is confluent: any failure of this generates a piece of externally-visible nondeterminism in $C^*[P]$. The correspondence of CCS and CSP for extensionally deterministic processes easily establishes that the above also holds in CCS. In fact the proof can be adapted to establish the following slightly stronger result.

Proposition 3.2 *P is confluent if and only if $C^*[P]$ is weakly determinate.*

In [9] the author derived a similar result for *functional* processes: ones where each output stream is a prefix of a function of the input streams, which cannot refuse to input when there is no output pending and which cannot refuse to output when there is. It was shown there that (modulo a requirement that its structure of inputs is confluent) a process is functional if and only if putting an unbounded deterministic buffer (this time appropriately oriented) on each input and output *channel* creates a deterministic process. A finitary characterisation in terms of *output determinism*, where the ability to output and the value of each channel's output is completely determined by the trace, was also given. For example, a process P with two channels is a buffer (in the usual CSP sense [6], which makes sense widely) *if and only if*

$$BT[P] = COPY \quad \text{and} \quad COPY \gg P \text{ is output deterministic.}$$

where $COPY$ is a one-place buffer and $BT[P]$ places P in parallel with a process that transmits external inputs to P and P's outputs to the environment, ensuring that the lengths of its input and output traces differ by at most 1. This is straightforward in CSP thanks to the presence of many-way synchronisation. However the following definition works (up to syntax translation) in both CCS and CSP.

$$\begin{aligned} BT[P] &= P[\mathit{left} \leftrightarrow a, \mathit{right} \leftrightarrow b]\, T \\ T &= \mathit{left}?x \to a!x \to b?y \to \mathit{right}!x \to T \\ &\quad \Box\ b?y \to \mathit{right}!x \to \mathit{left}?x \to a!x \to T \end{aligned}$$

$BT[P] = COPY$ shows that the function that P computes (which exists by the output determinism condition) is the identity function. The role of the second clause in T (in the context of the check) is to ensure that P never outputs more than one item per input. This gives a very finitary check of an infinitary specification, which works equally well in CSP and CCS.

References

[1] Brookes S.D., C.A.R. Hoare and A.W. Roscoe, *A theory of communicating sequential processes*, Journal of the ACM **31**, 3, 560–599, 1984.

[2] Brookes, S.D., and A.W. Roscoe, *An improved failures model for CSP*, Proceedings of the Pittsburgh seminar on concurrency, Springer LNCS 197, 1985.

[3] de Nicola, R., and M. Hennessy, *Testing equivalences for processes*, TCS **34**, 1, 83–134, 1987.

[4] Milner R., "A calculus of communicating systems", Springer LNCS **92**, 1980.

[5] Milner R., "Communication and concurrency", Prentice Hall, 1989.

[6] Roscoe, A. W., "The Theory and Practice of Concurrency," Prentice Hall Series in Computer Science, Prentice Hall Publishers, London, New York (1998), 565pp. With associated web site web.comlab.ox.ac.uk/oucl/publications/books/concurrency/.

[7] A.W. Roscoe, *Finitary refinement checks for infinitary specifications*, Proc CPA 2004. Obtainable from web.comlab.ox.ac.uk/oucl/work/bill.roscoe/publications/96.pdf.

[8] A.W. Roscoe, *Revivals, stuckness and reponsiveness*, Unpublished manuscript (2005) Obtainable from web.comlab.ox.ac.uk/oucl/work/bill.roscoe/publications/105.pdf.

[9] A.W. Roscoe, *The pursuit of buffer tolerance*, Unpublished manuscript (2005) Obtainable from web.comlab.ox.ac.uk/oucl/work/bill.roscoe/publications/106.pdf.

[10] A.W. Roscoe, J.C.P. Woodcock and L. Wulf, *Non-interference through determinism*, Journal of Computer Security **4**, 1, 27–54, 1996.

[11] D.J. Walker, *Bisimulation and divergence in CCS*, Information and Computation **85** pp202–241 (1990).