DEDUCING CAUSAL RELATIONSHIPS IN CCS

Jeremy Gunawardena,
Hewlett-Packard Laboratories, Information Systems Centre,
Filton Road, Stoke Gifford, Bristol BS12 6QZ.

Abstract

We introduce purely parallel processes, a class of finite CCS processes defined by a restricted syntax which allows synchronization but forbids choice. Such processes are deterministic in the sense of Milner, [7]. We define a function which associates to each purely parallel process a labelled partially ordered set which is also deterministic but in the sense of Vaandrager, [10]. Our main result is that this induces a bijection from equivalence classes of purely parallel processes, under weak bisimulation, to finite deterministic pomsets. The motivation for this work comes from practical problems in verification and we give two applications of our results.

1 Introduction

In this paper we are concerned with studying the causal relationships between events in a CCS process. We concentrate on the subset of purely parallel processes, which is defined by a syntax which allows synchronization - of a restricted form - but forbids explicit use of the choice operator. The restrictions force all purely parallel processes to be deterministic in the sense of Milner, [7, Chapter 11]. In particular, the behaviour of such a process (ie: its equivalence class under weak bisimulation) is entirely determined by its traces.

We construct a function \( \gamma \) which associates to each purely parallel process, \( P \), a labelled partially ordered set (poset), \( \gamma(P) \), which encodes the causal relationships between the occurrences of observable actions. This gives, in effect, a causal denotational semantics for purely parallel processes. A key property is that the traces of \( P \) coincide with the sequences (linearizations) of \( \gamma(P) \).

The poset \( \gamma(P) \) is also deterministic, but now in the sense of Vaandrager, [10]. Vaandrager has shown that two deterministic event structures are isomorphic if, and only if, they have the same set of step sequences. We observe that when the event structures are conflict-free (ie: they correspond to posets) a stronger result holds: two deterministic posets are isomorphic if, and only if, they have the same set of sequences. It follows that the isomorphism class of \( \gamma(P) \) is also determined by trace level information.

Our main result is that \( \gamma \) induces a bijection from equivalence classes of purely parallel processes, under weak bisimulation, to isomorphism classes of finite, deterministic posets. In one direction this follows from the properties mentioned above of determinism for processes and posets. We relate the weak bisimulation class of the process to the isomorphism class of the poset by "bouncing" off the trace level. The other direction requires a careful explicit construction. We sketch the proofs; full details may be found in [1].

In the second part we discuss two applications of the above results. The first is the verification of a proprietary key-distribution protocol. This provided the original motivation for the work described here. The second is a rigorous justification of a test, due to Kung, [4], for deadlock-freedom of systolic programs.

It is a pleasure to thank Robin Milner for his remarks on an early version of this paper; they provided an essential stimulus at a critical stage of the work. We are also grateful to two anonymous referees whose comments led to a considerable improvement in presentation. This work was undertaken as part of project VESPA at Hewlett-Packard's Information Systems Centre in Bristol, England.

2 Purely Parallel Processes

We shall assume familiarity with CCS, as expounded in either [6] or, more recently, [7], and also with a few of the elementary concepts of the theory of event structures, [11]. For the purposes of this paper a more appropriate reference for the latter is [10].

A few comments on notation and terminology may be appropriate. CCS processes engage in actions. The observable actions are assumed to be elements of some universe, \( \mathcal{S} \), which does not include the hidden action \( \tau \). The notation \( P \overset{\alpha}{\rightarrow} Q \), where \( \alpha \in \mathcal{S} \cup \{\tau\} \), indicates that the process \( P \) engages in the action \( \alpha \) and evolves into the process \( Q \). This indexed relation can be extended, in a well-known way, to a relation,
$\Rightarrow$, indexed over strings $s \in \Sigma^\omega$ of observable actions, where hidden events are now ignored. A string $s$ is a trace of $P$ if $P \Rightarrow Q$ for some $Q$. The empty string $\epsilon$ is always a trace: by definition $P \Rightarrow P$.

We shall always use the weak semantics for CCS: processes $P$ and $Q$ are deemed to have the same behaviour if there is a weak bisimulation between them, denoted $P \approx Q$. Since all our processes are finite we shall feel free to use the phrase “observational equivalence” as a synonym for this.

In describing causal relationships it appears essential to employ a different mathematical setting to the algebra of expressions used in CCS or other process calculi. Causality exists not at the level of actions but at the level of action occurrences. In the event structure view, action occurrences, or events, are the primitive notion. The action is recovered from the event by a labelling function which provides the necessary extra layer of abstraction: different events may have the same action label and hence represent (different occurrences of) the same action.

Semantics based on causality are frequently assumed to adopt the true concurrency view: the interleaving equation $a|b = a.b + b.a$ is assumed not to hold. We should stress that this is not the case in the present paper.

2.1 Pure parallelism, confluence and determinism

A purely parallel process is an expression constructed from the following syntax.

$$P ::= (NIL \mid \alpha \mid P; P \mid P[M] \mid P||P)$$

The set of all such expressions will be denoted $p^3$. With the exception of the operator “;”, which corresponds to sequentiality, all the operators in the syntax are derived from the standard CCS operators. Instead of describing them in this way, it may be more illuminating to write down the derived rewrite rules which they obey. This defines $p^3$ as a mini concurrency language in its own right but we shall prefer to think of it as a subset of CCS. The introduction of sequentiality in place of prefixing - which may be recovered as a special case via $\alpha.P = \alpha; P$ - is very convenient in applications but is essentially unnecessary. It turns out, as a Corollary to the proof of Theorem 3 below, that any $P \in p^3$ is observationally equivalent to a process written in the syntax

$$P ::= (NIL \mid \alpha.P \mid P||P).$$

The rewrite rules have the following form.

- $NIL$ does nothing and has no rules.
- $\alpha$ introduces an action which may be the hidden action $\tau$.
  $$\alpha \xrightarrow{\alpha} NIL$$
- $Q; R$ is sequential composition. Recall from [6, §5.6] that $P$ is directly equivalent to $NIL$, written $P \equiv NIL$, if and only if $P$ offers no actions, hidden or otherwise.
  $$\frac{Q \xrightarrow{\alpha} Q_1}{Q; R \xrightarrow{\alpha} Q_1; R} \quad \frac{R \xrightarrow{\beta} R_1}{Q; R \xrightarrow{\beta} Q; R_1}$$
- $Q[M]$ is restriction over a set $M$ of actions.
  $$\frac{Q \xrightarrow{\alpha} Q_1 \quad (\alpha \notin M)}{Q[M] \xrightarrow{\alpha} Q_1[M]}$$
- $Q||R$ is parallel composition of a restricted form. In order to form $Q||R$ it is necessary that $L(Q) \cap L(R) = \emptyset$. Here, $L(Q)$ denotes the strict sort of $P$ which is defined by structural induction as in [6, §5.5]. Let $L(Q, R)$ denote the set of actions on which $Q$ and $R$ can synchronize: $L(Q, R) = (L(Q) \cap L(R)) \cup (L(R) \cap L(Q))$, 