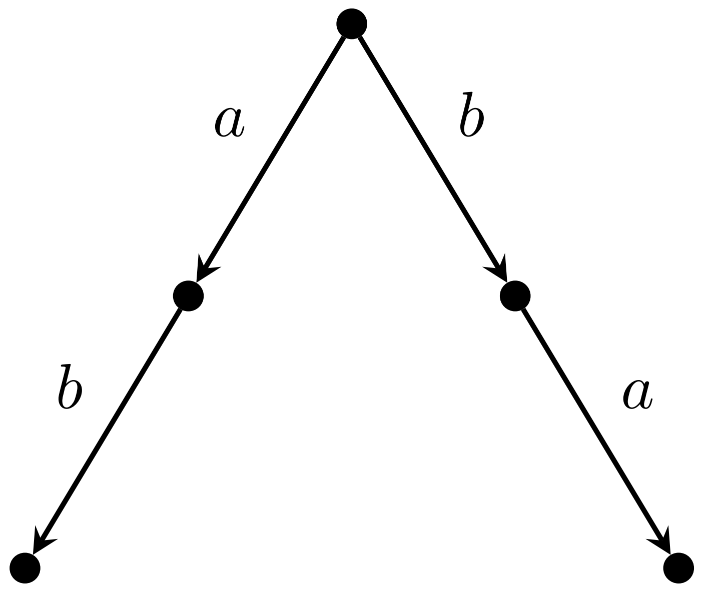
From Transitions to Executions

Eleftherios Matsikoudis and Edward A. Lee
University of California, Berkeley

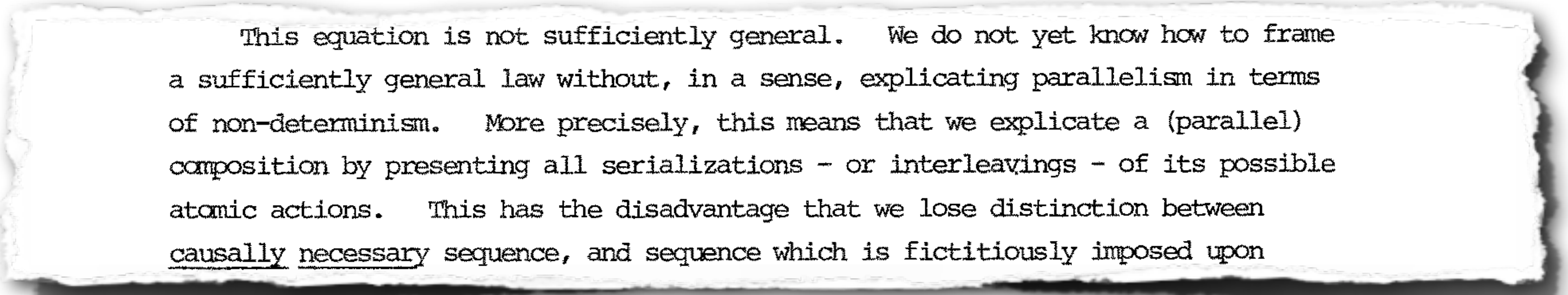
CMOS'12

Motivation

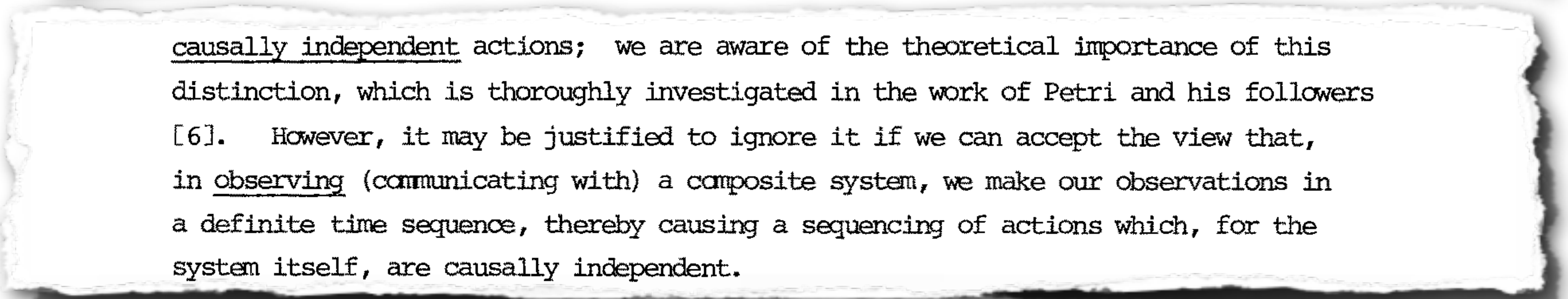
$$a.\mathbf{0} \mid b.\mathbf{0} = a.b.\mathbf{0} + b.a.\mathbf{0}$$



Motivation cont.



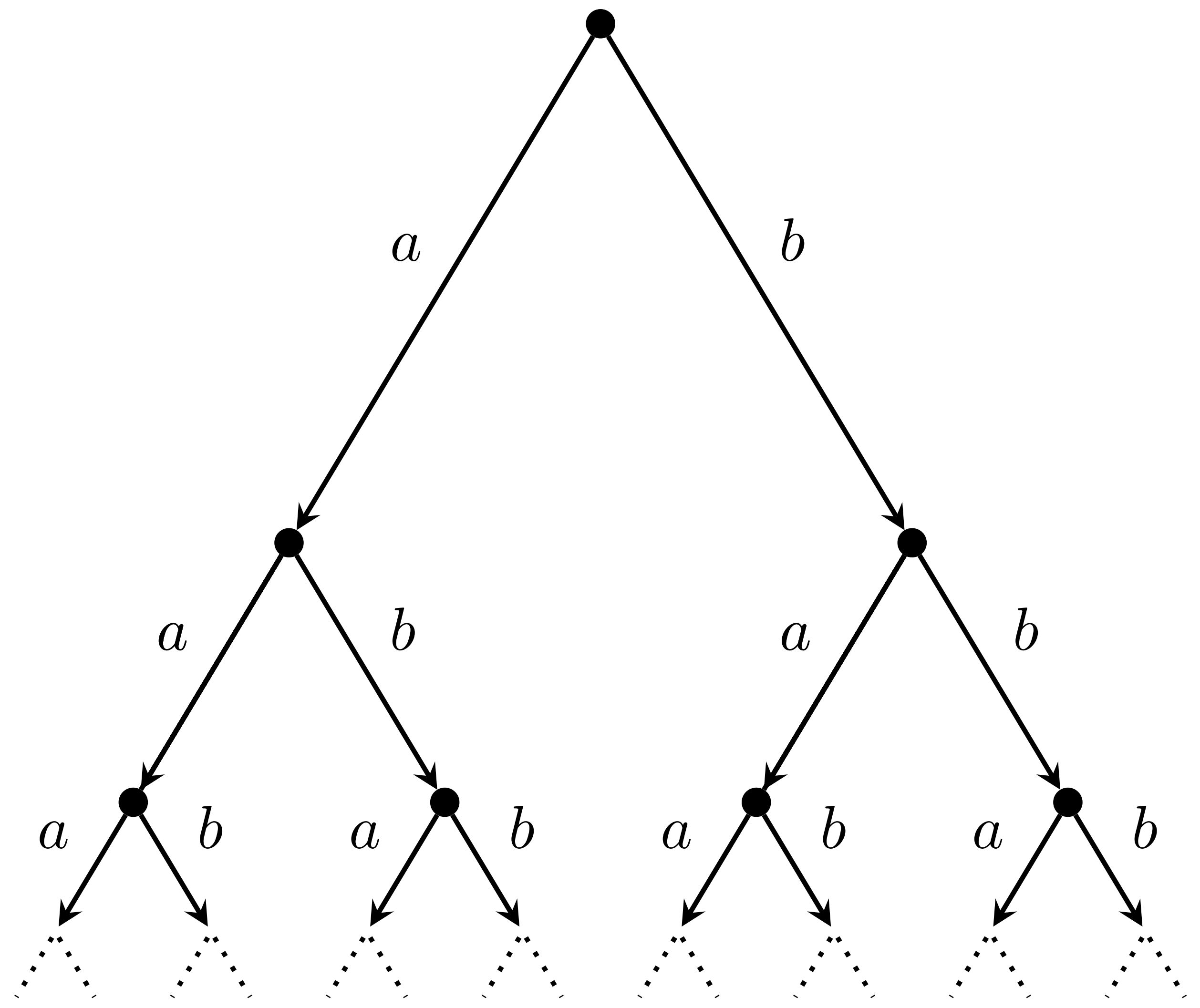
This equation is not sufficiently general. We do not yet know how to frame a sufficiently general law without, in a sense, explicating parallelism in terms of non-determinism. More precisely, this means that we explicate a (parallel) composition by presenting all serializations - or interleavings - of its possible atomic actions. This has the disadvantage that we lose distinction between causally necessary sequence, and sequence which is fictitiously imposed upon



causally independent actions; we are aware of the theoretical importance of this distinction, which is thoroughly investigated in the work of Petri and his followers [6]. However, it may be justified to ignore it if we can accept the view that, in observing (communicating with) a composite system, we make our observations in a definite time sequence, thereby causing a sequencing of actions which, for the system itself, are causally independent.

Motivation cont.

$$\mathbf{fix}(X = a.X) \mid \mathbf{fix}(X = b.X) = \mathbf{fix}(X = a.X + b.X)$$

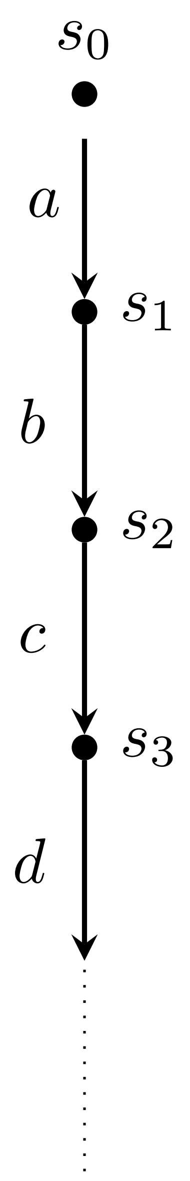


Motivation cont.

$$\begin{tikzcd}
s_0 \arrow[d, "a"'] \\
s_1
\end{tikzcd}$$

Motivation cont.

$$\begin{tikzcd} s_0 \arrow[d, "a"'] \\ s_1 \arrow[d, "b"'] \\ s_2 \arrow[d, "c"'] \\ s_3 \arrow[d, "d"'] \\ \vdots \end{tikzcd}$$



Labelled transition coalgebras

$$\mathsf{Pow} \circ (L \times \mathsf{Id})$$

Thm. *B is a bisimulation between $\langle C_1, \tau_1 \rangle$ and $\langle C_2, \tau_2 \rangle$ if and only if for any c_1 and c_2 such that $c_1 \mathrel{B} c_2$, the following are true:*

(a) if $c_1 \xrightarrow{l}_{\tau_1} c_1'$, then there is c_2' such that $c_2 \xrightarrow{l}_{\tau_2} c_2'$ and $c_1' \mathrel{B} c_2'$;

(b) if $c_2 \xrightarrow{l}_{\tau_2} c_2'$, then there is c_1' such that $c_1 \xrightarrow{l}_{\tau_1} c_1'$ and $c_1' \mathrel{B} c_2'$.

Labelled execution coalgebras

$$\mathsf{Pow} \circ \mathsf{Seq} \circ (L \times \mathsf{Id})$$

Thm. *B is a bisimulation between $\langle C_1, \varepsilon_1 \rangle$ and $\langle C_2, \varepsilon_2 \rangle$ if and only if for any c_1 and c_2 such that $c_1 \; B \; c_2$, the following are true:*

(a) if $c_1 \rhd_{\varepsilon_1} e_1$, then there is e_2 such that $c_2 \rhd_{\varepsilon_2} e_2$ and $e_1 \; \mathsf{Seq}(L \times B) \; e_2$;

(b) if $c_2 \rhd_{\varepsilon_2} e_2$, then there is e_1 such that $c_1 \rhd_{\varepsilon_1} e_1$ and $e_1 \; \mathsf{Seq}(L \times B) \; e_2$.

First side-effect

$$\begin{tikzcd} s_0 \arrow[d, "l_0"] \\ s_1 \end{tikzcd}$$

$$\begin{tikzcd} s_2 \arrow[d, "l_0"] & s_3 \arrow[d, "l_1"] \\ s_3 & s_4 \end{tikzcd}$$

s_0 and s_2 are *not* bisimilar,
even though the only execution starting from s_0
and the only execution starting from s_2
are in perfect agreement.

Second side-effect

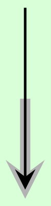
$$\begin{tikzcd}
\bullet\, s_0 \arrow[d, "l_0"] & \bullet\, s_0 \arrow[d, "l_0"] \\
\bullet\, s_1 \arrow[d, "l_1"] & \bullet\, s_2 \arrow[d, "l_2"] \\
\bullet\, s_3 & \bullet\, s_4
\end{tikzcd}
\qquad\qquad
\begin{tikzcd}
\bullet\, s_5 \arrow[d, "l_0"] & \bullet\, s_5 \arrow[d, "l_0"] \\
\bullet\, s_6 \arrow[d, "l_1"] & \bullet\, s_6 \arrow[d, "l_2"] \\
\bullet\, s_7 & \bullet\, s_8
\end{tikzcd}$$

s_0 and s_5 *are* bisimilar,
even though the two executions starting from s_0 diverge right away at s_0,
whereas those starting from s_5 diverge after the first step at s_6.

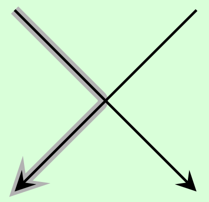
Abrahamson coalgebras

$\langle C, \varepsilon \rangle$ is Abrahamson if and only if the following are true:

(i) $\langle C, \varepsilon \rangle$ is suffix closed:



(ii) $\langle C, \varepsilon \rangle$ is fusion closed:



Thm. L-$\mathbf{LEC}_{\mathrm{Abr}}$ *is a* $(\mathsf{Pow} \circ \mathsf{Seq} \circ (L \times \mathsf{Id}))$*-covariety.*

Cor. L-$\mathbf{LEC}_{\mathrm{Abr}}$ *has a final coalgebra.*

Underlying labelled transition coalgebras

for every $S \in \mathsf{Pow\,Seq}(L \times C)$, $\eta(C)(S) = \{\mathsf{head}\, s \mid s \in S \text{ and } s \neq \langle\ \rangle\}$

$$\begin{tikzcd}
\mathsf{Pow\,Seq}(L \times C_1) \arrow[r, "\eta(C_1)"] \arrow[d, "\mathsf{Pow\,Seq}(L \times f)"'] & \mathsf{Pow}(L \times C_1) \arrow[d, "\mathsf{Pow}(L \times f)"] \\
\mathsf{Pow\,Seq}(L \times C_2) \arrow[r, "\eta(C_2)"'] & \mathsf{Pow}(L \times C_2)
\end{tikzcd}$$

Underlying labelled transition coalgebras cont.

Thm. *If h is a homomorphism from $\langle C_1, \varepsilon_1\rangle$ to $\langle C_2, \varepsilon_2\rangle$, then h is a homomorphism from $\langle C_1, \eta(C_1) \circ \varepsilon_1\rangle$ to $\langle C_2, \eta(C_2) \circ \varepsilon_2\rangle$.*

Cor. *If B is a bisimulation between $\langle C_1, \varepsilon_1\rangle$ and $\langle C_2, \varepsilon_2\rangle$, then B is a bisimulation between $\langle C_1, \eta(C_1) \circ \varepsilon_1\rangle$ and $\langle C_2, \eta(C_2) \circ \varepsilon_2\rangle$.*

Not suffix closed

$$\begin{tikzcd}
s_0 \arrow[d, "l"] \\
s_1
\end{tikzcd}
\qquad\qquad
\begin{tikzcd}
s_2 \arrow[d, "l"] \\
s_3 \arrow[d, "l"] \\
s_4
\end{tikzcd}$$

s_0 and s_2 are bisimilar among the two underlying labelled transition systems, which are identical, but not among the two labelled execution systems.

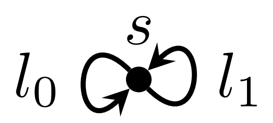
Not fusion closed

$$\begin{tikzcd}
s_0 \arrow[d, "l_0"] & s_1 \arrow[d, "l_1"] & s_1 \arrow[d, "l_2"] \\
s_1 \arrow[d, "l_1"] & s_2 & s_3 \\
s_2 & &
\end{tikzcd}$$

$$\begin{tikzcd}
s_4 \arrow[d, "l_0"] & s_5 \arrow[d, "l_1"] & s_5 \arrow[d, "l_2"] \\
s_5 \arrow[d, "l_2"] & s_6 & s_7 \\
s_7 & &
\end{tikzcd}$$

s_0 and s_2 are bisimilar among the two underlying labelled transition systems, which are identical, but not among the two labelled execution systems.

Not limit closed



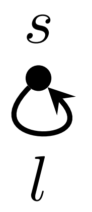
s is not bisimilar with itself among the overlying labelled execution system whose executions correspond to all infinite paths in the diagram,
and that whose executions correspond to the infinite paths that go through each of the two loops infinitely often.

Not limit closed cont.

$$\begin{tikzcd}
 & s_0 \arrow[loop above, "l_0"] \arrow[d, "l_0"] & \\
 & s_1 \arrow[dl, "l_0"'] \arrow[dr, "l_1"] & \\
s_2 \arrow[loop below, "l_0"] & & s_3
\end{tikzcd}$$

s_0 is not bisimilar with itself among the overlying Abrahamson system whose executions starting from s_0 correspond to all maximal paths in the diagram, and that whose executions are all the executions of the first system except the infinite execution stuttering around s_0.

Indeterminately terminating



s is not bisimilar with itself among the overlying labelled execution system whose single execution corresponds to the only infinite path in the diagram, and that whose executions correspond to all finite paths and the only infinite path.

$$\varepsilon(c) = \emptyset$$

s is not bisimilar with itself among the overlying labelled execution system that has no execution, and that whose only execution is the empty execution.

Generable coalgebras

for every $\tau : C \to \mathsf{Pow}(L \times C)$, $(\mathsf{gen}\,\tau)(c) = \{e \mid e \text{ is a } \tau\text{-orbit of } c\}$

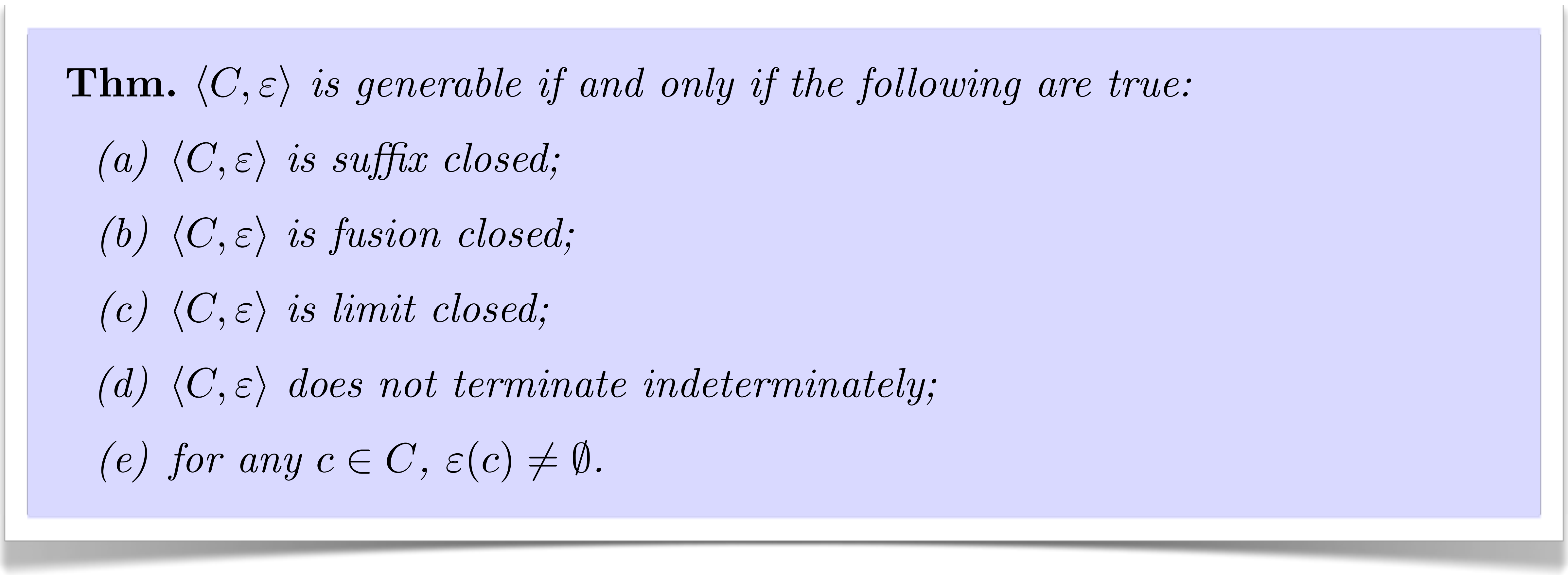
$\langle C, \varepsilon \rangle$ is generable if and only if there is τ such that $\varepsilon = \mathsf{gen}\,\tau$

Prop. *The following are true:*

(a) $\eta(C) \circ \mathsf{gen}\,\tau = \tau$;

(b) if ε *is generable, then* $\varepsilon = \mathsf{gen}(\eta(C) \circ \varepsilon)$.

Characterization



Thm. $\langle C, \varepsilon\rangle$ *is generable if and only if the following are true:*

(a) $\langle C, \varepsilon\rangle$ *is suffix closed;*

(b) $\langle C, \varepsilon\rangle$ *is fusion closed;*

(c) $\langle C, \varepsilon\rangle$ *is limit closed;*

(d) $\langle C, \varepsilon\rangle$ *does not terminate indeterminately;*

(e) for any $c \in C$, $\varepsilon(c) \neq \emptyset$.

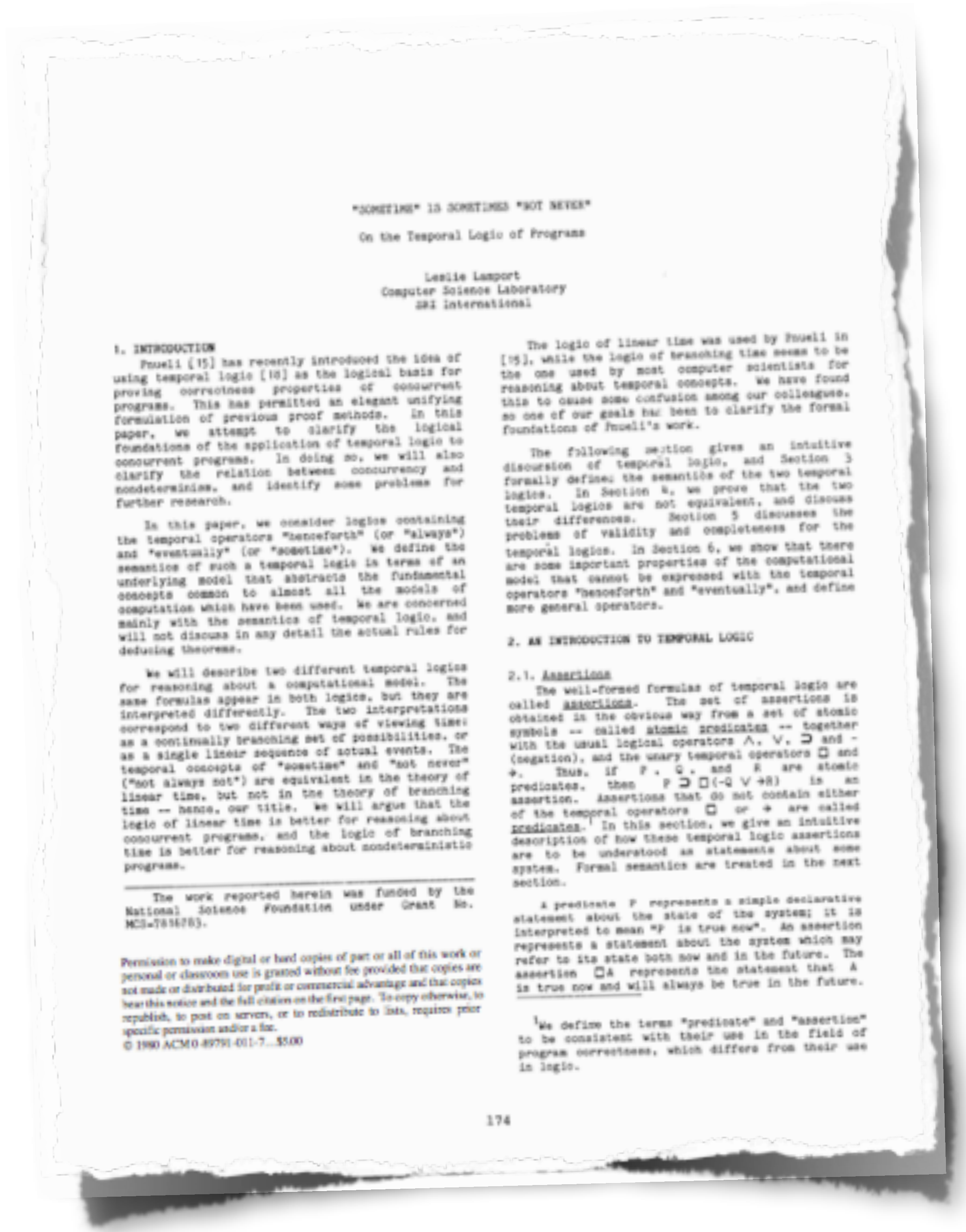
Characterization cont.

Thm. *If* $\langle C_1, \varepsilon_1 \rangle$ *and* $\langle C_2, \varepsilon_2 \rangle$ *are generable, then* h *is a homomorphism from* $\langle C_1, \varepsilon_1 \rangle$ *to* $\langle C_2, \varepsilon_2 \rangle$ *if and only if* h *is a homomorphism from* $\langle C_1, \eta(C_1) \circ \varepsilon_1 \rangle$ *to* $\langle C_2, \eta(C_2) \circ \varepsilon_2 \rangle$.

Cor. *If* $\langle C_1, \varepsilon_1 \rangle$ *and* $\langle C_2, \varepsilon_2 \rangle$ *are generable, then* B *is a bisimulation between* $\langle C_1, \varepsilon_1 \rangle$ *and* $\langle C_2, \varepsilon_2 \rangle$ *if and only if* B *is a bisimulation between* $\langle C_1, \eta(C_1) \circ \varepsilon_1 \rangle$ *and* $\langle C_2, \eta(C_2) \circ \varepsilon_2 \rangle$.

Thm. L-$\mathbf{LEC}_{\mathrm{Gen}}$ *and* L-$\mathbf{LTC}$ *are isomorphic.*

Related work



"SOMETIME" IS SOMETIMES "NOT NEVER"

On the Temporal Logic of Programs

Leslie Lamport
Computer Science Laboratory
SRI International

1. INTRODUCTION

Pnueli [15] has recently introduced the idea of using temporal logic [18] as the logical basis for proving correctness properties of concurrent programs. This has permitted an elegant unifying formulation of previous proof methods. In this paper, we attempt to clarify the logical foundations of the application of temporal logic to concurrent programs. In doing so, we will also clarify the relation between concurrency and nondeterminism, and identify some problems for further research.

In this paper, we consider logics containing the temporal operators "henceforth" (or "always") and "eventually" (or "sometime"). We define the semantics of such a temporal logic in terms of an underlying model that abstracts the fundamental concepts common to almost all the models of computation which have been used. We are concerned mainly with the semantics of temporal logic, and will not discuss in any detail the actual rules for deducing theorems.

We will describe two different temporal logics for reasoning about a computational model. The same formulas appear in both logics, but they are interpreted differently. The two interpretations correspond to two different ways of viewing time: as a continually branching set of possibilities, or as a single linear sequence of actual events. The temporal concepts of "sometime" and "not never" ("not always not") are equivalent in the theory of linear time, but not in the theory of branching time -- hence, our title. We will argue that the logic of linear time is better for reasoning about concurrent programs, and the logic of branching time is better for reasoning about nondeterministic programs.

The work reported herein was funded by the National Science Foundation under Grant No. MCS-7816783.

The logic of linear time was used by Pnueli in [15], while the logic of branching time seems to be the one used by most computer scientists for reasoning about temporal concepts. We have found this to cause some confusion among our colleagues, so one of our goals has been to clarify the formal foundations of Pnueli's work.

The following section gives an intuitive discussion of temporal logic, and Section 3 formally defines the semantics of the two temporal logics. In Section 4, we prove that the two temporal logics are not equivalent, and discuss their differences. Section 5 discusses the problems of validity and completeness for the temporal logics. In Section 6, we show that there are some important properties of the computational model that cannot be expressed with the temporal operators "henceforth" and "eventually", and define more general operators.

2. AN INTRODUCTION TO TEMPORAL LOGIC

2.1. Assertions

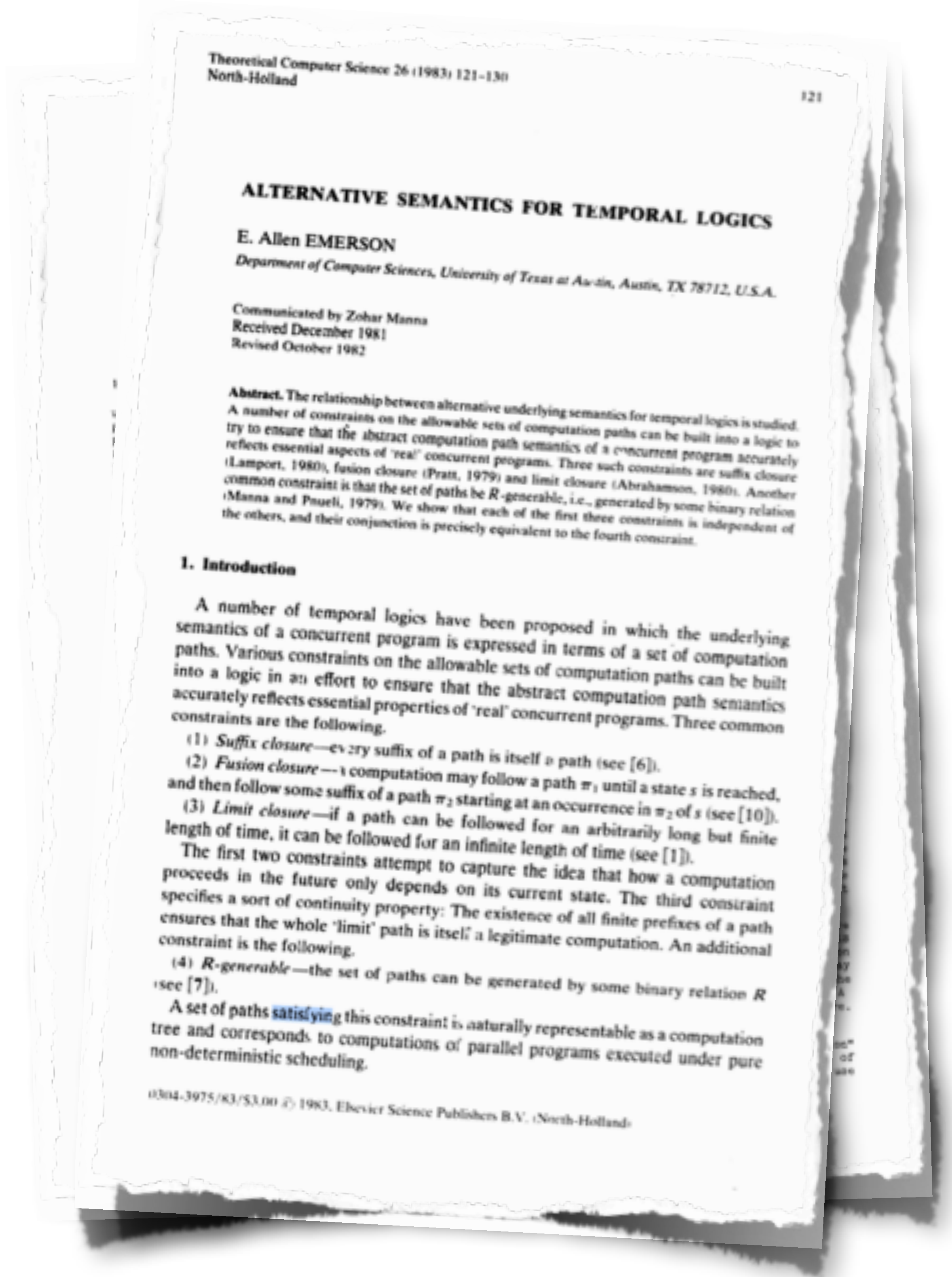
The well-formed formulas of temporal logic are called assertions. The set of assertions is obtained in the obvious way from a set of atomic symbols -- called atomic predicates -- together with the usual logical operators $\wedge$, $\vee$, $\supset$ and $-$ (negation), and the unary temporal operators $\Box$ and $\Diamond$. Thus, if P, Q, and R are atomic predicates, then $P \supset \Box(-Q \vee \Diamond R)$ is an assertion. Assertions that do not contain either of the temporal operators $\Box$ or $\Diamond$ are called predicates.[1] In this section, we give an intuitive description of how these temporal logic assertions are to be understood as statements about some system. Formal semantics are treated in the next section.

A predicate P represents a simple declarative statement about the state of the system; it is interpreted to mean "P is true now". An assertion represents a statement about the system which may refer to its state both now and in the future. The assertion $\Box A$ represents the statement that A is true now and will always be true in the future.

[1] We define the terms "predicate" and "assertion" to be consistent with their use in the field of program correctness, which differs from their use in logic.

174

Related work



Theoretical Computer Science 26 (1983) 121–130
North-Holland

ALTERNATIVE SEMANTICS FOR TEMPORAL LOGICS

E. Allen EMERSON

Department of Computer Sciences, University of Texas at Austin, Austin, TX 78712, U.S.A.

Communicated by Zohar Manna
Received December 1981
Revised October 1982

Abstract. The relationship between alternative underlying semantics for temporal logics is studied. A number of constraints on the allowable sets of computation paths can be built into a logic to try to ensure that the abstract computation path semantics of a concurrent program accurately reflects essential aspects of 'real' concurrent programs. Three such constraints are suffix closure (Lamport, 1980), fusion closure (Pratt, 1979) and limit closure (Abrahamson, 1980). Another common constraint is that the set of paths be *R*-generable, i.e., generated by some binary relation (Manna and Pnueli, 1979). We show that each of the first three constraints is independent of the others, and their conjunction is precisely equivalent to the fourth constraint.

1. Introduction

A number of temporal logics have been proposed in which the underlying semantics of a concurrent program is expressed in terms of a set of computation paths. Various constraints on the allowable sets of computation paths can be built into a logic in an effort to ensure that the abstract computation path semantics accurately reflects essential properties of 'real' concurrent programs. Three common constraints are the following.

(1) *Suffix closure*—every suffix of a path is itself a path (see [6]).

(2) *Fusion closure*—a computation may follow a path π_1 until a state s is reached, and then follow some suffix of a path π_2 starting at an occurrence in π_2 of s (see [10]).

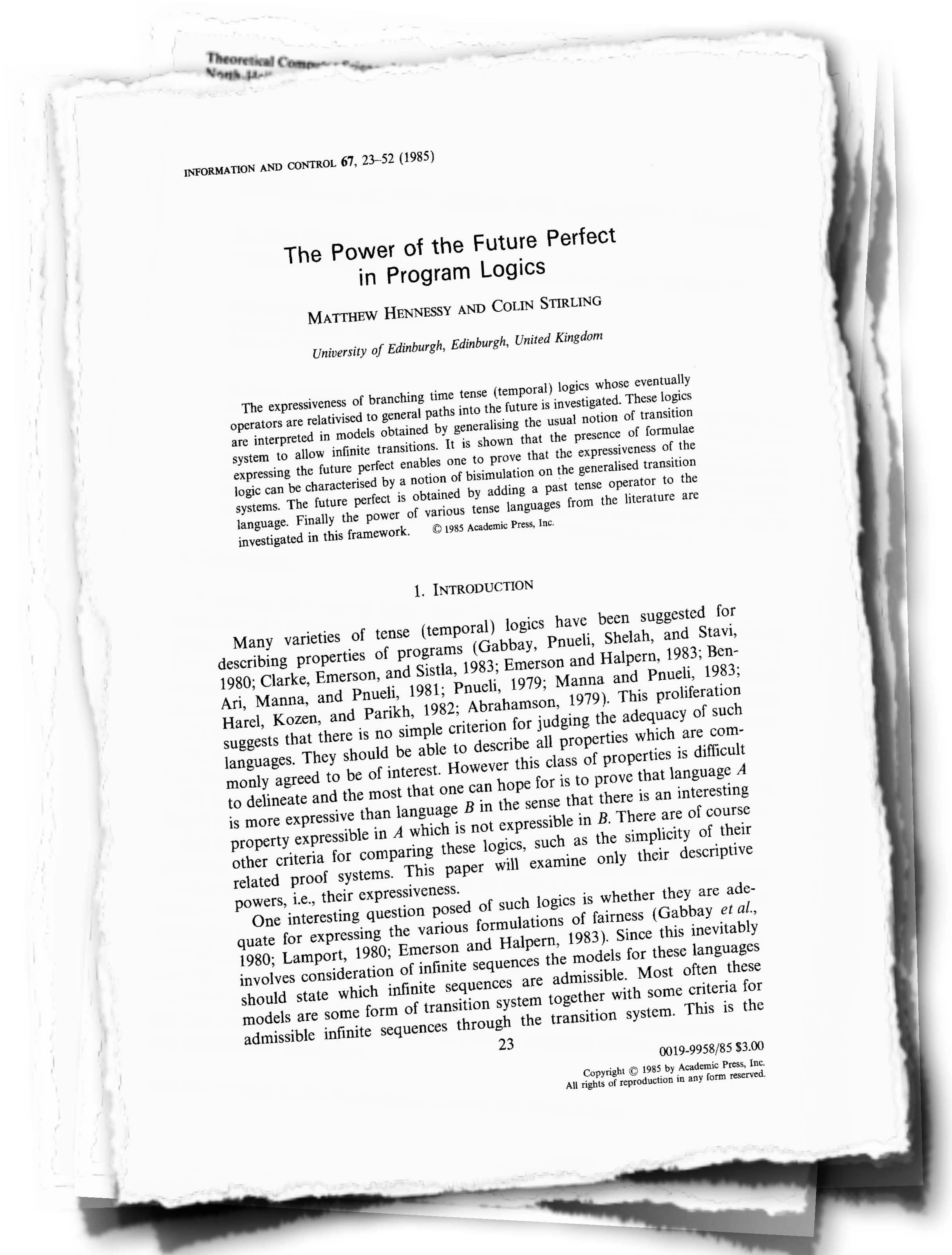
(3) *Limit closure*—if a path can be followed for an arbitrarily long but finite length of time, it can be followed for an infinite length of time (see [1]).

The first two constraints attempt to capture the idea that how a computation proceeds in the future only depends on its current state. The third constraint specifies a sort of continuity property: The existence of all finite prefixes of a path ensures that the whole 'limit' path is itself a legitimate computation. An additional constraint is the following.

(4) *R-generable*—the set of paths can be generated by some binary relation R (see [7]).

A set of paths satisfying this constraint is naturally representable as a computation tree and corresponds to computations of parallel programs executed under pure non-deterministic scheduling.

Related work



Theoretical Comp
North-Hol

INFORMATION AND CONTROL 67, 23–52 (1985)

The Power of the Future Perfect in Program Logics

MATTHEW HENNESSY AND COLIN STIRLING

University of Edinburgh, Edinburgh, United Kingdom

The expressiveness of branching time tense (temporal) logics whose eventually operators are relativised to general paths into the future is investigated. These logics are interpreted in models obtained by generalising the usual notion of transition system to allow infinite transitions. It is shown that the presence of formulae expressing the future perfect enables one to prove that the expressiveness of the logic can be characterised by a notion of bisimulation on the generalised transition systems. The future perfect is obtained by adding a past tense operator to the language. Finally the power of various tense languages from the literature are investigated in this framework. © 1985 Academic Press, Inc.

1. INTRODUCTION

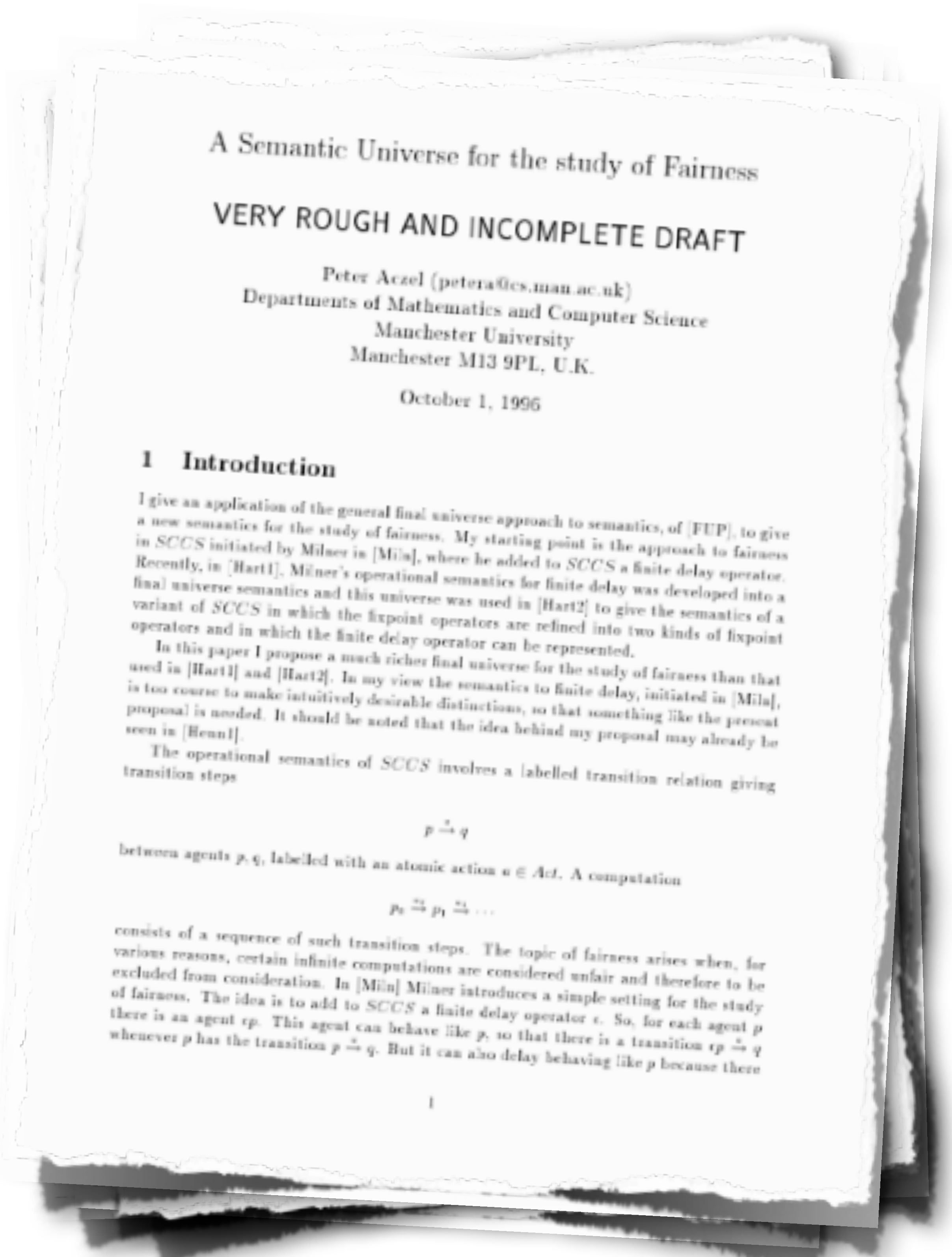
Many varieties of tense (temporal) logics have been suggested for describing properties of programs (Gabbay, Pnueli, Shelah, and Stavi, 1980; Clarke, Emerson, and Sistla, 1983; Emerson and Halpern, 1983; Ben-Ari, Manna, and Pnueli, 1981; Pnueli, 1979; Manna and Pnueli, 1983; Harel, Kozen, and Parikh, 1982; Abrahamson, 1979). This proliferation suggests that there is no simple criterion for judging the adequacy of such languages. They should be able to describe all properties which are commonly agreed to be of interest. However this class of properties is difficult to delineate and the most that one can hope for is to prove that language *A* is more expressive than language *B* in the sense that there is an interesting property expressible in *A* which is not expressible in *B*. There are of course other criteria for comparing these logics, such as the simplicity of their related proof systems. This paper will examine only their descriptive powers, i.e., their expressiveness.

One interesting question posed of such logics is whether they are adequate for expressing the various formulations of fairness (Gabbay *et al.*, 1980; Lamport, 1980; Emerson and Halpern, 1983). Since this inevitably involves consideration of infinite sequences the models for these languages should state which infinite sequences are admissible. Most often these models are some form of transition system together with some criteria for admissible infinite sequences through the transition system. This is the

23

0019-9958/85 $3.00
Copyright © 1985 by Academic Press, Inc.
All rights of reproduction in any form reserved.

Related work



A Semantic Universe for the study of Fairness

VERY ROUGH AND INCOMPLETE DRAFT

Peter Aczel (petera@cs.man.ac.uk)
Departments of Mathematics and Computer Science
Manchester University
Manchester M13 9PL, U.K.

October 1, 1996

1 Introduction

I give an application of the general final universe approach to semantics, of [FUP], to give
a new semantics for the study of fairness. My starting point is the approach to fairness
in *SCCS* initiated by Milner in [Miln], where he added to *SCCS* a finite delay operator.
Recently, in [Hart1], Milner's operational semantics for finite delay was developed into a
final universe semantics and this universe was used in [Hart2] to give the semantics of a
variant of *SCCS* in which the fixpoint operators are refined into two kinds of fixpoint
operators and in which the finite delay operator can be represented.

In this paper I propose a much richer final universe for the study of fairness than that
used in [Hart1] and [Hart2]. In my view the semantics to finite delay, initiated in [Miln],
is too course to make intuitively desirable distinctions, so that something like the present
proposal is needed. It should be noted that the idea behind my proposal may already be
seen in [Henn1].

The operational semantics of *SCCS* involves a labelled transition relation giving
transition steps

$$p \xrightarrow{a} q$$

between agents p, q, labelled with an atomic action $a \in Act$. A computation

$$p_0 \xrightarrow{a_0} p_1 \xrightarrow{a_1} \cdots$$

consists of a sequence of such transition steps. The topic of fairness arises when, for
various reasons, certain infinite computations are considered unfair and therefore to be
excluded from consideration. In [Miln] Milner introduces a simple setting for the study
of fairness. The idea is to add to *SCCS* a finite delay operator ϵ. So, for each agent p
there is an agent ϵp. This agent can behave like p, so that there is a transition $\epsilon p \xrightarrow{a} q$
whenever p has the transition $p \xrightarrow{a} q$. But it can also delay behaving like p because there

1

Related work

URL: http://www.elsevier.nl/locate/entcs/volume29.html 25 pages

A Fully Abstract Presheaf Semantics of SCCS with Finite Delay

Thomas T. Hildebrandt

BRICS, Computer Science Department, Aarhus University , Denmark[1,2]

Abstract

We present a presheaf model for the observation of *infinite* as well as finite computations. We apply it to give a *denotational* semantics of SCCS with finite delay, in which the meanings of recursion are given by *final* coalgebras and meanings of finite delay by *initial* algebras of the process equations for delay. This can be viewed as a first step in representing *fairness* in presheaf semantics. We give a concrete representation of the presheaf model as a category of *generalised synchronisation trees* and show that it is coreflective in a category of *generalised transition systems*, which are a special case of the general transition systems of Hennessy and Stirling. The open map bisimulation is shown to coincide with the *extended bisimulation* of Hennessy and Stirling. Finally we formulate Milners operational semantics of SCCS with finite delay in terms of generalised transition systems and prove that the presheaf semantics is *fully abstract* with respect to extended bisimulation.

1 Introduction

When reasoning about and describing the behaviour of concurrent agents it is often the case that some infinite computations are considered *unfair* and consequently ruled out as being *inadmissible*. An economical way of studying this situation was proposed by Milner in [17] showing how to express a fair parallel composition in his calculus SCCS (*synchronous* CCS) by adding a *finite, but unbounded* delay operator. Syntactically the finite delay of an agent t is written ϵt. The agent ϵt can perform an unbounded number of 1-actions $\epsilon t \xrightarrow{1} \epsilon t$ (delays) but *must eventually* perform an action $\epsilon t \xrightarrow{a} t'$ if t can perform an action $t \xrightarrow{a} t'$ or *stop* if t cannot perform any actions. In other words, its actions are the same as for (the possibly infinite delay) $\delta t = \mathrm{rec}\, x.(1 : x + t)$, except that infinite unfolding of the recursion is not allowed.

[1] BRICS, Basic Research in Computer Science, Centre of the Danish National Research Foundation.
[2] This work was initiated during a stay at LFCS, University of Edinburgh, Scotland.

Conclusion

- labelled execution systems have very rich branching structure
- must be suffix closed and fusion closed to be well behaved
- must be not limit closed or indeterminately terminating to be justified

Future work

- abstraction
- application
- stratification

Questions?