1 Introduction

The language of operator nets is a graphical programming model whose programs are given mathematical semantics under a particular data algebra and particular sets of inputs. Though operator nets appear similar to dataflow graphs, there are significant differences between the two models. In particular, operator nets may imply a variety of operational semantics.

An operator net consists of a set of vertices and a set of directed edges corresponding to infinite sequences of data values from some underlying set. A program in the language consists of a set of equations that relate the output edge of a vertex to a function applied to the input edges of the vertex. These equations themselves can be considered a language: the functional programming language Lucid [WA85].

Operator nets intuitively reflect the concept of data flowing through a network. Since there is no notion of explicit sequencing or flow of control in the language, computations are naturally distributed. Because of this, it is possible to define an operational semantics for operator nets that corresponds intuitively to communication flow in a distributed system. In such a model, each edge is considered as a communication channel that carries message values from one vertex (process) to another.

An operational semantics for deterministic operator nets has previously been specified [GM87a]. When one considers operator nets that behave nondeterministically several problems arise, one of which is the importance of compositionality. A semantics where an operator net is modeled as a relation consisting of pairs of input histories and sets of output histories, is not sufficient to obtain a compositional model, as one can not distinguish between relations which, although they have identical input/output pairs, achieve this output in a different way. The following sections summarize what will be in the full paper.
2 Operational Semantics for Operator Nets

The operational semantics for distributed systems is expressed in terms of the 
histories of an operator net. A history is similar to the notion of a global 
state [CL85]. Each edge of an operator net is associated with a possibly-infinite 
history sequence of datons, some of which may be undefined (⊥). The set of 
possible histories for a net is determined by the possible computations that can 
occur. A computation is a sequence of events where each event is a function from 
a history to a new history resulting from processing occurring at a particular 
vertex.

The operational semantics for operator nets extends the language of operator 
ets by adding two new operators. The delay operator maps a sequence into a 
sequence with interspersed histons denoting timing delays. This operator allows 
us to specify nondeterministic operator nets [GM87b]. The second new operator 
is a buffer. The operational semantics of this operator is that it takes as input 
a sequence \( x \) and a constant \( y \) and it buffers its input from \( x \) until \( y \) values have 
been defined.

We define the operational semantics for operator nets in terms of the input/output behavior of a net. If the set of histories restricted to the input 
and output arcs are identical for two operator nets, we say that these nets are 
equivalent in terms of the operational semantics. In the full paper, we give 
formal definitions for the set of histories for a net and equivalence in terms of 
the operational semantics. This equivalence is used in the following section to 
demonstrate compositionality of the semantics.

3 Compositionality

Proving a semantics compositional means that if you have two nets that have 
the same semantics, then plugging these nets into any context should result in 
an identical semantics as well. More formally: Let \([n]_o\) denote the operational 
semantics associated with \( n \). Let \( n_1 \) and \( n_2 \) be two operator nets with \( x \) input 
edges. Let \( S \) be the set of all operator nets that contain a vertex \( v \) with \( x \) 
input edges. Let \( C^o \) be the set of all contexts for nets with \( x \) inputs, i.e. the 
set resulting from deleting the vertex \( v \) with all its adjacent edges from each 
operator net in set \( S \). Let \( C(n) \) denote the operator net resulting from putting 
the net \( n \) into the context \( C \). Then the semantics is compositional if

\[
[n_1]_o \equiv [n_2]_o \quad \iff \quad \forall C \in C^o \quad [C(n_1)]_o \equiv [C(n_2)]_o
\]

Brock & Ackerman first showed that a semantics based on history relations 
is not compositional [BA81]. They presented two operator nets, where in the 
one net the output is channeled through a buffer, where in the other net it is 
not. The buffer insures that the first output is only produced after its second
output has been determined. We propose an operational semantics based on
sets of valid histories, which would map these two nets to two distinct sets.

Proving our semantics compositional is really a first step towards proving it
to be fully abstract with respect to the denotational semantics. Informally a
semantics is fully abstract if it contains the minimal amount of detail in order
to make it compositional. More formally: Let \([n]_d\) denote the denotational
semantics associated with \(n\), then the operational semantics is fully abstract if

\[ [n_1]_d \equiv [n_2]_d \iff \forall C \in C^s \ [C(n_1)]_d \equiv [C(n_2)]_d \]

The proof of the compositionality of our operational semantics will imply (\(\rightarrow\)),
which is called adequacy [PS89]. The importance of full abstraction is stated in
[JK88]:

"When designing modular verification methods, in which the verifica-
tion of a network can be split into independent subverifications of
it's components, a fully abstract model indicates what aspects of a
network must be specified".

Many compositional models have been proposed, but only a few have been
proven fully abstract. In [BA81], a semantics based on scenario sets is suggested,
which is compositional but not fully abstract. In [Kok87], Kok presented a
fully abstract semantics based on finite-word-vector functions and in [PS89]
Panangaden & Shanbhogue presented a fully abstract semantics based on traces
and residuals. We expect to prove our semantics fully abstract in a subsequent
paper.

References

computation. Formalization of Programming Concepts, Springer Ver-

global states of distributed systems. ACM Transactions on Computer

[GM87a] J.I. Glasgow and G.H. MacEwen. A computational model for dis-
tributed systems using operator nets. In Conference on Parallel Ar-

[GM87b] J.I. Glasgow and G.H. MacEwen. Reasoning about knowledge in mul-
tilevel secure distributed systems. In Proceedings of the IEEE Sym-

publication, 1988.
