

Modal Logic and Lucid Specifications

J. Sanders

Programming Research Group
University of Oxford, Oxford, U.K.

D.B. Skillicorn

Department of Computing and Information Science
Queen's University, Kingston, Canada

Abstract

We outline a method of building modal logics whose models are transition systems. Thus they can be used to describe many popular distributed systems. Verification proofs and proofs of other properties of interest (security, realtime behaviour) may be easier in the modal logic setting. There is also some potential for describing properties usually considered too operational to be treated within an abstract formal system.

1 Introduction

Techniques for constructing sequential, imperative programs that satisfy specifications are relatively well understood. The early work in the area envisaged programs as being developed using the skills of the programmer and then, in a quite separate step, being shown to meet their specifications. This idea has gradually given way to a more calculational approach, in which specifications are refined from some initial, possibly non-executable form, into an executable form that presumably satisfies other properties of interest such as efficiency. Successful refinements preserve properties of the specification, so that the final version satisfies the specification by construction. Thus the emphasis has shifted from *post hoc* verification to *property preserving transformations*. This second approach works even better with functional programs.

We consider a specification to be divided into two parts: the first part, the functional specification, is transformed to an implementation. Hence no proof obligation remains. In several kinds of systems, there is a second part of the specification that is logically factored out because it is more subject to change than the functional part. For example, it may contain details of the timing requirements of the system, degree of parallelism, or security policy. The advantage of the calculational style is that this second kind of property need only be proven once, at what ever stage of the refinement it

- seems easiest. This avoids the common problem of proving that an abstract design satisfies a specification, and then that an implementation also does.

In constructing parallel implementations, new problems arise. This is mostly because the formalisms are not so well behaved. If the calculational approach is to succeed, a formal system must be abstract enough to allow us to state specifications and reason about them without considering operational details; but at the same time detailed enough to allow us to refine specifications into implementations that are close to the physical system on which they execute.

Finding systems with these characteristics is not easy. For example, saying anything very strong about a parallel implementation usually requires some kind of fairness property. Fairness properties are usually considered to be rather operational notions and it seems unfortunate that they should necessarily be present in reasoning about specifications. Systems in which security or real-time properties are important are particularly difficult because these properties depend on very detailed aspects of the implementation.

Some progress has been made with finding such formalisms. Some of the best known are Unity [2], Back's *action systems* [1], various kinds of transitions systems [7], temporal logics [5], and of course Lucid or operator nets [4, 3, 6]. All of these systems allows some degree of reasoning about computation. The non-logic based approaches are good at expressing behavioural properties of systems but make it hard to express and prove logical properties. Temporal logics handle logical properties well, but do not do so well with behavioural properties. What we have tried to do is integrate what appear to be the best of both kind of approaches into a modal logic, in which logical properties can be expressed, which has as its models Kripke structures that correspond to concurrent transition systems.

2 Modal Logics from Systems

We show how to build a modal logic based on any kind of model of computation that has atomic operations or transitions. Let $pre(x)$ denote the preconditions for the occurrence of operation x and $post(x)$ the postconditions that hold immediately after the occurrence of x . We say that any two atomic operations x and y are consistent if, whenever there is a state in which both x and y are enabled (that is, $pre(x)$ and $pre(y)$)

$$post(x) \rightarrow pre(y)$$

and

$$post(y) \rightarrow pre(x)$$

(\rightarrow means logical implication.) Consistent operations are those which are independent of each other in some sense, so that the occurrence of one does not affect the possible occurrence of the other.

We use the notation of Hughes and Cresswell, in which L represents the modal necessity operator and M the modal possibility operator. We construct a modal logic with the following axioms:

1. All axioms of modal logic K .
2. For each operation x , an axiom

$$pre(x) \rightarrow M(post(x))$$

3. For each pair of consistent transitions x and y an axiom

$$\begin{aligned} pre(x) \wedge pre(y) &\rightarrow L(post(x) \rightarrow pre(y)) \wedge \\ &L(post(y) \rightarrow pre(x)) \wedge \\ &M^2(post(x) \wedge post(y)) \end{aligned}$$

4. For each triple x , y , and z of consistent transitions, an axiom

$$pre(x) \wedge pre(y) \wedge pre(z) \rightarrow M^3(post(x) \wedge post(y) \wedge post(z))$$

Such a logic is constructed so that its Kripke structures look like Stark's concurrent transition systems [7]. The intuition behind the third and fourth axioms is as follows: if two transitions are consistent then the occurrence of one cannot affect the occurrence of the other; hence if one occurs then the other must remain 'enabled'; and if both occur, then it cannot be possible afterwards to distinguish the order in which they occurred (and eventually we want to think of them as occurring simultaneously). The fourth axiom extends this to three consistent transitions; it is required because pairwise use of the third axiom is not strong enough to guarantee a common state in which all three transitions have occurred; Stark calls this a cube axiom.

We say that two transitions x and y compose if $post(x) \rightarrow pre(y)$ and we write composition as concatenation.

Lemma 1 $pre(xy) \leftrightarrow pre(x)$ and $post(xy) \leftrightarrow post(y)$.

Lemma 2 *Composition of transitions is associative.*

Two transitions x and yz are consistent if x and y are consistent and x and z are consistent. We can easily prove the following theorem:

Theorem 3 *If x and yz are consistent then*

$$\begin{aligned} pre(x) \wedge pre(yz) &\rightarrow L^2(post(yz) \rightarrow pre(x)) \wedge \\ &L(post(x) \rightarrow pre(yz)) \wedge \\ &M^3(post(x) \wedge post(yz)) \end{aligned}$$

This can be generalized to

Theorem 4 *If $x = x_1x_2\dots x_n$ and $y = y_1y_2\dots y_m$ are consistent then*

$$\begin{aligned} & pre(x) \wedge pre(y) \rightarrow L^n(post(x) \rightarrow pre(y)) \wedge \\ & L^m(post(y) \rightarrow pre(x)) \wedge \\ & M^{n+m}(post(x) \wedge post(y)) \end{aligned}$$

This logic describes what might be called minimal parallelism systems, because only one transition can occur at a given time. There is no way to model the occurrence of multiple transitions simultaneously. We can build a similar logic which corresponds to a maximal parallelism view of the computation in the following way: if t and u are consistent transitions define $t \sim u$ if and only if $post(t) = post(u)$.

Lemma 5 *\sim is an equivalence relation.*

We now define a logic as before except that for each equivalence class of transitions $[t]$ we add an axiom

$$pre([t]) \rightarrow M(post([t]))$$

for each pair of consistent composite transitions $[t]$ and $[u]$ we add an axiom

$$\begin{aligned} & pre[t] \wedge pre[u] \rightarrow L(post[u] \rightarrow pre[t]) \wedge \\ & L(post[t] \rightarrow pre[u]) \wedge \\ & M^2(post[t] \wedge post[u]) \end{aligned}$$

and the obvious axiom for any three consistent composite transitions.

This logic is essentially a quotient of the first logic by the equivalence relation \sim . It allows maximal concurrency because the interleaving of transitions from $[t]$ and $[u]$ is not specified – hence it can model the simultaneous occurrence of these transitions.

3 Examples

We show what this kind of logic looks like for some common concurrent systems. Let us begin with the Petri net shown in Figure 1. The added axioms are:

$$\begin{aligned} X : a \wedge \neg b & \rightarrow M(b \wedge \neg a) \\ Y : b \wedge \neg c & \rightarrow M(c \wedge \neg b) \\ Z : c \wedge \neg d & \rightarrow M(d \wedge \neg c) \\ T : d \wedge \neg a & \rightarrow M(a \wedge \neg d) \end{aligned}$$

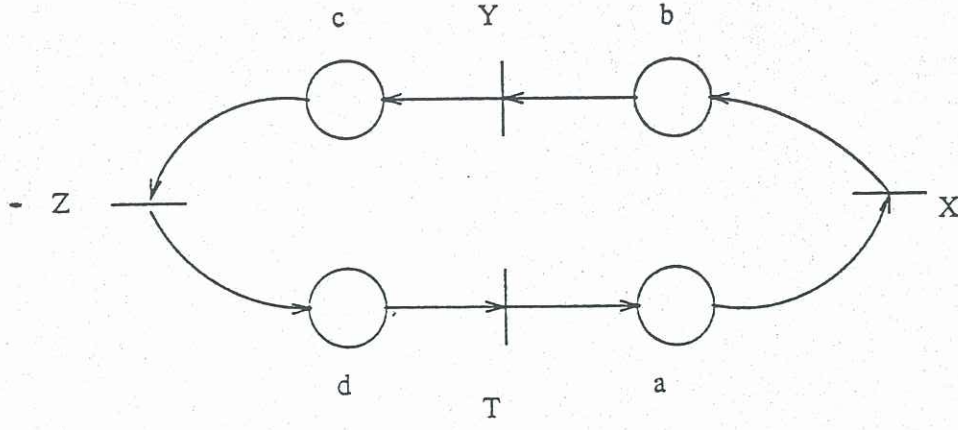


Figure 1: Simple Petri Net

for the single transitions (where a is considered to be the proposition “there is a token on place a ”), and

$$\begin{aligned}
 XZ : & a \wedge \neg b \wedge c \wedge \neg d \text{ —} \\
 & L(\neg a \wedge b \rightarrow c \wedge \neg d) \wedge \\
 & L(\neg c \wedge d \rightarrow a \wedge \neg b) \wedge \\
 & M^2(\neg a \wedge b \wedge \neg c \wedge d)
 \end{aligned}$$

$$\begin{aligned}
 YT : & b \wedge \neg c \wedge d \wedge \neg a \text{ —} \\
 & L(\neg b \wedge c \rightarrow d \wedge \neg a) \wedge \\
 & L(\neg d \wedge a \rightarrow b \wedge \neg c) \wedge \\
 & M^2(\neg b \wedge c \wedge \neg d \wedge a)
 \end{aligned}$$

The model for this logic is shown in Figure 2. It is easy to see that the Kripke structure divides into four parts, depending on how many tokens are present in the Petri net. Notice that when four tokens are present no transitions are possible because none of the preconditions are satisfied.

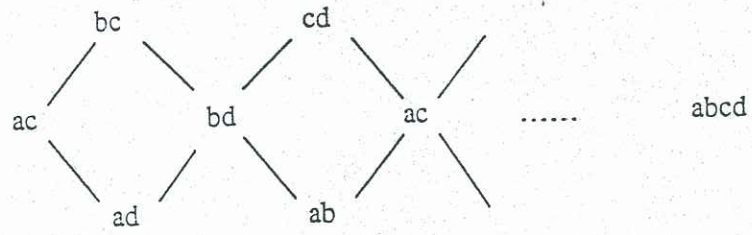
The same thing can be done with Lucid programs/operator nets. Consider the simple operator net in Figure 3, in which operator A is the operator *next* and the operator B is pointwise. Here we get three infinite families of axioms (assuming piped operation of the net):

$$\begin{aligned}
 a_i & \rightarrow M(a_{i+1}) \\
 a_{i+1} & \rightarrow M(b_i) \\
 b_i & \rightarrow M(c_i)
 \end{aligned}$$

There is an infinite set of axioms arising from the consistency of A , B , and the input operation of the form

$$a_{i+1} \wedge b_j \text{ — } L(b_i \rightarrow b_j) \wedge L(c_j \rightarrow a_{i+1}) \wedge M^2(c_j \wedge b_i)$$

a — b — c — d — a — ...



abc — abd — acd — bcd — abc — ...

Figure 2: Model for Petri Net

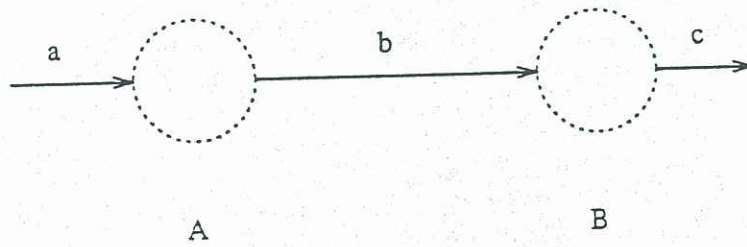


Figure 3: A Simple Operator Net

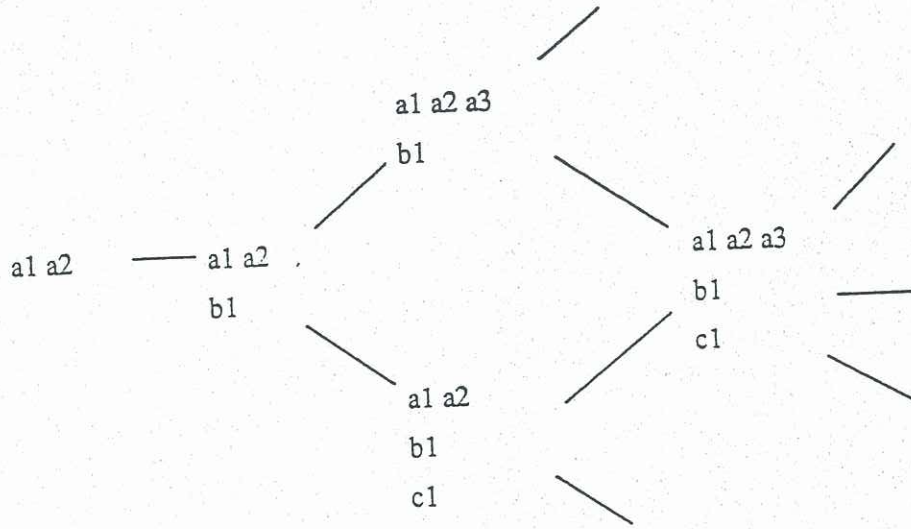


Figure 4: Model for Operator Net

This has models with the obvious structure in which it does not matter in which order operations occur unless there is a data dependency between them. A small part of one is shown in Figure 4.

4 Realtime Systems

One of the difficulties with realtime systems is that their behaviour is sensitive to extremely operational considerations. We have already seen how our logic can be used to model systems that are sequential and those that have maximal parallelism. We would like to be able to determine the effect of other limited parallelism on specifications.

Suppose we begin with a sequential system and increase the available parallelism to two. In terms of the transition system this means that all commuting diamonds of the form shown in Figure 5 can be replaced by a single transition across the diagonal representing the simultaneous occurrence of the component transitions. Notice that the transitions must have been consistent for this to be possible.

In the logic this corresponds to replacing an axiom of the form

$$\begin{aligned} pre(x) \wedge pre(y) \rightarrow L(post(x) \rightarrow pre(y)) \wedge \\ L(post(y) \rightarrow pre(x)) \wedge \\ M^2(post(x) \wedge post(y)) \end{aligned}$$

with one of the form

$$pre(x) \wedge pre(y) \rightarrow \bar{M}(post(x) \wedge post(y))$$

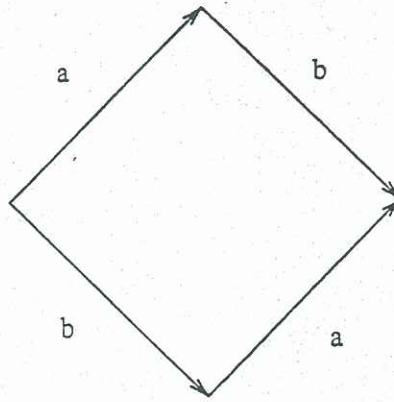


Figure 5: Commuting Transitions

In general, the existence of n -way parallelism requires replacing M^{2n-2} operators by new \bar{M} operators. Thus we can “factor in” information about physical parallelism at a late stage in the specification. There is some hope that it is possible to determine which proofs about specifications are invariant under substitutions of this kind. We are exploring using indexed categories to capture the relationship of these logics to one another.

5 Conclusions

We have shown how to build modal logics from different kinds of distributed system models with atomic events. Such modal logics provide a different way at looking at these systems; and it may sometimes be easier to prove properties about them working in extensions of these modal logics rather than proving some properties at an abstract level and others at an implementation level.

References

- [1] R.J.R. Back. A method for refining atomicity in parallel algorithms. In *PARLE89 Parallel Architectures and Languages Europe*, Springer Lecture Notes in Computer Science 366, pages 199–216, June 1989.
- [2] K.M. Chandy and J. Misra. *Parallel Program Design: A Foundation*. Addison-Wesley, 1988.

- [3] J.I. Glasgow and G.H. MacEwen. Lucid: A specification language for distributed systems. *Verification Workshop III*, February 1985.
- [4] J.I. Glasgow and G.H. MacEwen. The development and proof of a formal specification for a multilevel secure system. *Transactions on Computer Systems*, 5, No.2:151-184, May 1987.
- [5] A. Pnueli. The temporal logic of concurrent programs. *Theoretical Computer Science*, 13:45-60, 1981.
- [6] D.B. Skillicorn and J.I. Glasgow. Real time specification using Lucid. *IEEE Transactions on Software Engineering*, pages 221-229, February 1989.
- [7] E. Stark. Concurrent transition systems. *Theoretical Computer Science*, to appear 1989.