Modalities in Analysis and Verification

Mads Dam*
Swedish Institute of Computer Science

January 1, 1996

There is currently a strong interest in programming languages and models which attempt to integrate concurrency and distribution with structuring and abstraction mechanisms normally seen in sequential settings. Examples are manifold: concurrent and distributed extensions of ML such as Facile or CML, Erlang, concurrent constraint programming languages such as AKL or OZ, PICT, and the Actor programming model. Also in the object-oriented community strong trends in the direction of treating concurrency and distribution as "first-class citizens" can be discerned, viz. OMG's work on CORBA, Microsoft's OLE, or Ericsson's DELOS language. Even though a strict scrutiny of some of these examples might not invariably warrant this terminology, we refer to systems in models or languages like the above as "higher order concurrent systems". Common to them is that they provide processes or object as concurrent or distributed entities communicating along named channels along which information is passed, in the form of e.g. values, channels, references to other entities, or program scripts. The behavioral encapsulation of such entities, whether it be used for automated program analysis, for verification, or for dynamic request-provider matching must reflect their communication capabilities, for instance by means of type systems extended with modal or temporal information. However, such an integration of functional (in the general sense of "parametric") and reactive behaviour poses severe, and very interesting, challenges to the currently established technologies for program verification and analysis. There are two basic difficulties:

1. Because system components can execute in parallel and interfere there is no simple and obvious way in which the behaviour of a composite system can be inferred from that of its parts. This in contrast to the situation for sequential languages where this is the paradigm for e.g. denotational or operational semantics, and type systems.

2. Because processes can be parametric upon general subcomponents they can not in general be represented as finite state transition systems. Rather,

*Work supported by ESPRIT BRA project 6454 "CONFER". Adress: SICS, Box 1263, S-164 28 Kista, Sweden. Email: mfd@sics.se. Tel. +46 8 752 1549. Fax +46 8 751 7230.

they produce dynamic process networks where the efficient runtime creation (and killing) of subprocesses is central. Also formalisms like the π-calculus, even though it does not pass processes around, rely crucially on dynamic process creation for its expressive power. As a consequence the substantial amount of technology developed for communicating finite state machines is not readily applicable for these kinds of applications.

Of course these points are crucially dependent on the range of properties one is interested in. At one end it is easy to devise type systems in which no temporal information is captured at all, like the extensions of ML's type system in Facile or CML. For more general program properties there seems to be two viable directions:

1. Automated approaches, using e.g. symbolic or abstract interpretation-based techniques to extract finite-state (and hence analysable) approximations from models which are inherently infinite-state.

2. Axiomatic approaches using meta-theoretical reasoning, temporal reasoning, or compositional techniques.

In the former direction, symbolic techniques have been used in the context of model checking and bisimulation checking in the context of static process network fragments of value-passing process calculi and the π-calculus [8, 9, 13]. However, these approaches by themselves have serious difficulties in dealing with dynamic process creation. One way of overcoming this problem is to use abstraction to derive finite-state approximations of underlying infinite-state models. For instance, Nielson and Nielson [19] derives a CCS-like finite-state process representing the causal aspects of general CML programs using a type and effect system. Such finite-state representations can then be analysed in various ways, for instance for communication topology or channel utilisation (c.f. [20]). Abstract interpretation techniques have been very successful in for instance sharply reducing state space requirements for problems with very large, but finite, state spaces [5]. An important property is to preserve validity under abstraction. For instance, in [5] validity in a universal fragment of CTL* is preserved. Other authors have considered other fragments [6, 11]. Also positive and negative interpretations can be combined to provide sound, but incomplete interpretations of richer logics, as in [16, 7].

At the end of the day, however, if exact and very generally applicable analyses are called for then automated approaches are in our opinion likely to fall short of the target. As examples of the type of properties we ultimately wish to verify, consider the following kinds of properties of a process P:

- No matter what agent P receives on channel a, if that agent ever tries to access x then a notification will be emitted on channel b.

- P will eventually output a reference to an α list on channel c.

- In any environment, if message m is ever emitted along *out* then m was earlier received along *in*.

While often considered impractical it is conceivable that such properties can in principle be dealt with using proof-based methods. For instance, the operational semantics of higher-order concurrent programs is often not too difficult to embed in a framework for formalised mathematics (c.f. [18]). However, such an approach is unlikely to provide by itself the kind of support needed for any practical applications.

Ultimately, for systems and properties of this generality we believe that the bottom line is the difficult problem of compositionality: How can functional and temporal properties of a process $P(x, y)$ be inferred, automatically or manually, from those of its components x and y in the setting of dynamic process networks and higher-order communication? In the area of temporal logic the problem of compositionality has occupied researchers for a long time, and many techniques have been proposed, including rely-guarantee pairs (c.f. [22, 1]), history variables (c.f. [21]), quotienting (c.f. [4, 3]), reduction (c.f. [17]), phantom moves, simulations, edge propositions, or quiescent traces (c.f. [2, 12, 15, 24]), to name but a few. These techniques, however, give only partial and ad-hoc solutions in that they work only for particular concurrency primitives, static process networks and, most often, linear time logic only. Recently, in [10], we presented an approach to compositional verification which seems to overcome many of the difficulties of earlier work in this direction, and which seemed to suggest that maybe the problem of providing general, sound, and powerful *reasoning tools* for compositional verification may not be quite as hard as it has been considered to be in the past [1]. Drawing inspiration from earlier work by Stirling [23] and Winskel [25] we showed how a compositional proof system for establishing modal μ-calculus properties of *general* CCS processes could be built in a systematic fashion. Since we did not limit attention to the finite state fragment of CCS dynamic process networks could easily be represented, and we gave a simple example of a dynamic process network to show the power of our approach. Compositional proof system for CCS vs the modal μ-calculus were considered earlier, for instance by Andersen et al (c.f. [3]), but applicable to finite-state problems only.

Much work remains to be done before we can truly claim that verification of higher-order concurrent systems is feasible. We need to experiment more with proof systems such as that of [10] to figure out its scope in practice as well as theory, to extend the work to parameter-passing languages, and to find good ways of embedding automated approches into a basically interactive, proof system based approach. Moreover, other approaches, including symbolic ones, to dynamic process network verification needs to be investigated too, for instance along the lines of [14]. However, we see the general area of open systems verification based on quite generally applicable, interactive, proof-oriented methods as having quite considerable long-term potential in helping to bridge the quite acute and growing gap between systems development practice and formal methods.

[1] This is not the same as suggesting that compositional verification *in itself* is easy, which it is not.

References

[1] M. Abadi and G. Plotkin. A logical view of composition and refinement. In *Proc. Eighteenth Ann. ACM Symp. Principles of Programming Languages*, pages 323–332, 1991.

[2] K. R. Abrahamson. Modal logic of concurrent nondeterministic programs. Lecture NOtes in Computer Science, pages 21–33, 1979.

[3] H. Andersen, C. Stirling, and G. Winskel. A compositional proof system for the modal μ-calculus. In Proc. LICS'94, 1994.

[4] H. Andersen and G. Winskel. Compositional checking of satisfaction. *Formal methods in System Design*, 1(4), 1992.

[5] E. M. Clarke, O. Grümberg, and D. E. Long. Model checking and abstraction. *ACM Transactions on Programming Languages and Systems*, 16(5):1512–1542, 1994.

[6] R. Cleaveland, S. Purushothaman Iyer, and D. Yankelevitch. Abstractions for preserving all CTL* formulae. Technical Report, N. C. State University, 1994.

[7] R. Cridlig. Semantic analysis of shared-memory concurrent languages using abstract model checking. In *Proc. PEPM'95*, 1995.

[8] M. Dam. Model checking mobile processes (full version). SICS report RR 94:1, 1994. Prel. version appeared in Proc. Concur'93, LNCS 715, pp. 22–36.

[9] M. Dam. On the decidability of process equivalences for the π-calculus. Submitted for publication, 1994.

[10] M. Dam. Compositional proof systems for model checking infinite state processes. In *Proc. CONCUR'95,* Lecture Notes in Computer Science, 962:12–26, 1995.

[11] D. Dams, O. Grumberg, and R. Gerth. Abstract inetrpretation of reactive systems: Abstractions preserving $\forall$CTL*, $\exists$CTL* and CTL*. In *Proc. IFIP WG2.1/2.2/2.3 Working Conference on Programming Concepts, Methods, and Calculi (PROCOMET)*, 1994.

[12] R. Kuiper H. Barringer and A. Pnueli. Now you may compose temporal logic specification. In *Proc. ACM Symp. Theory of Computing*, pages 51–63, 1984.

[13] M. Hennessy and H. Lin. Symbolic bisimulations. Dept. of Computer Science, University of Sussex, Report 1/92, 1992.

[14] H. Hungar. Local model checking for parallel compositions of context-free processes. In Proc. CONCUR'94, *Lecture Notes in Computer Science*, 836:114–128, 1994.

[15] B. Jonsson. Compositional specification and verification of distributed systems. *ACM Transactions on Programming Languages and Systems*, 16(2):259–303, 1994.

[16] P. Kelb. Model checking and abstraction: A framework approximating both truth and failure information. Tech. Rep. Univ. Oldenburg, 1994.

[17] K. G. Larsen and X. Liu. Compositionality through an operational semantics of contexts. *Journal of Logic and Computation*, 1:761–795, 1991.

[18] Tom F. Melham. A mechanized theory of the π-calculus in HOL. *Nordic Journal of Computing*, 1(1):50–76, 1994.

[19] F. Nielson and H. R. Nielson. From CML to process algebras. In *Proc. CONCUR'93,* Lecture Notes in Computer Science, 715, 1993.

[20] F. Nielson and H. R. Nielson. Higher-order concurrent programs with finite communication topology. In *Proc. POPL'94*, pages 84–97, 1994.

[21] N. Soundarajan. A proof technique for parallel programs. *Theoretical Computer Science*, 31:13–29, 1984.

[22] E. W. Stark. A proof technique for rely/guarantee properties. *Proc. ESOP'86, Lecture Notes in Computer Science*, 206:369–391, 1986.

[23] C. Stirling. Modal logics for communicating systems. *Theoretical Computer Science*, 49:311–347, 1987.

[24] D. Gries Van Nguyen and S. Owicki. A modal and temporal proof system for networks of processes. Proc. 12th Annual ACM Symposium on Principles of Programming Languages, pages 121–131, 1985.

[25] G. Winskel. A complete proof system for SCCS with modal assertions. *Lecture Notes in Computer Science*, 206:392–410, 1985.