A Causal Logic of Events in Formalized Computational Type Theory *

Mark Bickford[a] and Robert L. Constable[b]

[a] *ATC, NY*
[b] *Cornell University, Department of Computer Science*

Abstract

We provide a logic for distributed computing that has the explanatory and technical power of constructive logics of computation. In particular, we establish a proof technology that supports *correct-by-construction programming* based on the notion that concurrent processes can be extracted from proofs that specifications are achievable.

1 Introduction

1.1 Historical Context

Models of computation have been important in mathematics since Greek geometry of 300 BC, and perhaps for much longer. We call these models *formal* if they can be implemented by (idealized) machines. The sustained development of *formal computing models* and their implementation is much more recent, a 20th century activity with some foreshadowing by Babbage in the late 19th century. The main focus is digital computation, and it has been revolutionary—creating a computational aspect of every science and giving birth to a new discipline called computer science, starting with Turing in 1936 [Tur37]. Digital computation has even been proposed as a new foundation for physics [Hey02, Whe82, Whe89].

In the late 20th century, the Internet and other networks of machines made distributed computing a transformative global resource. Reasoning about networks required a new model of computation. The resulting model of distributed computation is enormously rich,

*This material is based upon work supported by the National Science Foundation under Grant No. 0208536.

and computer scientists are only beginning to create the concepts and tools to understand it deeply and exploit its potential in science, as well as in technology and commerce.

One of the critical challenges researchers have faced in understanding every model of computation is creating a declarative language that relates the dynamic nature of computation with the declarative basis of scientific theories. An illustrative example of this challenge already appears in Euclidean geometry.

Euclid's propositions and postulates are a mixture of constructions and declarative statements. For example, he says essentially "given two points, we can draw a line (segment) connecting them". He declares that in any triangle, the length of any two sides is greater than that of the remaining one. Corresponding to this is the problem of making a construction, namely "given two line segments whose combined length is greater than a third, construct a triangle with these segments as sides."

Euclidean geometry is a mixture of propositions, problems, postulates, and constructions. There are a finite number of basic postulates, and a finite number of atomic straight edge and compass construction methods (or schemes). This intuitive hybrid language sufficed for two thousand years. The geometric model of computing was far from formal, it was not even rigorous. Logicians then discovered how to make the declarative language rigorous, and eventually formal, using quantifiers, but "the quantifiers killed the constructions". That is, instead of saying given points A and B we can *construct* a line segment between them, Hilbert said, given points A and B, *there exists* a line segment connecting them. Symbolically,

$$\forall A, B : \mathit{Points}.\ \exists L : \mathit{Line}.L = [AB].$$

1.2 Computational Logic

It took a few decades to sort out a declarative language with computational meaning. L.E.J. Brouwer showed the way, and in due course a computational (or constructive) interpretation of formal logic was achieved, called the Brouwer, Kolmogorov, Heyting (BKH) interpretation. We will use this below. See the book [GTL89] for the BKH interpretation.

This computational interpretation of the predicate calculus restored the balance between computation and assertion, and it became the basis for *logics of computation* that applied well to functional and procedural programs as demonstrated by deBruijn, Scott, Martin-Löf, Girard, Constable, Huet, Coquand, Paulin and others. These logics have enabled a very potent proof technology with applications both to mathematics and to software development. One of the key ideas in the logic of computation is the notion of *proofs-as-programs* [BC85], which will be of central concern here.

1.3 The Logical Challenge of Distributed Computing

The issue before us now is to find an adequate logic for distributed computing that has the explanatory and technical power of constructive logics of computation. In particular, we aspire to a proof technology that supports *correct-by-construction programming* based on

the notion that concurrent processes can be extracted from proofs that specifications are achievable. The goal has been elusive until now.

Equally elusive in the case of networked computation is finding a declarative language for specifying distributed computing problems at very high levels of abstraction. Languages such as TLA+ [Lam03] describe computation at the level of execution models, and even at their most general, such models are not sufficiently abstract to apply well in all the circumstances we have in mind.

We present a very abstract specification language which can be understood without direct reference to a computational model. As in the case of the language of Computational Type Theory (CTT [ABC$^+$, CAB$^+$86a, Con02]), there is a computation model behind it that is manifest in rules for reasoning. Likewise, in the setting of our computational theory of typed events (CTT-E), the inference rules will exploit the underlying computational interpretation. The computational interpretation through the inference rules is sufficiently strong that from a proof that a specification is achievable, we can automatically exact an executable distributed system.

We have formalized the logic and its implementation in the Nuprl system [ACE$^+$00, CAB$^+$86a] and the ScoRes distributed runtime environment [BG05] so that the creative steps of distributed system design and verification can be undertaken at a high logical level, and the detailed system programming can automated by the *extractor/compiler*. The extractor/compiler contains a large amount of detailed systems programming knowledge that is automatically applied. The designer can ignore many of these details. However, in a proof that Nuprl and ScoRes are correct, this knowledge must be made explicit. This has not yet been accomplished. Eventually, it could be done using formalizations of Java and of virtual machine models like JVM of the kind being formalized in Isabelle, HOL [GM93, NPW02, PN90, Pau88]. However, our focus is on the design and verification stage, and on the contributions possible at this level to computer science and to computing technology and software development.

Another aspect of our work that we only touch on briefly is the nature of formal interactive proof using the Nuprl 5 Logical Programming Environment. The entire theory of event structures on communication graphs has been formalized in Nuprl 5 by Mark Bickford and made available at the Nuprl web site www.nuprl.org. This theory contains over 2,500 definitions and theorems and is completely formally checked. It is a large knowledge base for understanding distributed computing at a fine level of detail.

The automated reasoning techniques implemented in the course of this formalization and supported by Stuart Allen and Richard Eaton, as well, represent a significant step in the implementation of the process of understanding distributed systems and designing protocols for communication, control, and security. This work is part of the long tradition begun by Newell, Simon, and Shaw [NSS57] of automating reasoning. Taken in its full extent, from pure mathematics to the verification of deployed systems, such work is one of the enduring contributions of computer science to intellectual history.

1.4 Formulas and problems

Here is how we interpret the statements of a typed predicate logic. For atomic predicates to **assert** or **solve** $P(t_1, \cdots, t_n)$ means to provide a proof or a construction $p(t_1, \cdots, t_n)$.

If P, Q are problem statements (predicate formulas), then to assert

$P \,\&\, Q$ means to find **proofs** or **constructions** p and q for P, Q respectively.

$P \vee Q$ means to find a proof or construction p for P and **mark it** as applying to P or to find a proof or construction q for Q and mark it as apply to Q.

$P \Rightarrow Q$ means to find an **effective procedure** f that takes a proof or construction p for P and computes $f(p)$ a proof or construction for Q.

$\neg P$ means that there is no proof or construction for P.

$\forall x\!:\!A.P$ means that there is an **effective procedure** f that takes any element of type A, say a, and computes a proof or construction $f(a)$ for $P[a/x]$.

$\exists x\!:\!A.P$ means that **we can construct an object** a of type A and find a proof or construction p_a of $P[a/x]$, taken together, $< a, p_a >$ solves this problem or proves this formula.

2 Event Systems

2.1 General

Our theory is designed to account for the behavior of a wide variety of systems, from interacting computers on the Internet to interacting components in a single computer or in a brain. It can also describe cause and effect behavior in physical systems on the scale of galaxies or subatomic particles. The right theory can be a unifying force in the study of computation in all its many forms. Our theory is another step toward a comprehensive account of distributed computing in its broadest sense. It is heavily influenced by the insights of Lamport[Lam78] and Winskel [Win80, Win89].

2.1.1 Events

Events are the atomic units of the theory. They are the occurrences of atomic actions in space/time. Although they have duration, we don't speak of it, considering them to be instantaneous moments at which "things happen". These events are *causally ordered*, e before e', denoted $e < e'$. As Lamport postulated, *causal order* is the structure of time.

We abstract away the duration of an event, which would be related to the physical time that the action requires. The structure of *event space* is determined by the organization of events into discrete *loci*, each a separate locus of actions through time at which events are sequentially ordered. The entities (locations) are separate; for example, they do not share state, they can be distinguished by messages. All actions take place at these locations (or by these entities). Actions are "located at these entities", and conversely, these entities are all (potentially) active. New entities can be created over time. At some locations, atomic actions produce random values. When seen as an entity, these loci can have *properties* such as physical coordinates. These are examples of *observable properties* of a *locus of action.*

2.1.2 Observables

We are interested in actions with observable results. Observables are known by *identifiers* and have *types.* For example, an observable might be a discrete value such as the spin of an electron, up or down; it might be the charge, positive or negative. We might observe the state of a device, on or off, or the values of a memory location, say an integer. The physical coordinates might be a quadruple of (computable) real numbers. The list of observables of an entity is its *state.*

Interaction among entities is determined by connections among them called *communication links* or *interaction channels.* These links form a discrete *interaction topology.* We allow that each entity is connected, perhaps by multiple links, to every other entity. The link structure can be dynamic.

Interaction is achieved by messages communicated on links. At each locus, every event can emit a signal (send a message). Sending a signal along a link to an entity will eventually cause that signal to be received by that entity, so the links are *reliable*, and reception cannot be blocked by the receiver. The action of detecting (or receiving) a signal is called an *external* event at the locus of reception. In addition, there can be *internal* events as the result of internal actions of the entity. All events are either external or internal, and either kind can emit a signal. The actions have names in the type Action.

Internal events can have *preconditions* or *guards* that determine the conditions under which they take place. The externally caused actions are not guarded; they happen whenever the signal arrives.

2.1.3 Computation and message automata

The universe is run by computation. It is the force that makes things happen. Computation is digital, built from discrete atomic actions. We can build the entire edifice on functional update of the state and of the message queues on the interaction links. The form of a state update is $s' := f(s, v)$ where s is the current state, v is a signal received or the value of an action and s' is the new state. We take arbitrary computable functions f as possible updating steps.

Ultimately we will describe the entities as automata, called *message automata.* Depending on the resolution at which we describe them, they can be as simple as atomic particles

or as complex as separate distributed systems, such as agents (human or robotic) or even large systems like a planet.

2.2 Event structures with order (EOrder)

It is possible to say a great deal without mentioning values, observables, and states; so we first axiomatize *event structures with order* but without values or states.

2.2.1 Signature of EOrder

The signature of these events requires two types, and two partial functions. The types are *discrete*, which means that their defining equalities are decidable. We assume the types are disjoint. We define $\mathbb{D}$ as $\{T : Type \mid \forall x, y : T.\ x = y \text{ in } T \vee \neg\ (x = y \text{ in } T)\}$, the large type of discrete types.

Events with order (EOrder)

E: $\mathbb{D}$
Loc:$\mathbb{D}$
pred?: $E \to E + Loc$
sender?: $E \to E + Unit$

The function *pred?* finds the predecessor event of e if e is not the first event at a locus or it returns the *location* if e is the first event. The *sender?(e)* value is the event that sent e if e is a *receive*, otherwise it is a unit. We can define the location of an event by tracing back the predecessors until the value of *pred* belongs to *Loc*. This is a kind of partial function on E. From *pred? and sender?* we can define these Boolean valued functions:

$$first(e) = \textit{if is\_left } (pred?(e)) \textit{ then true else false}$$
$$rcv?(e) = \textit{if is\_left } (sender?(e)) \textit{ then true else false}$$

The relation *is_left* applies to any disjoint union type $A + B$ and decides whether an element is in the left or right disjunct (see Naive Computational Type Theory [Con02]). We can "squeeze" considerable information out of the two functions *pred?* and *sender?*. In addition to *first* and *rcv?*, we can define the order relation

$$pred!(e, e') == (\neg\ first(e') \Rightarrow e = pred?(e')) \vee e = sender(e').$$

We will axiomatize this as a strongly well-founded order relation.

The transitive closure of *pred!* is Lamport's *causal order* relation denoted $e < e'$. We can prove that it is also strongly well-founded and decidable; first we define it.

The nth power of relation R on type T, is defined as

$xR^o y$ iff $x = y$ *in* T
$xR^n y$ iff $\exists z : T.\ xRz\ \&\ zR^{n-1}y$

The *transitive closure* of R is defined as xR^*y iff $\exists n : \mathbb{N}^+ \cdot (xR^n y)$.

Causal order is x *pred*$!^*y$, *abbreviated* $x < y$.

2.2.2 Axioms for event structures with order (EOrder)

There are only three axioms that constrain event systems with order beyond the typing constraints.

Axiom 1 *If event e emits a signal, then there is an event e' such that for any event e'' which receives this signal, $e'' = e'$ or $e'' < e'$.*

$$\forall e : E. \exists e' : E.\ \forall e'' : E.\ (\ rcv?(e'')\ \&\ sender?(e'') = e) \Rightarrow (e'' = e' \vee e'' < e)$$

Axiom 2 *The pred? function is injective.*

$$\forall e, e' : E.\ loc(e) = loc(e') \Rightarrow pred?(e) = pred?(e') \Rightarrow e = e'$$

Axiom 3 *The pred! relation is strongly well founded.*

$$\exists f : E \rightarrow \mathbb{N}.\ \forall e, e' : E. pred!(e, e') \Rightarrow f(e) < f(e')$$

To define f in Axiom 3 we arrange a linear "tour" of the event space. We can imagine that space as a subset of $\mathbb{N} \times \mathbb{N}$ where $\mathbb{N}$ numbers the locations and discrete time. Events happen as we examine them on this tour, so a receive can't happen until we activate the send. Local actions are linearly ordered at each location. Note, we need not make any further assumptions.

We can define the finite list of events before a given event at a location, namely

before(e) == if first(e) then[]
else *pred*?(e) append *before* (*pred*? (e))

Similarly, we can define the finite tree of all events *causally before* e, namely

prior(e) == if first(e) then[]
else if *rcv*?(e)
then $< e, prior(sender?(e)), prior(pred?(e)) >$
else $< e, prior(pred?(e)) >$

2.2.3 Properties of events with order

We can prove many interesting facts about events with order. The basis for many of the proofs is induction over causal order. We prove this by first demonstrating that causal order is strongly well founded.

Theorem 1 $\exists f : E \rightarrow \mathbb{N}.\ \forall e, e' : E.\ \ e < e' \Rightarrow f(e) < f(e')$

The argument is simple. Let $x \lhd y$ denote $pred!(x, y)$ and let $x \lhd^n y$ denote $pred!^n(x, y)$. Recall that $x \lhd^{n+1} y$ iff $\exists z : E.\ x \lhd z\ \&\ z \lhd^n y$. From Axiom 3 there is function $f_o : E \rightarrow \mathbb{N}$ such that $x \lhd y$ implies $f_o(x) < f_o(z)$. By induction on $\mathbb{N}$ we know that $f_o(z) < f_o(y)$. From this we have $f_o(x) < f_o(y)$. So the function f_o satisfies the theorem. The simple picture of the argument is

$$x \lhd z_1 \lhd z_2 \lhd \ldots \lhd z_n \lhd y$$

so

$$f_o(x) < f_o(z_1) < \ldots < f_o(z_n) < f_o(y).$$

We leave the proof of the following induction principle to the reader.

Theorem 2 $\forall P : E \rightarrow Prop.\ \forall e' : E.\ ((\forall e : E.\ e < e'.\ P(e)) \Rightarrow P(e')) \Rightarrow \forall e : E.\ P(e)$

Using induction we can prove that causal order is decidable.

Theorem 3 $\forall e, e' : E.\ e < e' \vee \neg\ (e < e')$

We need the lemma.

Theorem 4 $\forall e, e' : E.\ (e \lhd e' \vee \neg\ (e \lhd e'))$

This is trivial from the fact that $pred!(x, y)$ is defined using a decidable disjunction of decidable relations, recall

$$x \lhd y \text{ is } pred!(x, z)$$

and

$$pred!(x, y) = \neg\ first(y) \Rightarrow x = pred?(y) \vee x = sender?(y).$$

The local order given by $pred?$ is a total order. Define $x <_{loc} y$ is $x = pred?(y)$.

Theorem 5 $\forall x, y : E.\ \ (x <_{loc} y \vee x = y \vee y <_{loc} x)$

References

[ABC+] Stuart F. Allen, Mark Bickford, Robert Constable, Richard Eaton, Christoph Kreitz, Lori Lorigo, and Evan Moran. Innovations in computational type theory using Nuprl. To appear in Journal of Applied Logic 2006.

[Abr94] S. Abramsky. Proofs as processes. *Journal of Theoretical Computer Science*, 135(1):5–9, 1994.

[Abr95] Uri Abraham. On interprocess communication and the implementation of multi-writer atomic registers. *Journal of Theoretical Computer Science*, 149:257–298, 1995.

[Abr99] Uri Abraham. *Models for Concurrency*, volume 11 of *Algebra, Logic and Applications Series*. Gordon and Breach, 1999.

[Abr00] S. Abramsky. Process realizability. In F. L. Bauer and R. Steinbrüggen, editors, *Foundations of Secure Computation: Proceedings of the 1999 Marktoberdorf Summer School*, pages 167–180. IOS Press, 2000.

[ACE+00] Stuart Allen, Robert Constable, Richard Eaton, Christoph Kreitz, and Lori Lorigo. The Nuprl open logical environment. In David McAllester, editor, *Proceedings of the 17th International Conference on Automated Deduction*, volume 1831 of *Lecture Notes in Artificial Intelligence*, pages 170–176. Springer Verlag, 2000.

[AHR02] Myla Archer, Constance Heitmeyer, and Elvinia Riccobene. Proving invariants of I/O automata with TAME. *Automated Software Engineering*, 9(3):201–232, 2002.

[BB90] G. Berry and G. Boudol. The chemical abstract machine. In *Conference Record of the 17th Annual ACM Symposium on Principles of Programming Languages*, pages 81–94, 1990.

[BC85] J. L. Bates and Robert L. Constable. Proofs as programs. *ACM Transactions on Programming Languages and Systems*, 7(1):53–71, 1985.

[BCH+00] K. Birman, R. Constable, M. Hayden, J. Hickey, C. Kreitz, R. van Renesse, O. Rodeh, and W. Vogels. The Horus and Ensemble projects: Accomplishments and limitations. In *DARPA Information Survivability Conference and Exposition (DISCEX 2000)*, pages 149–161, Hilton Head, SC, 2000. IEEE Computer Society Press.

[BG03a] Andreas Blass and Yuri Gurevich. Abstract state machines capture parallel algorithms. *ACM Transactions on Computational Logic*, 4(4):578–651, 2003.

[BG03b] Andreas Blass and Yuri Gurevich. Algorithms: A quest for absolute definitions. *Bulletin of the European Association for Theoretical Computer Science*, (81):195–225, 2003.

[BG05] Mark Bickford and David Gauspari. Scores: A programming logic for distributed systems. Technical Report Technical Report 05-0007, ATC-NY, 2005.

[BGHP98] Andrew Barber, Philippa Gardner, Masahito Hasegawa, and Gordon D. Plotkin. From action calculi to linear logic. In Mogens Nielsen and Wolfgang Thomas, editors, *Computer Science Logic, 11^{th} International Workshop, Annual Conference of the EACSL, Aarhus, Denmark, August 23–29, 1997, Selected Papers*, volume 1414 of *Lecture Notes in Computer Science*, pages 78–97. Springer, 1998.

[BGS03] Marcel Becker, Limei Gilham, and Douglas R. Smith. Planware II: Synthesis of schedulers for complex resource systems. Submitted for publication, 2003.

[BH99] Mark Bickford and Jason J. Hickey. Predicate transformers for infinite-state automata in Nuprl type theory. In *Proceedings of 3^{rd} Irish Workshop in Formal Methods*, 1999.

[BKvRC01] Mark Bickford, Christoph Kreitz, Robbert van Renesse, and Robert Constable. An experiment in formal design using meta-properties. In J. Lala, D. Mughan, C. McCollum, and B. Witten, editors, *DARPA Information Survivability Conference and Exposition II (DISCEX-II)*, volume II of *IEEE Computer Society Press*, pages 100–107, Anaheim, CA, 2001.

[BKvRL01] Mark Bickford, Christoph Kreitz, Robbert van Renesse, and Xiaoming Liu. Proving hybrid protocols correct. In Richard Boulton and Paul Jackson, editors, *14^{th} International Conference on Theorem Proving in Higher Order Logics*, volume 2152 of *Lecture Notes in Computer Science*, pages 105–120, Edinburgh, Scotland, September 2001. Springer-Verlag.

[CAB+86a] Robert L. Constable, Stuart F. Allen, H. M. Bromley, W. R. Cleaveland, J. F. Cremer, R. W. Harper, Douglas J. Howe, T. B. Knoblock, N. P. Mendler, P. Panangaden, James T. Sasaki, and Scott F. Smith. *Implementing Mathematics with the Nuprl Proof Development System.* Prentice-Hall, NJ, 1986.

[CAB+86b] Robert L. Constable, Stuart F. Allen, H. M. Bromley, W. R. Cleaveland, J. F. Cremer, R. W. Harper, Douglas J. Howe, T. B. Knoblock, N. P. Mendler, P. Panangaden, James T. Sasaki, and Scott F. Smith. *Implementing Mathematics with the Nuprl Proof Development System.* Prentice-Hall, NJ, 1986.

[CC99] Michel Charpentier and K. Mani Chandy. Towards a compositional approach to the design and verification of distributed systems. In Jeannette Wing, Jim

Woodcock, and J. Davies, editors, *FM99: The World Congress in Formal Methods in the Development of Computing Systems*, volume 1708 of *Lecture Notes in Computer Science*, pages 570–589. Springer Verlag, 1999.

[CE82] Edmund M. Clarke and E. Allen Emerson. Synthesis of synchronization skeletons from branching time temporal logic. In *Proc. Workshop on Logics of Programs*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71. Springer–Verlag, 1982.

[CH88] Thierry Coquand and G. Huet. The calculus of constructions. *Information and Computation*, 76:95–120, 1988.

[CK93] K. Mani Chandy and Carl Kesselman. CC++: A declarative concurrent object oriented programming notation. In Gul Agha, Peter Wegner, and Akinori Yonezawa, editors, *Research Directions in Concurrent Object-Oriented Programming*, chapter 11, pages 281–313. MIT Press, Boston, 1993.

[Cle99] W. Rance Cleaveland, editor. *5^{th} International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 1579 of *Lecture Notes in Computer Science*. Springer, 1999.

[CM88] K. M. Chandy and J. Misra. *Parallel Program Design: A Foundation*. Addison-Wesley, 1988.

[Con71] Robert L. Constable. Constructive mathematics and automatic program writers. In *Proceedings of the IFIP Congress*, pages 229–233. North-Holland, 1971.

[Con02] Robert L. Constable. Naïve computational type theory. In H. Schwichtenberg and R. Steinbrüggen, editors, *Proof and System-Reliability, Proceedings of International Summer School Marktoberdorf, July 24 to August 5, 2001*, volume 62 of *NATO Science Series III*, pages 213–260, Amsterdam, 2002. Kluwer Academic Publishers.

[dAH01] Luca de Alfaro and Thomas A. Henzinger. Interface theories for component-based design. In *Proceedings of the First International Workshop on Embedded Software(EMSOFT)*, volume 2211 of *Lecture Notes in Computer Science*, pages 148–165. Springer-Verlag, 2001.

[EC82] E. Allen Emerson and Edmund M. Clarke. Using branching time temporal logic to synthesize synchronization skeletons. *Science of Computer Programming*, 2(3):241–266, 1982.

[Esc01] Robert Eschbach. A verification approach for distributed abstract state machines. In *PSI'02: Revised Papers from the 4th International Andrei Ershov Memorial Conference on Perspectives of System Informatics*, volume 2244 of *Lecture Notes in Computer Science*, pages 109–115. Springer-Verlag, London, UK, 2001.

[EvdMM98] Kai Engelhardt, Ron van der Meyden, and Yoram Moses. A program refinement framework supporting reasoning about knowledge and time. In Jerzy Tiuryn, editor, *Proc. Foundations of Software Science and Computation Structures (FOSSACS 2000)*, pages 114–129, Berlin/New York, 1998. Springer-Verlag.

[FHMV97] Ronald Fagin, Joseph Y. Halpern, Yoram Moses, and Moshe Y. Vardi. Knowledge-based programs. *Distributed Computing*, 10(4):199–225, 1997.

[GGV04] Uwe Glaesser, Yuri Gurevich, and Margus Veanes. Abstract communication model for distributed systems. *IEEE Transactions on Software Engineering*, 30(7):458–472, 2004.

[GM93] Michael Gordon and Tom Melham. *Introduction to HOL: A Theorem Proving Environment for Higher-Order Logic.* Cambridge University Press, Cambridge, 1993.

[GPS01] Cordell Green, Dusko Pavlovic, and Douglas R. Smith. Software productivity through automation and design knowledge. In *Software Design and Productivity Workshop*, 2001.

[GRS05] Yuri Gurevich, Benjamin Rossman, and Wolfram Schulte. Semantic essence of asml. *Theoretical Computer Science*, 343(3):370–412, October 2005.

[GSW96] Carla P. Gomes, Douglas R. Smith, and Stephen J. Westfold. Synthesis of schedulers for planned shutdowns of power plants. In *Proceedings of the Eleventh Knowledge-Based Software Engineering Conference*, pages 12–20. IEEE Computer Society Press, 1996.

[GT97] V. K. Garg and A. I. Tomlinson. Using the causal domain to specify and verify distributed programs. *Acta Informatica*, pages 667–686, 1997.

[GTL89] J-Y. Girard, P. Taylor, and Y. Lafont. *Proofs and Types*, volume 7 of *Cambridge Tracts in Computer Science.* Cambridge University Press, 1989.

[Hal00] Joseph Y. Halpern. A note on knowledge-based programs and specifications. *Distributed Computing*, 13(3):145–153, 2000.

[Hey02] Anthony J.G. Hey. *Feynman & Computation.* Westview Press, Boulder, 2002.

[HF89] Joseph Y. Halpern and Ronald Fagin. Modeling knowledge and action in distributed systems. *Distributed Computing*, 3(4):159–177, 1989.

[HLvR99] Jason J. Hickey, Nancy Lynch, and Robbert van Renesse. Specifications and proofs for Ensemble layers. In Cleaveland [Cle99], pages 119–133.

[HM03] D. Harel and R. Marelly. *Come, Let's Play: Scenario-Based Programming Using LSCs and the Play-Engine.* Springer-Verlag, New York, 2003.

[Hoa85] C. A. R. Hoare. *Communicating Sequential Processes.* Prentice Hall, 1985.

[HS99] Joseph Y. Halpern and Richard A. Shore. Reasoning about common knowledge with infinitely many agents. In *Proceedings of the 14th IEEE Symposium on Logic in Computer Science*, pages 384–393, 1999.

[HvR97] Mark Hayden and Robbert van Renesse. Optimizing layered communication protocols. In *Proceedings of the High Performance Distributed Computing*, Portland, Oregon, August 1997.

[Isa] Isabelle home page. `http://www.cl.cam.ac.uk/Research/HVG/Isabelle`.

[KHH98] Christoph Kreitz, Mark Hayden, and Jason J. Hickey. A proof environment for the development of group communications systems. In *Fifteen International Conference on Automated Deduction*, number 1421 in Lecture Notes in Artificial Intelligence, pages 317–332. Springer, 1998.

[KM94] K. Koskimies and E. Makinen. Automatic synthesis of state machines from trace diagrams. *Software–Practice and Experience*, 24(7):643–658, 1994.

[Kre99] Christoph Kreitz. Automated fast-track reconfiguration of group communication systems. In Cleaveland [Cle99], pages 104–118.

[Kre03] Christoph Kreitz. The FDL navigator: Browsing and manipulating formal content. Cornell University, Ithaca, NY, 2003. `http://www.nuprl.org/documents/Kreitz/03fdl-navigator.html`.

[KRS99] S. S. Kulkarni, J. Rushby, and N. Shankar. A case-study in component-based mechanical verification of fault-tolerant programs. In A. Arora, editor, *Proceedings of the 19th IEEE International Conference on Distributed Computing Systems Workshop on Self-Stabilizing Systems, Austin, TX*, pages 33–40. IEEE Computer Society Press, 1999.

[KW01] J. Klose and H. Wittke. An automata based interpretation of live sequence charts. In *Proceedings of Seventh International Coference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'01)*, 2001.

[Lam78] Leslie Lamport. Time, clocks and the ordering of events in a distributed system. *Comms. ACM*, 21(7):558–65, 1978.

[Lam93] Leslie Lamport. Hybrid systems in TLA+. In Grossman, Nerode, Ravn, and Rischel, editors, *Hybrid Systems*, volume 736 of *Lecture Notes in Computer Science*, 1993.

[Lam94] Leslie Lamport. The temporal logic of actions. *ACM Transactions on Programming Languages and Systems*, 16(3):872–923, 1994.

[Lam03] Leslie Lamport. *Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers.* Addison-Wesley, Boston, 2003.

[LKvR+99] Xiaoming Liu, Christoph Kreitz, Robbert van Renesse, Jason J. Hickey, Mark Hayden, Kenneth Birman, and Robert Constable. Building reliable, high-performance communication systems from components. In David Kotz and John Wilkes, editors, *17th ACM Symposium on Operating Systems Principles (SOSP'99)*, volume 33(5) of *Operating Systems Review*, pages 80–92. ACM Press, DIGITAL Systems Research Center 1999.

[LRSA02] J. Loyall, P. Rubel, R. Schantz, and M. Atighetc. Emerging patterns in adaptive, distributed real-time, embedded middleware. In Weerasak Witthawaskul, editor, *9th Conference of Pattern Language of Programs*, 2002.

[LT89] Nancy Lynch and Mark Tuttle. An introduction to Input/Output automata. *Centrum voor Wiskunde en Informatica*, 2(3):219–246, September 1989.

[Lyn96] Nancy Lynch. *Distributed Algorithms.* Morgan Kaufmann Publishers, San Mateo, CA, 1996.

[Mes03] Jos'e Meseguer. Software specification and verification in rewriting logic. Unpublished, 2003.

[Mil89] Robin Milner. *Communication and Concurrency.* Prentice-Hall, London, 1989.

[Mil93a] Robin Milner. Higher-order action calculi. In Egon Börger, Yuri Gurevich, and Karl Meinked, editors, *Computer Science Logic*, volume 832 of *Lecture Notes in Computer Science*, pages 238–260, 1993.

[Mil93b] Robin Milner. Structures for the λ-calculus action structures. University of Edinburgh, Edinburgh, UK, August 1993. Marktoberdorf.

[Mil94] R. Milner. Action structures and the π-calculus. In Helmut Schwichtenberg, editor, *Proof and Computation*, volume 139 of *NATO Advanced Study Institute, International Summer School held in Marktoberdorf, Germany, July 20–August 1, 1993, NATO Series F*, pages 219–280. Springer, Berlin, 1994.

[Mil96] Robin Milner. Calculi for interaction. *Acta Informatica*, 33(8):707–737, 1996.

[Mis01] Jayadev Misra. *A Discipline of Multiprogramming.* Springer Verlag, 2001.

[ML82] Per Martin-Löf. Constructive mathematics and computer programming. In *Proceedings of the Sixth International Congress for Logic, Methodology, and Philosophy of Science*, pages 153–175, Amsterdam, 1982. North Holland.

[MP92] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification.* Springer-Verlag, Berlin, 1992.

[MP95] Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems: Safety.* Springer-Verlag, Berlin, 1995.

[Mur91] Chetan Murthy. An evaluation semantics for classical proofs. In *Proceedings of Sixth Symposium on Logic in Comp. Sci.*, pages 96–109. IEEE, Amsterdam, The Netherlands, 1991.

[MW84] Z. Manna and P. Wolper. Synthesis of communicating processes from temporal logic specifications. *ACM Trans. Program. Lang. and Syst.*, 6(1):68–93, 1984.

[NPS90] Bengt Nordström, Kent Petersson, and Jan M. Smith. *Programming in Martin-Löf's Type Theory.* Oxford Sciences Publication, Oxford, 1990.

[NPW02] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, volume 2283 of *Lecture Notes in Computer Science.* Springer, 2002.

[NSS57] A. Newell, J.C. Shaw, and H.A. Simon. Empirical explorations with the logic theory machine: A case study in heuristics. In *Proceedings West Joint Computer Conference*, pages 218–239, 1957.

[Pau88] L.C. Paulson. A preliminary user's manual for Isabelle. Technical Report 133, University of Cambridge, Cambridge, England, 1988.

[Pau99] Lawrence C. Paulson. Mechanizing UNITY in isabelle. *ACM Transactions on Computational Logic*, 1999.

[PN90] L. Paulson and T. Nipkow. Isabelle tutorial and user's manual. Technical report, University of Cambridge Computing Laboratory, 1990.

[PS03a] Dusko Pavlovic and Douglas R. Smith. Software development by refinement. In Bernhard K. Aichernig and T. S. E. Maibaum, editors, *UNU/IIST 10th Anniversary Colloquium, Formal Methods at the Crossroads: From Panaea to Foundational Support*, volume 2757 of *Lecture Notes in Computer Science*, pages 267–286. Springer, 2003.

[PS03b] Dusko Pavlovic and Douglas R. Smith. Software development by refinement. In Bernhard K. Aichernig and T. S. E. Maibaum, editors, *UNU/IIST 10th Anniversary Colloquium, Formal Methods at the Crossroads: From Panaea to Foundational Support*, volume 2757 of *Lecture Notes in Computer Science*, pages 267–286. Springer, 2003.

[QS98] Shaz Qadeer and Natarajan Shankar. Verifying a self-stabilizing mutual exclusion algorithm. In David Gries and Willem-Paul de Roever, editors, *IFIP International Conference on Programming Concepts and Methods: PROCOMET'98*, pages 424–443, Shelter Island, NY, June 1998. Chapman & Hall.

[Sch97] Fred B. Schneider. *On Concurrent Programming.* Springer-Verlag, New York, 1997.

[SG96] Douglas R. Smith and Cordell Green. Toward practical applications of software synthesis. In *FMSP'96, The First Workshop on Formal Methods in Software Practice*, pages 31–39., 1996.

[SL90] D.R. Smith and M.R. Lowry. Algorithm theories and design tactics. *Science of Computer Programming*, 14(2–3):305–321, October 1990. Report KES.U.89.3, Kestrel Institute.

[Thé01] Laurent Théry. A machine-checked implementation of Buchberger's algorithm. *Journal of Automated Reasoning*, 26(2):107–137, February 2001.

[Tur37] A. M. Turing. On computable numbers, with an application to the Entscheidungs problem. In *Proceedings London Math Society*, pages 116–154, 1937.

[Var95] M. Y. Vardi. An automata-theoretic approach to fair realizability and synthesis. In P. Wolper, editor, *Computer Aided Verification, Proceedings of the 7th International Conference*, volume 939 of *Lecture Notes in Computer Science*, pages 267–292. Springer-Verlag, 1995.

[vRBH+98] Robbert van Renesse, Kenneth P. Birman, Mark Hayden, Alexey Vaysburg, and David Karr. Building adaptive systems using Ensemble. *Software: Practice and Experience*, 28(9):963–979, July 1998.

[Whe82] John A. Wheeler. The computer and the universe. *Int. J. Theoretical Physics*, 21:557–571, 1982.

[Whe89] John A. Wheeler. Information, physics, quantum: The search for links. In *Proc. 3d Int. Symp. Foundations of Quantum Mechanics*, Tokyo, 1989.

[Win80] G. Winskel. *Events in Computation.* PhD thesis, University of Edinburgh, 1980.

[Win89] G. Winskel. An introduction to event structures. In J. W. de Bakker et al., editors, *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, number 345 in Lecture Notes in Computer Science, pages 364–397. Springer, 1989.

[ZdRvEB85] Job Zwiers, Willem P. de Roever, and Peter van Emde Boas. Compositionality and concurrent networks: Soundness and completeness of a proofsystem. In *ICALP 1985*, pages 509–519, 1985.