Modal Logic Revisited

Frank Pfenning

The Scottfest
In honor of Dana Scott on his 70th Birthday
Pittsburgh, Pennsylvania
October 12, 2002

Source

[1] Dana Scott, **Advice on Modal Logic**. In: *Philosophical Problems in Logic: Some Recent Developments*, K. Lambert (ed.), pp. 143–173, Dordrecht Reidel, 1970. Based on a talk presented during a colloquium on free logic, modal logic and related areas held at the University of California at Irvine, May 1968.

Claimer: I am not making this up.

Disclaimer: The quotes may be taken out of context.

A Quote

Everyone knows how much more pleasant it is to give advice than to take it. Everyone knows how little heed is taken of all the good advice he has to offer. Nevertheless, this knowledge seldom restrains anyone, least of all the present author. He has been noting the confusions, misdirections of emphasis, and duplications of effort current in studies of modal logic and is, by now, anxious to disseminate all kinds of valuable advice on the subject.

Classical Modal Logic in Computer Science

- Reasoning about [concurrent] programs
  - A. Pnueli, **The Temporal Logic of Programs**, 1977.
    ACM Turing Award, 1996.
- Reasoning about hardware; model-checking
  - E. Clarke and A. Emerson, **Synthesis of Synchronization Skeletons for Branching Time Temporal Logic**, 1981.
    R. Bryant, E. Clarke, A. Emerson, K. McMillan;
    ACM Kanellakis Award, 1999.
- Knowledge representation
  - From frames to KL-ONE to **Description Logics**
    [MacGregor'87] [Borgida'89] [Baader'91]

A Quote

I feel it is important to be thinking of D *as* fixed in advance. *This does not mean that one knows all the elements of* D *in any constructive sense, [...]. Maybe, in the future we shall understand the logic of* potential totalities *(through intuitionism possibly?) but for the present our simple two-valued logic demands this idealization.*

Intuitionistic Modal Logic in Computer Science I

- **Logical frameworks** / Modal λ-Calculus
  [F. Pfenning & H.C. Wong'95]
- **Staged computation** / Run-time code generation
  [F. Pfenning & R. Davies'96]
  [P. Wickline, P. Lee, F. Pfenning, R. Davies'98]
- **Partial evaluation** / Binding-time analysis
  [Davies'96] [Z. Benaissa, E. Moggi, W. Taha, T.Sheard'99]
- **Monadic meta-language** / Type systems for effects
  [S. Kobayashi'97] [P.N. Benton, G.M. Biermann, V.C.V. de Paiva'98] [F. Pfenning & R. Davies'01]

Intuitionistic Modal Logic in Computer Science II

- **Intensionality** / Meta-programming
  [J. Despeyroux, F. Pfenning, C. Schürmann'97]
  [A. Nanevski'02]
- **Proof irrelevance** / Dead code elimination
  [F. Pfenning'01] [S. Awodey & A. Bauer'01]
- **Code mobility** / Distributed computation
  [J. Moody & F. Pfenning]

A Quote

Before embarking on details, here is one general piece of advice. One often hears that modal (or some other) logic is pointless because it can be translated into some simpler language in a first-order way. Take no notice of such arguments. There is no weight to the claim that the original system must therefore be replaced by the new one. What is essential is to single out important concepts and to investigate their properties.

Run-Time Code Generation

- Ordinary evaluation: compile, then run binary code
- Run-time code generation: generate additional code while program runs and then execute it
- Used for optimization, particularly in system code
- Status in 1995: white magic (super-hacker required!)
- Intuitionistic modal logic as organizing principle!
- R. Davies and F. Pfenning,
  A Modal Analysis of Staged Computation, 1996.

A Quote

So my advice is to continue with the two-valued logic because it is easy to understand and easy to use in applications; then when someone has made the other logic workable a switch should be reasonably painless.

Run-Time Code Generation via Modal Types

- Follow methodology of *judgments* [Martin-Löf'83]
- Need **source expression** to generate code
- Introduce a judgment of source expressions
- Internalize the judgment as a type, $\Box A$
- Which modal laws will $\Box A$ satisfy?

Ordinary Terms and Types

- Basic judgment: $M : A$ *"M is a proof term for A"*
  "M has type A"
- Parametric and hypothetical judgment (may reorder assumptions):

$$\underbrace{x_1{:}A_1, \ldots, x_n{:}A_n}_{\text{value variables } \Gamma} \vdash M : A$$

- Hypothesis rule: $\Gamma, x{:}A \vdash x : A$
- Substitution property:

  If $\Gamma \vdash M : A$
  and $\Gamma, x{:}A \vdash N : C$
  then $\Gamma \vdash [M/x]N : C$.

- λ-abstraction, application, β-reduction as usual.

Source Expressions

- Categorical judgment: $M :: A$ if $\bullet \vdash M : A$.
- $M :: A$ "M *is a source expression of type* A"
- Hypothetical judgment:

$$\underbrace{u_1 :: B_1, \ldots, u_k :: B_k}_{\text{expression variables}} \;;\; \underbrace{x_1 : A_1, \ldots, x_n : A_n}_{\text{value variables}} \vdash N : C$$

- Additional hypothesis rule: $(\Delta, u :: A); \Gamma \vdash u : A$
- Substitution property:

If $\Delta; \bullet \vdash M : A$
and $(\Delta, u :: A); \Gamma \vdash N : C$
then $\Delta; \Gamma \vdash [M/u]N : C$.

Source Expression Types

- Type: $\Box A$ "*terms denoting expressions of type* A"
- Introduction and elimination rules
- Constructor:

$$\frac{\Delta; \bullet \vdash M : A}{\Delta; \Gamma \vdash \mathsf{box}\, M : \Box A}\, \Box I$$

- Destructor:

$$\frac{\Delta; \Gamma \vdash M : \Box A \qquad (\Delta, u{::}A); \Gamma \vdash N : C}{\Delta; \Gamma \vdash \mathsf{let\,box}\, u = M \;\mathsf{in}\; N : C}\, \Box E$$

- Reduction:

$$\mathsf{let\,box}\, u = \mathsf{box}\, M \;\mathsf{in}\; N \longrightarrow [M/u]N$$

A Quote

Of course an elimination metatheorem can be justified on the basis of the axioms; this may often be the case. Thus my advice is to leave the elimination problems to the development *of the theory rather than having them complicate the* formalization*.*

Run-Time Code Generation as Intuitionistic S4

- Curry-Howard isomorphism applied to modal logic S4!
- $\Box A$ as a *generator* for binary code of type A
- $\vdash eval : \Box A \to A$
  $\lambda x.\, \mathsf{let\,box}\, u = x\, \mathsf{in}\, u$
  Generate code for A and execute it
- $\vdash apply : \Box(A \to B) \to \Box A \to \Box B$
  $\lambda f.\, \lambda x.\, \mathsf{let\,box}\, u = f\, \mathsf{in\ let\,box}\, w = x\, \mathsf{in\ box}\,(u\, w)$
  Compose the generators for a function and its argument
- $\vdash quote : \Box A \to \Box\Box A$
  $\lambda x.\, \mathsf{let\,box}\, u = x\, \mathsf{in\ box}\,(\mathsf{box}\, u)$
  Quote a generator

Contributions of Modal Logic

- Capture staging for run-time code generation
- A staging error in the program is a type error!
- Construct box M compiles to a code generator
- Construct let box $u = M$ in N binds u to a generator
- Expression variable u calls code generator
- (Aside: other computational interpretations possible!)
- Logic is quite familiar: intuitionistic S4
- Incorporated into MetaML
  [Z. Benaissa, E. Moggi, W. Taha, T.Sheard'99]
- Kripke interpretation:

  Possible worlds as stages in a computation!

Type Systems for Effects

- Monadic meta-language: generic framework to introduce effects into a pure language (mutable arrays, exceptions, call/cc, memoization, input/output, etc.)
- Type system to distinguish pure and effectful expressions
- Widely used as a practical programming technique [Haskell]
- Kripke interpretation:

  Possible worlds as states in an effectful computation!

Stable and Ephemeral Values

- Re-interpret value and expression typing judgments
- $W :: A$ — "W *is a stable value of type* A"
- W is available in all future worlds (states)
- W cannot be changed by an effect (e.g., assignment)
- Internalize: $\Box A$ are terms denoting stable values
- $V : A$ — "V *is an ephemeral value of type* A"
- V is available only in the present world (state)
- V can be destroyed or changed by an effect

Possible Values

- $E \div A$ — "*E is a computation of type A*"
- E may be effectful (change worlds)
- Internalize: $\Diamond A$ are terms denoting computations
- Unlike classical modal logic, $\Box$ and $\Diamond$ are not interdefinable (with and without negation)
- There are 6 distinct modal operators
  $\Box$, $\Diamond$, $\Box\Diamond$, $\Diamond\Box$, $\Diamond\Box\Diamond$, $\Box\Diamond\Box$

A Quote

Here is what I consider one of the biggest mistakes of all in modal logic: concentration on a system with just one *modal operator.*

New Judgmental Principles

- If $\Delta; \Gamma \vdash M : A$ then $\Delta; \Gamma \vdash M \div A$
  A value is a trivial computation
- If $\Delta; \Gamma \vdash E \div A$ and $\Delta; x{:}A \vdash F \div C$ then $\Delta; \Gamma \vdash \langle\!\langle E/x \rangle\!\rangle F \div C$
  We can compose certain computations
- Internalize as types:

$$\frac{\Delta; \Gamma \vdash E \div A}{\Delta; \Gamma \vdash \mathsf{dia}\, E : \Diamond A}\, \Diamond I$$

$$\frac{\Delta; \Gamma \vdash M : \Diamond A \qquad \Delta; x{:}A \vdash E \div C}{\Delta; \Gamma \vdash \mathsf{let\ dia}\, x = M\, \mathsf{in}\, E \div C}\, \Diamond E$$

Laws of Possibility in Intuitionistic S4

- $\vdash \mathit{here} : A \to \Diamond A$
  $\lambda x{:}A.\, \mathsf{dia}\, x$
  A value is a trivial computation

- $\vdash \mathit{chain} : \Diamond\Diamond A \to \Diamond A$
  $\lambda x.\, \mathsf{dia}\, (\mathsf{let\ dia}\, y = x\, \mathsf{in\ let\ dia}\, z = y\, \mathsf{in}\ z)$
  We can compose certain computations

- $\vdash \mathit{lift} : \Box(A \to B) \to (\Diamond A \to \Diamond B)$
  $\lambda x{:}\Box(A \to B).\, \lambda y{:}\Diamond A.\, \mathsf{let\ box}\, u = x\, \mathsf{in\ dia}\, (\mathsf{let\ dia}\, z = y\, \mathsf{in}\ u\, z)$
  We can lift stable functions to act on computations

A Quote

One bit of advice seems needed at this juncture: the aim of logic is not solely to provide completeness proofs. The real aim is conceptual clarification.

Monadic Meta-Language

- Usually (ML, Haskell) all values are stable
- Define

$$A \Rightarrow B \;=\; (\Box A) \to B$$
$$\bigcirc A \;=\; \Diamond \Box A$$

- $\Rightarrow, \bigcirc$ form a *monad* [Moggi'89]
- Axiomatically:

$$\vdash \; \_ : A \Rightarrow \bigcirc A$$

$$\vdash \; \_ : \bigcirc\bigcirc A \Rightarrow \bigcirc A$$

$$\vdash \; \_ : (A \Rightarrow B) \Rightarrow (\bigcirc A \Rightarrow \bigcirc B)$$

Contributions of Modal Logic

- Possible worlds as *states of computation*
- Understanding of monads via intuistionistic S4 modalities
- Cleaner, more tractable λ-calculus
- Many effects in programming languages are specific instances (mutable reference, exceptions, continuations, input/output, concurrency, non-determinism, etc.)

A Quote

Up to this point, our indices of possible worlds have been too vague, too abstract. [....] That is, an analysis of the individual must be attempted. The method is not yet discredited, because it has not really been carried out in sufficient detail. [....] One essential step to take in this analysis is to make the elements more specific. [....] My advice is to work on this problem.

Mobile Code and Distributed Computation

- L. Caires & L. Cardelli, **A Spatial Logic for Concurrency**, 2000, 2002
  - Classical logic
  - Multiple modal operators for time, space, and names
  - In the tradition of temporal logics
- Our goal: a *constructive* and *logical* foundation for mobile code and distributed computation
- **Disclaimer:** Unfinished work in progress
  [J. Moody & F. Pfenning]

A Re-Interpretation of Modal Operators

- Kripke interpretation:

  Each world corresponds to a locus of computation
  Reachability corresponds to (asymmetric) connections

- $M :: A$ — "M *can be computed anywhere*"
  box $M : \Box A$ is fully mobile

- $M : A$ — "M *can be computed here*"

- $E \div A$ — "E *can be computed somewhere*"
  dia $E : \Diamond A$ can be computed at some unknown world

- Example definable operator: $M : \Diamond\Box A$
  A mobile value is computed somewhere

- Example definable operator: $M : \Box\Diamond A$
  A mobile reference to a computation (resource) somewhere

Logical laws interpreted

- $\vdash \Box A \rightarrow A$
  We can reach the current machine (reflexivity)
- $\vdash \Box A \rightarrow \Box\Box A$
  We can forward mobile code (transitivity)
- $\vdash \Box(A \rightarrow B) \rightarrow \Box A \rightarrow \Box B$
  We can compose mobile code
- $\vdash A \rightarrow \Diamond A$
  Here is somewhere (reflexivity)
- $\vdash \Diamond\Diamond A \rightarrow \Diamond A$
  We can forward access to resources (transitivity)
- $\vdash \Box(A \rightarrow B) \rightarrow (\Diamond A \rightarrow \Diamond B)$
  We can send mobile code to a resource location

A Quote

We have to consider not only of the flow of time but also of alternative *courses of events. — No, come to think of it, that is not the answer either, for that only makes the individual concept* $\|\tau\|$ *fatter but not more amusing. — Or maybe it does. (Oh my, I see that much more thought and experimentation are needed to make the ideas into something useful. In any case I feel that a precise and general semantical framework is essential, and that is, as I have been trying to indicate, now available.)*

Summary: Modal Logic in Computer Science

- Classical
  - Temporal logics [1977]: reasoning with concurrency
  - Description logics [1987]: knowledge representation
  - Spatial logics [1998]: reasoning about mobile programs
- Intuitionistic
  - S4 [1996,2001]: staged computation, effects, mobility(?)
  - Linear-Time Temporal Logic [1996]: partial evaluation
  - Others [2000,2002]: intensionality, proof irrelevance, ...

A Final Quote

Postscript (December, 1969)

This paper was written very hastily in the latter part of May, 1968. The haste is apparent and the style intolerable; I find it now very painful reading.