Application of Logic to Digital System Design

Paul S. Miner

May 10, 1995

Outline

- Formal Specification
- Formal Proof Activities
- Degree of Rigor
- Obstacles to Getting Started

Formal Specification

Formal Specification: Use of notations derived from formal logic to describe

- *assumptions* about the world in which a system will operate
- *requirements* that the system is to achieve
- the intended *behavior* of the system

Styles of Specification Different approaches are used for these descriptions:

- *Properties*—enumeration of assumptions and requirements
- *Functions*—express desired behavior or design descriptions
- *State-machines*—express desired behavior or design descriptions
- ...

Assumptions at one level become requirements at a lower level.

Example of Property-Based Specification (Fault-tolerant clock synchronization)

1. There is a $\rho \ll 1$ such that for any clock C_p that is non-faulty during the interval from t_1 to t_2:

$$(1-\rho)(t_2 - t_1) \leq C_p(t_2) - C_p(t_1) \leq (1+\rho)(t_2 - t_1)$$

2. There is a δ such that if clocks C_p and C_q are non-faulty at time t, then:

$$|C_p(t) - C_q(t)| < \delta$$

Example of Functional Specification (Floating-point Operations)

IEEE floating-point arithmetic requires that each arithmetic operation *be performed as if it first produced a result correct to infinite precision and unbounded range, and then coerced this result to fit in the destination's precision.* [ANSI/IEEE STD 854-1987]

Floating-Point Operations
(declarations)

$$
\begin{aligned}
FP &= \text{the set of floating-point numbers} \\
Fin &= \text{the set of finitely valued floating-point numbers, } Fin \subset FP \\
M &= \text{the set of rounding modes} \\
\mathbf{R} &= \text{the set of real numbers} \\
fin1, fin2 &\in Fin \\
mode &\in M
\end{aligned}
$$

`fp_to_real:` $Fin \rightarrow \mathbf{R}$
`real_to_fp:` $R \times M \rightarrow FP$

Floating-Point Addition (Partial Specification)

For floating-point addition of two finitely valued arguments, $fin1$ and $fin2$, the expression

$$\texttt{fp\_to\_real}(fin1) + \texttt{fp\_to\_real}(fin2)$$

defines the infinitely precise result.

Floating-Point Addition
(continued)

The formal specification of floating-point addition for finitely valued arguments is:

$$\texttt{fp\_add\_finite}(fin1, fin2, mode) \mathrel{\hat{=}}$$
$$\quad \texttt{real\_to\_fp((fp\_to\_real(}fin1\texttt{)} + \texttt{fp\_to\_real(}fin2\texttt{)), } mode\texttt{)}$$

State-machine Specification

Design layers formalized as state machines

- State represents memory contents and hardware status
- Transition *function* defines state transitions

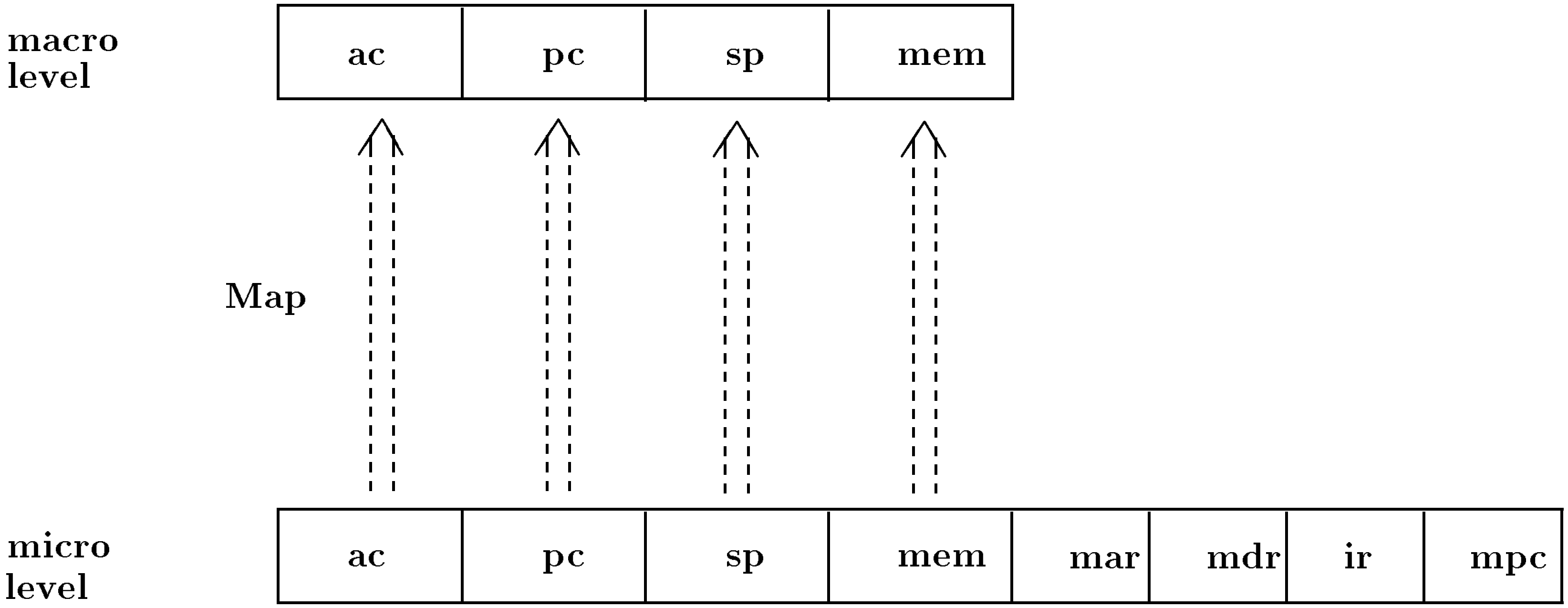
Interpretation maps lower level states into higher level states

$$\begin{tikzcd} s_i \arrow[r, "N_i"] & t_i \\ s_{i+1} \arrow[u, "Map"] \arrow[r, "N_{i+1}"] & t_{i+1} \arrow[u, "Map"'] \end{tikzcd}$$

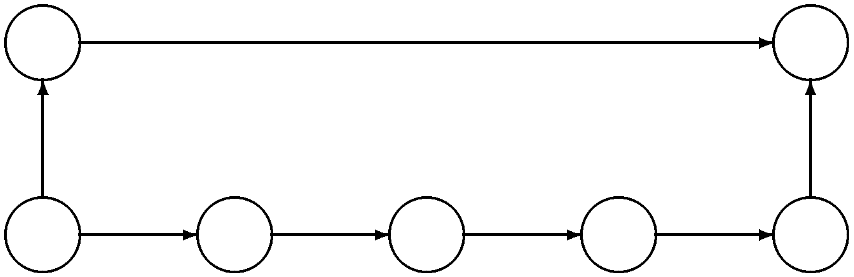
Need to show that diagram "commutes" to establish that layer $i+1$ correctly implements layer i:

$$Map(N_{i+1}(s_{i+1})) = N_i(Map(s_{i+1}))$$

Abstraction Mappings
(Data Abstraction)



Temporal Abstraction



Formal Proof Activities

Use of methods from formal logic to

1. *analyze* specifications for certain forms of consistency, completeness
2. *prove* that specified behavior will satisfy the requirements, given the assumptions
3. *prove* that a more detailed design *implements* a more abstract one

Formal Analysis of a Specification (1)

One property we might want to prove about the functions `real_to_fp` and `fp_to_real` from the floating-point addition specification is that for every finitely valued floating-point number and every rounding mode:

$$\texttt{real\_to\_fp(fp\_to\_real(}fin\texttt{),}mode\texttt{)} = fin$$

Verification of Fault-Tolerant Algorithms (2)

Top-level: Properties that algorithm should possess

Lower-level: Abstract description of the algorithm and underlying assumptions

Prove: The algorithm satisfies desired properties given the assumptions

Design Verification (3)

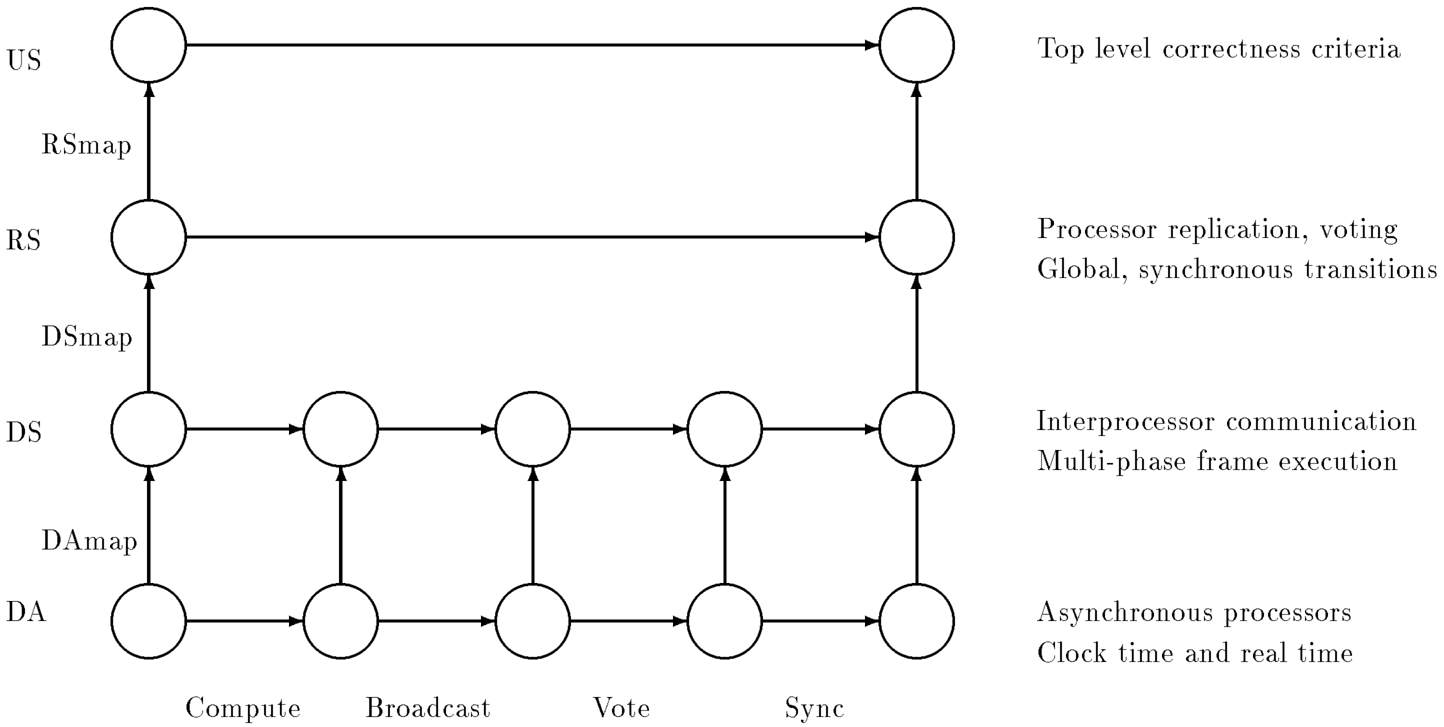
Top-level: Abstract description of system (and assumptions)

Lower-level: Detailed description of system (and assumptions)

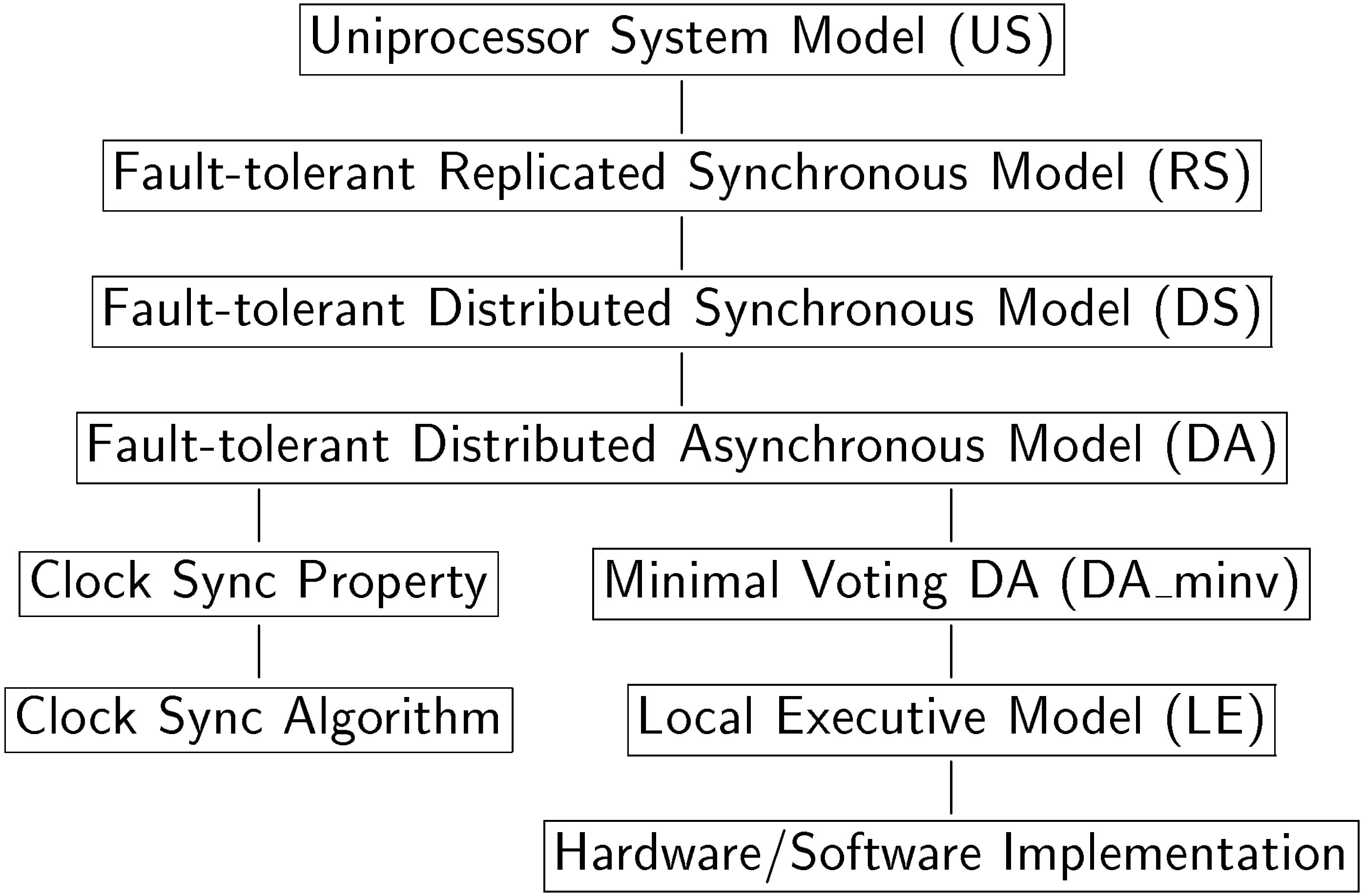
Prove: The detailed system description has the same behavior as the abstract description given the assumptions and an abstraction function relating the two systems.

Design Refinement and Proof
(Example: Reliable Computing Platform)

Single-frame state transition divided into four phases



Hierarchical Specification of the Reliable Computing Platform



Correctness of Specification

How do we know that a specification is what we intend?

- At intermediate levels in the design hierarchy, the specification is a refinement of higher-level requirements
  - If the higher levels are related by proof, then the requirements at this level are sufficient to ensure the high level requirements
- At the top-level, we must resort to review.
  - The presence of a formal specification enables formal challenges. A review can identify additional properties that a design must possess. Formal verification techniques can then be used to either prove the design already possesses the property or reveal its absence.
- At the bottom levels in a design hierarchy, we must show that the assumptions are consistent.

Levels of Rigor

Level 1: Specification (and proof) using conventional mathematical notation.

Level 2: Specification using a formal specification language with manual proofs.

Level 3: Specification in a formal language with automatically checked or generated proofs.

Mechanized Support for Formal Methods

General Purpose Theorem Provers: Specification and Proof using the logic supported by theorem prover. Proofs are semi-automatic. Many of the steps are automated, but some require insight.

Examples include: PVS, HOL, Nuprl, Nqthm/Acl2, IMPS, ...

Specialized Approaches:

Model Checking: Fully automatic proofs that state machine description of hardware possess specified properties. Specifications given in a decidable temporal logic. An example system is SMV. Tools employing related techniques include COSPAN and Murϕ.

Design Derivation: Design proceeds from a behavioral level description to a hardware design via a series of *correctness preserving refinements*. Example systems include DDD and DRS.

Software: Machine checked verification of Ada code (Penelope). Decision tables for specification of software requirements (TableWise).

Obstacles to getting started

- Diversity of tools
  - Choice of proper tool is difficult
  - Over 50 notations, methods, and tool listed on WWW Formal Methods Virtual Library
    http://www.comlab.ox.ac.uk/archive/formal-methods.html
  - Many domain specific tools and techniques
  - Little support for combining results from different tools
  - Some efforts currently in progress
- Immaturity of Tools
  - Most tools are research prototypes
- Lack of sufficient Libraries
- Education