ECE 208 Lecture Notes

based on

Mathematical Logic for Computer Science

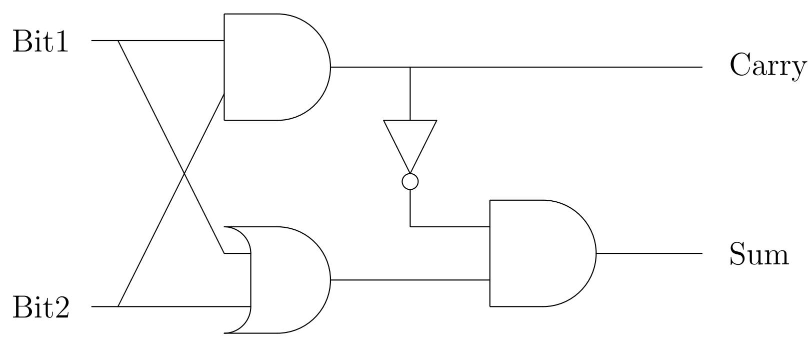
(Third Revised Edition)

Springer, 2012

Mordechai (Moti) Ben-Ari
http://www.weizmann.ac.il/sci-tea/benari/

Chapter 1: Introduction

- **Propositional logic** was seen in ECE 108 ...
- it's essentially the same as the 2-element Boolean algebra on $\{0, 1\}$
  – **switching algebra** – from ECE 124:



$$
\begin{aligned}
\text{Carry} &= \text{Bit1} \wedge \text{Bit2} \\
\text{Sum} &= \text{Bit1} \oplus \text{Bit2} = (\text{Bit1} \vee \text{Bit2}) \wedge \neg(\text{Bit1} \wedge \text{Bit2}) \ .
\end{aligned}
$$

– 'half-adder' circuit.

- What's the difference?
  - Boolean algebra focuses on operations: AND, OR, NOT, etc. ...
  - historically, logic focuses on *formal proof*:
    * defined by the rules of a *deductive system*;
    * formal in the sense that the rules depend only on the form, not the meaning, of formulas.
- Because of their formal nature, machines can check for proofs, and search for proofs, giving rise to *automatic theorem provers*.
- The semantic problem of determining whether a propositional-logic formula is *satisfiable* is notoriously hard (computationally):

- shorthand name for the problem is SAT;
- 'SAT-solver' tools have useful applications to other constraint-satisfaction problems, such as package management.

- Propositional logic extends to **predicate logic** – or *first-order* logic – with the addition of constant symbols, variable symbols, function symbols, and quantifiers:

$$
\begin{aligned}
&0 \in \mathbb{N} \ , \\
&(\forall n \in \mathbb{N}) \ [S(n) \in \mathbb{N}] \ , \\
&(\forall m, n \in \mathbb{N}) \ [m = n \iff S(m) = S(n)] \ , \\
&(\forall n \in \mathbb{N}) \ [0 \neq S(n)]
\end{aligned}
$$

  - these are the *Peano axioms* of arithmetic, with constant 0, variables m, n, and function symbol S, for 'successor.'
  - formulas that can be formally deduced from these axioms, and those of predicate logic, are *theorems* of arithmetic.

- The textbook uses $\rightarrow$, $\leftarrow$, and $\leftrightarrow$; we'll stick with $\Rightarrow$, $\Leftarrow$, and $\iff$.

- Predicate logic is used, for example,
  - in databases,
  - in 'satisfiability modulo theory' (SMT) solvers, which combine SAT-solving and automatic theorem-proving,
  - and in formal verification of software and hardware.

- **Temporal logic** is interpreted w.r.t. a sequence of discrete time instants (in *linear* TL).

- Some **temporal operators**:

  - $\bigcirc p$ – 'next p': p will be true at the next time instant;
  - $\Box p$ – 'henceforth p': p is true now and throughout the future;
  - $\Diamond p$ – 'eventually p': p is true either now or at some future time.
  - $p \mathcal{U} q$ – 'p until q': until q becomes true, p will be true.

- Allows for 'temporal reasoning':

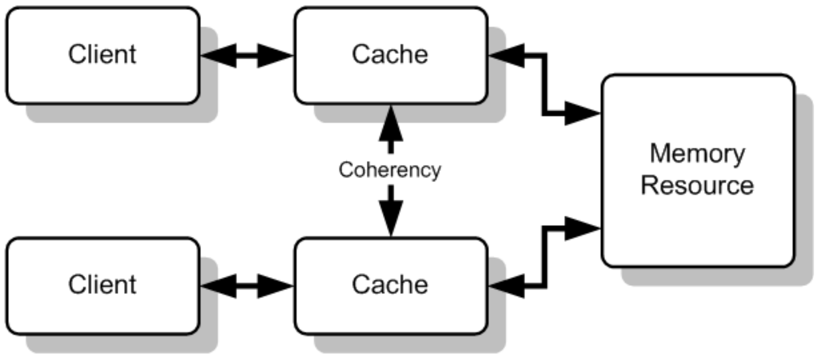
  - $\Box(p \implies \bigcirc p) \implies \Box(p \implies \Box p)$:
    if, whenever p is true at some future time, it's also true at the next instant, then if p ever becomes true, it will stay true from then on.
  - $\Box p \iff (p \wedge \bigcirc \Box p)$:
    p is true 'henceforth' if and only if it's true now, and at the next instant, it's true 'henceforth.'
  - $\Diamond p \iff (p \vee \bigcirc \Diamond p)$:
    p is eventually true if and only if it's true now, or at the next instant it's eventually true.
  - $p \mathcal{U} q \iff q \vee (p \wedge \bigcirc (p \mathcal{U} q))$

- Temporal logic is used, for example, in the formal verification of concurrent systems:

  - system modelled as a collection of interacting finite state machines (say);
  - problem is to check rigorously that system satisfies a given TL specification;
  - such 'model-checking' was the first big industrial application of formal verification.

- Example applications of formal verification:

Cache coherency:



  - 2 key methods of tracking coherency:

    Write-update: when one processor writes to memory, update all caches' entries.

    Write-invalidate: when one processor writes to memory, mark other caches' corresponding entries as invalid (saving BW);

  - 2 key architectures:

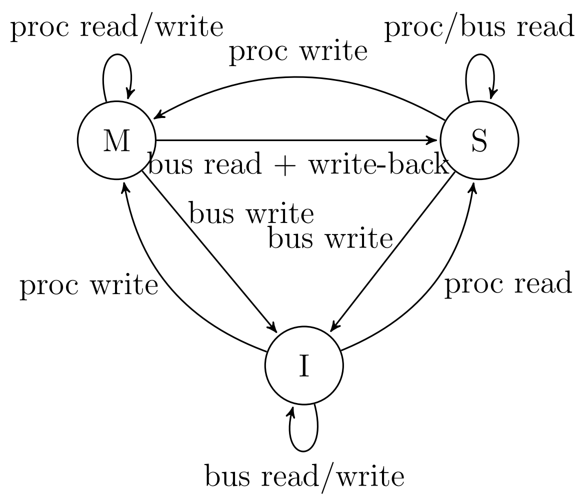
    Snooping: based on common read/write bus, and monitoring (bus *snooping* or *sniffing*) by individual cache controllers;

    Directory-based centralized directory monitors reads and writes, updates status of cache entries.

  - **Example**: in a snooping, write-invalidate protocol, each memory block in each cache might have 3 possible states:
    * modified, shared, or invalid.
  - When processor P writes to block B in its own cache, the state of block B changes to **modified** in that cache, and (via bus snooping) to **invalid** in other caches (and in the main memory);
  - processor P can then continue to read from and write to that block;

– but if another processor attempts to read from the ‘invalid’ block B in its own cache, a ‘cache miss’ occurs, and a bus request is sent for the contents of that block;
– processor P then writes the contents to the bus and to the main memory (latter action is a *write-back*);
– block B then takes the value **shared** in the caches of both processors, because they both contain the up-to-date value, as now does the main memory;
– if other processors request the contents of the block, they will be read from memory, and the block will take the value **shared** in those processors’ caches too;
– when some processor writes to block B in its cache, the cycle repeats.

Verification amounts to specifying and proving correct behaviour of a collection of interacting finite state machines:



Example spec: If processor P writes x to block B, then any read of block B (from any cache) yields x, until there’s another write to block B.

5G networks

- Increased speed due to storage of data 'closer' to subscriber ...
- creates similar consistency issues to those of cache coherency.

Cloud computing services

- Amazon Web Services uses formal verification extensively ...
- for security 'of' and 'in' the cloud.
- Increased assurance leads to faster adoption by customers.

- Formal verification not only finds bugs exhaustively, but demonstrates to clients and regulators that systems are 'correct.'