Logic for CS Undergraduates: A Sketch

Informal Formality - Distinguishing Mathematical, Computational, and the "Natural"

1. Deduction: Propositional Logic and Truth-Value Semantics
2. First-Order Logic for Novices: FOL, FOL=, Equational Mostly
3. First-Order (mostly) Structures
4. Interpretations and Representations
5. Computational Interpretations and Stored-Program Models
6. Interpretations and Representing Worlds

Dennis E. Hamilton for LogTeach-22, 2022-07-26
dennis.hamilton@acm.org; orcmid@msn.com

1. Deduction: Propositional Logic and Truth-Value Semantics

- Symbology distinguished from computation operations

  E.g, $p \Leftrightarrow q$, $p \Rightarrow q$, $p \vee q$, $p \wedge q$, $\neg p$
- Propositional Truth-valued semantics without constants (T & F in tabulations at most)
- Emphasize
  - Logic being about proper arguments concerning hypothetical cases, not assertion of fact (by itself)
  - How modus ponens serves as a fundamental deduction form
  - That one cannot conclude much about p, q, given only assertion of $p \Rightarrow q$
  - Satisfiability and soundness, rejecting "$p, \neg p \vdash p \wedge \neg p$", other inconsistencies
  - The fixity of propositions in a chain of deductions

2. First-Order Logic for Novices: FOL, FOL=, Equational Mostly

- Establishing Domains of Discourse, Terms, Predicates and Quantifiers
- Continue keeping it unlike programming: $\forall x \ \ \exists y$
- Classes and relations via predicates, careful with or avoid Cartesian $A \times B$?
- Use FOL= and equational forms with understood $\forall x\ y\ z$ … connecting with familiar algebraic forms and free variables
- Provide axiomatic introduction of familiar forms: integer, rational domains
- **Introduce Boolean Algebras** as valuable with respect to digital representations
- Non-programming notation: $\top$, $\bot$, $\sim x$, $x \cap y$, $x \cup y$, $x \dot{-} y$, $x \subseteq y$, $x \oplus y$
- The correspondence of concrete Boolean algebras and one of Boolean vectors
- Partial orderings

3. First-Order (mostly) Structures

Setting up for interpretation and representation

- Characterize structure ‹*s*› as a triple, ‹*s*› $=$ $\langle$*sD*, *sF*, *sT*$\rangle$
  - *sD*, the domain of discourse, the individuals
  - *sF*, the functions (and predicates) over the domain
  - *sT*, the FOL(=) applied theory that characterizes the structure
- *sT* introduces primitive individuals, functions, and axioms that extend over *sD*
- It is not expected to exhibit *sF*, although it is useful to establish elements of it in the usual manner by namings, cases, and recurrences
- Well-definedness and extensionality apply: $\forall x\ [f(x) = g(x)]$ implies that formulations of *f* and *g* in *sT* distinguish the same function in *sF*.
- For denumerable *sD*, establish canonical forms for differentiation of the individuals in the terms of *sT*. *Bonus*: canonical-form context-free grammars
- Structural induction

4. Interpretations and Representations

- By ***interpretation*** of structure in another, the usual model-theoretic arrangement is intended, without demanding a model
- Specify how individuals/primitives/axioms of structure ‹α› are interpreted by corresponding structure ‹β› individuals/functions/predicates such that whatever is logically deducible in ‹α› holds for *the counterparts* in ‹β›.
- If equality in ‹α› is sent to equality in a ‹β› interpretation, the interpretation is an implementation, otherwise it is a simulation (after Thomas Forster)
- Show mathematical engineering with equivalence classes to avoid constraints in ‹β› to confine the interpretation of ‹α›.
- Successful ***representation*** is achieved when there is a straightforward determination of α*D* individuals corresponding to functionally-determined β*D* individuals (perhaps with canonical forms) in the ‹β› interpretation.

5. Computational Interpretations, Models, and Stored Programs

- Posit that computational interpretation is transitive: If ‹β› has a computational interpretation, so does a representation of ‹α› in ‹β›, and hence ‹α›.
- A structure ‹μ› is an abstract (stored-program) **computational model** if there is a μ*T*-determined universal function, say μ.App(*p*,*x*), such that for every computable function *f(x)* in μ*F*, $\exists p \; \forall x$ [μ.App(*p*,*x*) = *f*(*x*)] is the case for μ*D* denumerable.
- μ.App(*p*,*x*) is a direct computational interpretation of ‹μ› in this case. It is an abstraction of stored-program computation and there are useful examples in practical applications of computers (compiling, interpreting, etc.).
- Church-Turing universality is established by explanation of exactly what the claim is and that other universal models are demonstrable by simulations in a ‹μ›.
- Intractability can be introduced via satisfiability and then P =? NP informally.
- In all of this so far, we remain confined to a world of abstract entities.

6. Interpretations and Representing Worlds

As far as the laws of mathematics refer to reality, they are not certain; and as far as they are certain, they do not refer to reality.

Albert Einstein, 1921

- Emphasize that mathematical structures are not mechanisms.
- Explain that while much about computers can be simulated and verified using logical and mathematical representations, that is still abstraction above the actual electrical and physical nature of digital computers.
- Even the 0s and 1s are abstractions engineered and represented with extraordinary dependability. We might better say that the 0s and 1s are manifest in an intended interpretation.
- When software development and computer science involve theories of structures in the world, we need to emphasize that such theories do not capture the world entire. Fidelity to the natural world and human affairs is an empirical challenge, never merely a formal one.
- Most of all, we must neither foster nor embrace notions that our world is apprehended by mechanism.