

COS 326 So Far and What's Next!

David Walker
Princeton University

326 So Far

Functional Programming. Why?

- An elegant new way to craft algorithms
- A simple platform for developing our understanding of formal methods:
 - Simple semantics, so we can understand what our programs do precisely
 - $(\text{fun } x \rightarrow e) \ v \ \text{---} \rightarrow \ e[v/x]$
 - Often needed to craft clear specifications
 - Basis for sophisticated proof systems (Lean)

What's next: Logic

- Specify and prove program properties, we need to understand the meaning of the specifications we write, which involve more than just functions, and develop precise rules for reasoning
- Concretely, we'll examine two tools: SMT solvers & Lean

Propositional Logic

David Walker
Princeton University

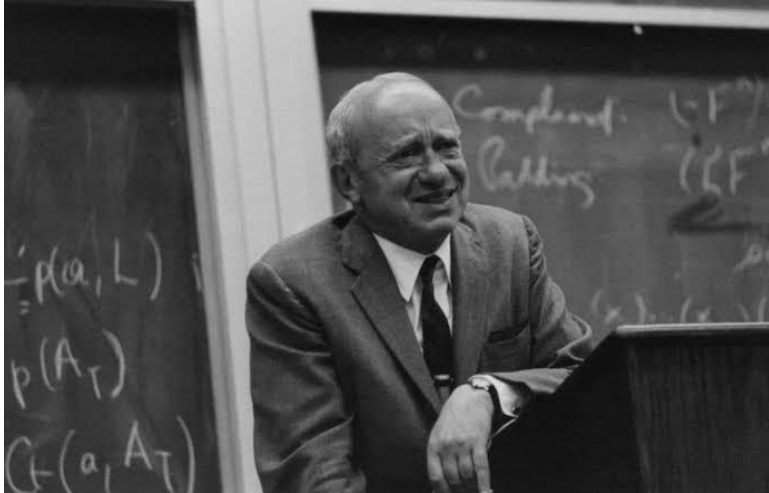
Following notes by Zak Kincaid

What is a logic?

A collection of claims one can make

An understanding of what constitutes a proof or justification of those claims

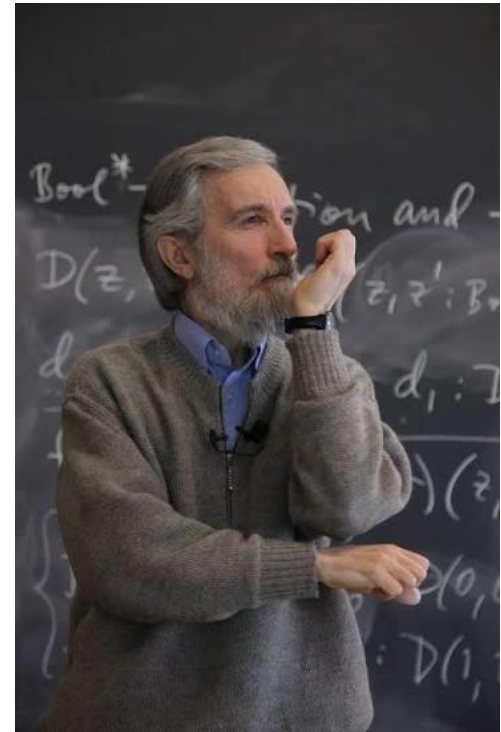
How to Define a Logic



Alfred Tarski (1901-1983)

Logic begins by providing a collection of propositions (formula).

Given some data structure or “model” of the world, we can explain how to assign truth values to propositions
- a **model-theoretic view** of logic

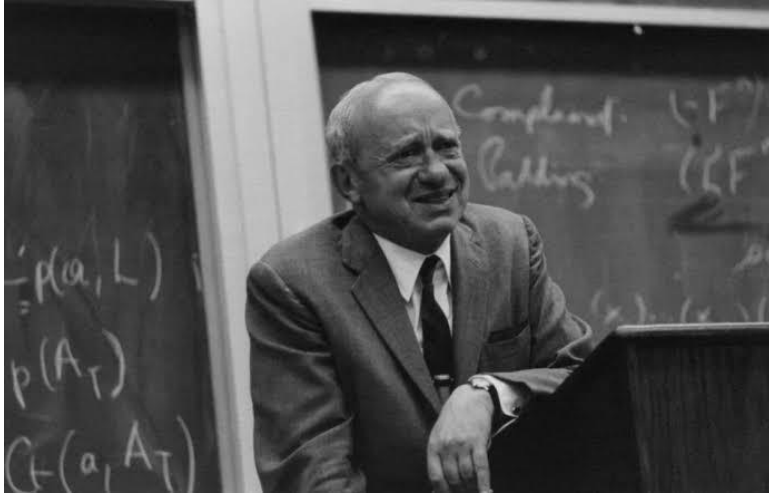


Per Martin-Lof (1942-)

Logic begins with a definition of the judgements (claims) of interest.

Understanding a claim about a proposition arises from understanding what constitutes a verification/proof of that claim
- a **proof-theoretic view** of logic

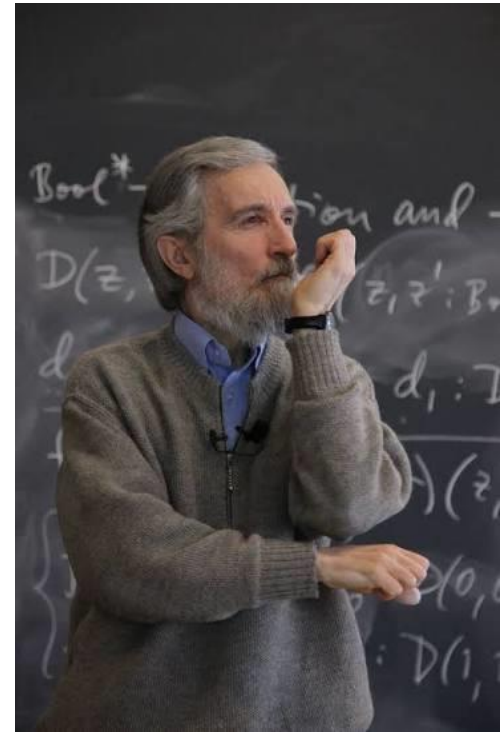
How to Define a Logic



Alfred Tarski (1901-1983)

- a **model-theoretic view** of logic

Understanding this view allows us to use tools such as SAT and SMT solvers to verify assertions and synthesize solutions to complex problems automatically



Per Martin-Löf (1942-)

- a **proof-theoretic view** of logic

Understanding this view allows us to use tools such as Lean and other proof assistants to construct and verify complex mathematical arguments

Defining a Logic: Said Another Way

Syntax

The things we can say: Which phrases (formulae) are well formed.

Semantics

What each phrase means. A class of *structures* that make each formula true or false.

Proof System

A collection of judgements: the things we can say about a formula. For instance, whether it is valid.

A collection of inference rules: rules that tell us what constitutes a proof of that a judgement is valid

Defining a Logic: Said Another Way

today's focus

Syntax

The things we can say: Which phrases (formulae) are well formed.

today's focus

Semantics

What each phrase means. A class of *structures* that make each formula true or false.

Proof System

A collection of judgements: the things we can say about a formula. For instance, whether it is valid.

A collection of inference rules: rules that tell us what constitutes a proof that a judgement is valid

Propositional Logic

In propositional logic, we are interested in reasoning about *indivisible facts*.

Each of these facts is either true or false.

When using propositional logic to solve a real-world problem, we'll:

- identify the real-world facts of interest,
- associate those facts with certain propositions
- represent relationships between propositions using more elaborate formula

Propositional Logic

A given instance of propositional logic is defined by a **propositional signature** – that is, a set of **atomic propositions**.

$$P = \{ p, q, r, \dots \}$$

- p = “it is raining”
- q = “version 5.4.0 of gcc is installed”
- r = “the program halts on input 7”

Formulas: An Inductive Definition

The set of formulas over $\mathcal{P}$ is the *smallest* set containing every atomic proposition, plus **true** and **false**, and closed under the connectives:

(**“Not”**) If F is a formula, then $\neg F$ is a formula

(**“And”**) If F and G are formulas, then $F \wedge G$ is a formula

(**“Or”**) If F and G are formulas, then $F \vee G$ is a formula

(**“Implies”**) If F and G are formulas, then $F \rightarrow G$ is a formula

“Smallest” matters: nothing is a formula unless these rules force it to be

(Whenever we have provided a BNF definition in the past, we have implicitly expected to include the smallest set of syntactic elements that satisfy the definition.)

In BNF Notation

Formula(P) is the set of propositional formulae you can generate starting with propositions in the set P. We let p range over the propositions in P.

Formula(P) :

```
F, G ::= p
      | true  | false
      | ¬F    | F ∧ G  | F ∨ G  | F → G
```

Object Language vs. Metalanguage

To define some **new language**, we have to explain it using an **old one** we hope that everyone knows. The ***Metalanguage*** is the "old one" and the ***Object Language*** is the "new one" we are defining.

Object language

The language being studied.

Here: propositional logic.

Metalanguage

The language we describe it in.

Here: English — and soon, OCaml.

$\wedge$ is a symbol of the object language

the English word “and” is metalanguage.

Syntactic Conventions

When parentheses are omitted, the connectives bind in this order:



Repeated implications associate to the *right*:

$F \rightarrow G \rightarrow H$ means $F \rightarrow (G \rightarrow H)$

(it is not a coincidence
that OCaml Function types
also associate to the right!)

So, for example:

$p \wedge q \rightarrow \neg r \vee p$ means $(p \wedge q) \rightarrow ((\neg r) \vee p)$

Implementing Syntax in OCaml

The grammar becomes a datatype:

```
type 'a formula =  
  | Atom      of 'a  
  | True  
  | False  
  | Not       of 'a formula  
  | And       of 'a formula * 'a formula  
  | Or        of 'a formula * 'a formula  
  | Implies   of 'a formula * 'a formula
```

- `'a` is the type of atomic propositions — the signature $\mathcal{P}$, as a type.
- A value of this type *is* an abstract syntax tree.
- Our metalanguage is now OCaml. The object language has not changed.

Semantics

Syntax says which formulas exist.
Semantics says what they mean.

Structures

A **propositional structure M** over a signature **P** is a function that assigns each proposition a truth value.

$$M : P \rightarrow \{ \text{true}, \text{false} \}$$

M is also called an ***interpretation*** or a ***truth assignment***.

M fixes a truth value for every atomic proposition.

Structures

Given a signature $P = \{p, q\}$, there are exactly four different structures:

```
M1 = { p ↦ true,  q ↦ true  }  
M2 = { p ↦ false, q ↦ true  }  
M2 = { p ↦ true,  q ↦ false }  
M3 = { p ↦ false, q ↦ false }
```

In general, a signature with n atoms has 2^n structures, which is quite a lot. 😊

Satisfaction: $M \models F$

We write $M \models F$ (“M satisfies F”, or “M is a model of F”) when F holds in the structure M.

We write $M \not\models F$ when F does not hold in structure M.

$M \models \text{true}$	<i>always</i>
$M \models \text{false}$	<i>never</i>
$M \models p$	iff $M(p) = \text{true}$
$M \models \neg F$	iff $M \not\models F$
$M \models F \wedge G$	iff $M \models F$ and $M \models G$
$M \models F \vee G$	iff $M \models F$ or $M \models G$
$M \models F \rightarrow G$	iff $M \not\models F$ or $M \models G$

Note: one case per formula (so our definition is complete)

Note: The definition is *compositional*: the meaning of a formula is determined by the *meanings* of its parts, rather than by examining the *syntax* of its parts.

Semantics in OCaml

A structure is a function. Satisfaction is a function too:

```
let rec satisfies (m : 'a -> bool) (f : 'a formula) : bool =  
  match f with  
  | Atom a          -> m a  
  | True            -> true  
  | False           -> false  
  | Not f           -> not (satisfies m f)  
  | And (f, g)      -> satisfies m f && satisfies m g  
  | Or (f, g)       -> satisfies m f || satisfies m g  
  | Implies (f, g)  -> not (satisfies m f) || satisfies m g
```

OCaml checks the completeness of our definition:

The pattern matching is exhaustive, so we give a meaning to all elements of our syntax of formula.

Different Metalanguages

Mathematical/English metalanguage

$M \models p$ defined as follows:

$M \models p$ iff $M(p) = \text{true}$

$M \models \text{true}$ always

$M \models \neg F$ iff $M \not\models F$

$M \models F \wedge G$ iff both ...

$M \models F \vee G$ iff either ...

$M \models F \rightarrow G$ iff $M \not\models F$ or $M \models G$

OCaml metalanguage

```
let rec satisfies m f =  
  match f with  
  
  | Atom a -> m a  
  
  | True -> true  
  
  | Not f -> not (satisfies m f)  
  
  | And (f,g) -> ... && ...  
  
  | Or (f,g) -> ... || ...  
  
  | Implies (f,g) ->
```

Calculating Truth

The semantics gives us a way to calculate the truth of a statement if we are provided a structure.

Calculating Truth

The semantics gives us a way to calculate the truth of a statement if we are provided a structure.

Example: Take the formula and the structure:

$$F = p \vee \neg q$$

$$M = \{ p \mapsto \text{true}, q \mapsto \text{true} \}$$

Calculating Truth

The semantics gives us a way to calculate the truth of a statement if we are provided a structure.

Example: Take the formula and the structure:

$$F = p \vee \neg q \qquad M = \{ p \mapsto \text{true}, \quad q \mapsto \text{true} \}$$

(1) Apply the rule for $\vee$:

$$M \models p \vee \neg q \quad \text{iff} \quad M \models p \quad \text{or} \quad M \models \neg q$$

Calculating Truth

The semantics gives us a way to calculate the truth of a statement if we are provided a structure.

Example: Take the formula and the structure:

$$F = p \vee \neg q \qquad M = \{ p \mapsto \text{true}, \quad q \mapsto \text{true} \}$$

(1) Apply the rule for $\vee$:

$$M \models p \vee \neg q \quad \text{iff} \quad M \models p \quad \text{or} \quad M \models \neg q$$

(2) first possibility: consider whether $M \models p$

- By the rule for propositions p , $M(p) = \text{true}$,
 - so we have $M \models p$
 - so we have $M \models p \vee \neg q$

Calculating Truth

The semantics gives us a way to calculate the truth of a statement if we are provided a structure.

Example: Take the formula and the structure:

$$F = p \vee \neg q$$

$$M = \{ p \mapsto \text{false}, \quad q \mapsto \text{true} \}$$

Calculating Truth

The semantics gives us a way to calculate the truth of a statement if we are provided a structure.

Example: Take the formula and the structure:

$$F = p \vee \neg q \qquad M = \{ p \mapsto \text{false}, \quad q \mapsto \text{true} \}$$

(1) Apply the rule for $\vee$:

$$M \models p \vee \neg q \quad \text{iff} \quad M \models p \quad \text{or} \quad M \models \neg q$$

Calculating Truth

The semantics gives us a way to calculate the truth of a statement if we are provided a structure.

Example: Take the formula and the structure:

$$F = p \vee \neg q$$

$$M = \{ p \mapsto \text{false}, q \mapsto \text{true} \}$$

(1) Apply the rule for $\vee$:

$$M \models p \vee \neg q \quad \text{iff} \quad M \models p \quad \text{or} \quad M \models \neg q$$

$$M \not\models p \quad \text{since } M(p) = \text{false}$$

$$M \models q \quad \text{since } M(q) = \text{true}$$

$$M \not\models \neg q \quad \text{since } M \models q$$


$$M \not\models p \vee \neg q$$

Example: Does $M \models p \vee \neg q$?

Take the formula and the structure:

$$F = p \vee \neg q \qquad M = \{ p \mapsto \text{false}, \quad q \mapsto \text{true} \}$$

Apply the rule for $\vee$:

$$M \models p \vee \neg q \quad \text{iff} \quad M \models p \quad \text{or} \quad M \models \neg q$$

- $M(p) = \text{false}$, so $M \not\models p$.
- $M(q) = \text{true}$, so $M \models q$, hence $M \not\models \neg q$.

Neither disjunct holds, so $M \not\models p \vee \neg q$. M is a **countermodel** of F .

All Four Structures

There are only four structures over $\{p, q\}$, so we can simply check them all:

$M(p)$	$M(q)$	$M \models p \vee \neg q$
true	true	true
true	false	true
false	true	false
false	false	true

- Three structures satisfy $p \vee \neg q$ — they are its **models**.
- One does not — it is a **countermodel**.

Terminology Summary



model $M \models F$

M makes F true.



countermodel $M \not\models F$

M makes F false.



satisfiable some M

F has at least one model.



valid every M

Every structure is a model of F. Also called a tautology.



unsatisfiable no M

No structure is a model of F.

Examples

Satisfiable

- $p \vee \neg q$
- $p \wedge q \wedge r$
- $p \vee \neg p$
- p
- q

Valid

- $p \vee \neg p$
law of the excluded middle
- $((p \rightarrow q) \rightarrow p) \rightarrow p$
Peirce's law

Every valid formula is satisfiable.

Not every satisfiable formula is valid.

$p \vee \neg q$ has three models and one countermodel, so it is **satisfiable but not valid**.

So *invalid* does **not** mean *unsatisfiable*.

Validity and Satisfiability are Dual

Key practical idea:

F is valid if and only if **$\neg F$ is unsatisfiable**

Why?

- F is valid exactly when all structures make it true:
 - ie: no structure makes F false.
- If $\neg F$ is unsatisfiable then there is no model that makes F false

Propositional logic in practice:

- To prove F is valid, ask a propositional logic solver for a model of $\neg F$.
 - Solver can't find a model $\Rightarrow$ F is valid.
 - Solver returns a model $\Rightarrow$ it is a counterexample

Validity and Satisfiability are Dual

Key practical idea:

F is valid if and only if **$\neg F$ is unsatisfiable**

Propositional logic in practice:

- Propositional logic solvers (SAT solvers) operate by looking for a model of F
 - return that model when it is found, or
 - UNSAT
- To prove F is valid, ask a propositional logic solver for a model of $\neg F$.
 - Solver can't find a model $\Rightarrow$ F is valid.
 - Solver returns a model $\Rightarrow$ it is a counterexample

Summary: Propositional Logic

Syntax: $F ::= p \mid \text{true} \mid \text{false} \mid \neg F \mid F \wedge G \mid F \vee G \mid F \rightarrow G$

Semantics:

- A structure M assigns true or false to every atom.
- $M \models F$ means that M satisfies F ; M is a *model* of F
- $M \not\models F$ means that M does not satisfy F ; M is a *countermodel* of F
- If all M satisfy F then F is valid.