

Propositional Logic II: Meta-Theory

**Type Theory and Mechanized Reasoning
Lecture 5**

Introduction

Administrivia

- Assignment 1 is officially due 11:59PM tomorrow.
- Assignment 2 will be released tomorrow as well.
- If you haven't gotten Agda set up on your machine, please do so ASAP.

Objectives

1. Use propositional logic as a setting for learning **important concepts and terms** in logic.
2. Look at what we can say *about* logic, not just with logic.
3. Being discussing **normal forms**, which are used for representing formulas in the context of CS.

Unicode cheatsheet

\rightarrow is `\rightarrow`

\mathbb{N} is `\bN`

\times is `\times`

\wedge is `\and`

λ is `\lambda`

\vee is `\or`

\neg is `\neg`

$_p$ is `\^p`

$::$ is `:::`

\cup is `\cup`

Practice Problem

*Write a function **replace-implies** which recursively replaces $A \rightarrow B$ in **Form** with $\neg A \vee B$.*

(see the starter code for more details)

Agda Tutorial: Pattern Matching

Standard Pattern Matching

```
my-pred :  $\mathbb{N} \rightarrow \mathbb{N}$   
my-pred zero = zero  
my-pred (suc n) = n
```

We define a function in multiple lines in which the argument varies by pattern.

We can automatically split on a variable using **C-c C-c**.

Anonymous Functions

```
foo : List ℕ  
foo = map (λ { x → x * 10 }) (1 :: 2 :: 3 :: [])
```

```
my-pred : ℕ → ℕ  
my-pred = λ { zero → zero  
           ; (suc n) → n }
```

As in most languages, we have lambdas.

We can even pattern match within lambdas.

Case Expressions

`case_of_` : $\{A\ B : \text{Set}\} \rightarrow A \rightarrow (A \rightarrow B) \rightarrow B$
`case` `x` `of` `f` = `f` `x`

Because of this, we don't need special **case** notation.

Casing is just `function application(!)`

Simple Example

```
my-pred :  $\mathbb{N} \rightarrow \mathbb{N}$   
my-pred n = case n of  $\lambda$   
  { zero  $\rightarrow$  zero  
    ; (suc n)  $\rightarrow$  n }
```

Because of this, we can get **OCaml-like** function definitions.

This is nice for simple functions, but it has its drawbacks...

With Abstractions

```
filter : {A : Set} → (A → Bool) → List A → List A
filter p [] = []
filter p (x :: xs) with p x
filter p (x :: xs)   | true  = x :: filter p xs
filter p (x :: xs)   | false = filter p xs
```

The slightly more canonical way of pattern matching on intermediate values is by using **with abstractions**.

demo

Understanding Check

*Write a function **split** of type*

$\{A : \text{Set}\} \rightarrow \{n : \mathbb{N}\} \rightarrow \text{Vec } A \ n \rightarrow \text{Fin } n \rightarrow \text{List } A \times \text{List } A$

which split a vector into two lists given an index.

(the function `Data.Vec.toList` will be useful here.)

Propositional Logic: Recap

Recall: Boolean Connectives

$$x \mathbf{\wedge} y$$

conjunction (and)

Propositional logic is the study of logical (Boolean) **connectives**.

- » *What do connectives mean? How do they affect truth?*
- » *What connectives exist? How many do we need?*
- » *How do connectives interact?*

Recall: Programming Conditionals

```
if is_raining and not is_warm:  
    # some code
```

Propositional logic is the study of **Bools**.

- » *When can I replace one conditional with another?*
- » *Why is there only **and**, **or**, and **not**?*
- » *Why can conditionals short-circuit?*

Recall: Propositional Variables

$$x \wedge y$$

propositional variables

Recall: Propositional Variables

$$x \wedge y$$

propositional variables

We're interested in how propositions *interact* with respect to connectives.

Recall: Propositional Variables

$$x \wedge y$$

propositional variables

We're interested in how propositions *interact* with respect to connectives.

We don't care what the actual propositions are.

Recall: Propositional Variables

$$x \wedge y$$

propositional variables

We're interested in how propositions *interact* with respect to connectives.

We don't care what the actual propositions are.

We think of these variables in the same way as we think of *variables in algebra*.*

Recall: Propositional Variables

$$x \wedge y$$

propositional variables

We're interested in how propositions *interact* with respect to connectives.

We don't care what the actual propositions are.

We think of these variables in the same way as we think of **variables in algebra**.*

*We will assume a countable number of variable symbols like in algebra

Syntax: In Agda

```
data Formula : Set where
  _p : String → Formula
  ¬p _ : Formula → Formula
  _p∧_ : Formula → Formula → Formula
  _p∨_ : Formula → Formula → Formula
  _p→_ : Formula → Formula → Formula
```

The **tree structure** of formulas is implicit in it being an ADT.

At a high level

At a high level

Given an expression $3x + (2y \times 6h^2)$ and values for x , y , and h , we can **compute** the value of the entire expression.

At a high level

Given an expression $3x + (2y \times 6h^2)$ and values for x , y , and h , we can **compute** the value of the entire expression.

Given a *formula* $\neg(x \wedge y) \vee (x \rightarrow \neg z)$ if we know the values of x , y , and z , we can compute the value of the the formula.

At a high level

Given an expression $3x + (2y \times 6h^2)$ and values for x , y , and h , we can **compute** the value of the entire expression.

Given a *formula* $\neg(x \wedge y) \vee (x \rightarrow \neg z)$ if we know the values of x , y , and z , we can compute the value of the the formula.

We think of a propositional variable as having the values of either **true** or **false**.

Valuations (In Mathematical English)

Valuations (In Mathematical English)

Definition. A valuation is a function from all possible propositional values to $\{\text{true}, \text{false}\}$.

Valuations (In Mathematical English)

Definition. A valuation is a function from all possible propositional values to $\{\text{true}, \text{false}\}$.

A valuation v is like a **state of affairs**.

Valuations (In Mathematical English)

Definition. A valuation is a function from all possible propositional values to $\{\text{true}, \text{false}\}$.

A valuation v is like a **state of affairs**.

***The idea.** If we know the state of affairs, we can determine the truth or falsity of any statement.*

Partial Valuations

$$v(x) = \text{true}$$

$$v(y) = \text{false}$$

$$v(z) = \text{false}$$

$$v(_) = \text{false}$$

$$(x \wedge y) \vee z$$

Note. We will typically only care about a **small collection** of variables.

We can assume all **unspecified variables** are assigned to be false.

Evaluation: At a High Level

Evaluation: At a High Level

A valuation function can be **lifted** from propositional variables to arbitrary formulas.

Evaluation: At a High Level

A valuation function can be **lifted** from propositional variables to arbitrary formulas.

This is where we give a formula **meaning** with respect to a state of the affairs.

Evaluation: At a High Level

A valuation function can be **lifted** from propositional variables to arbitrary formulas.

This is where we give a formula **meaning** with respect to a state of the affairs.

We have to say what we want the truth of a statement to be based on its constituent parts.

demo

Evaluation: In English

Evaluation: In English

The `evaluation function` we wrote in Agda for a valuation v is written \bar{v} .

Evaluation: In English

The `evaluation function` we wrote in Agda for a valuation v is written \bar{v} .

We say that a valuation v **makes ϕ true** if $\bar{v}(\phi) = \text{true}$.

Evaluation: In English

The `evaluation function` we wrote in Agda for a valuation v is written \bar{v} .

We say that a valuation v **makes ϕ true** if $\bar{v}(\phi) = \text{true}$.

I leave it as an exercise to write out a "mathy" definition, but the shape will be similar, e.g.,

$$\bar{v}(\neg P) = \begin{cases} \text{false} & \bar{v}(P) = \text{true} \\ \text{true} & \text{otherwise} \end{cases}$$

Propositional Logic: Semantic Notions

Validity and Satisfiability

Validity and Satisfiability

Definition. A formula ϕ is **valid** (written $\models \phi$) if *every* valuation v makes ϕ true, e.g.

$$x \vee \neg x$$

Validity and Satisfiability

Definition. A formula ϕ is **valid** (written $\models \phi$) if *every* valuation v makes ϕ true, e.g.

$$x \vee \neg x$$

We say that ϕ is a **tautology**.

Validity and Satisfiability

Definition. A formula ϕ is **valid** (written $\models \phi$) if *every* valuation v makes ϕ true, e.g.

$$x \vee \neg x$$

We say that ϕ is a **tautology**.

Definition. A formula ϕ is **satisfiable** if there is *some* valuation which makes ϕ true, e.g.

$$x \vee y$$

Example

Proposition. $A \vee \neg A$ is a tautology for any formula A .

Proof. Let's try it.

Validity and Satisfiability

Theorem. ϕ is a tautology if and only if $\neg\phi$ is unsatisfiable.

Proof. Let's try it.

Entailment

Definition. A set of formulas Γ **entails** ϕ (written $\Gamma \models \phi$) if *every* valuation which make *every* formula in Γ true, also makes ϕ true.

Example. $\{x \rightarrow y, y \rightarrow z\} \models x \rightarrow z$

The Deduction Theorem

Theorem. $\Gamma \cup \{\phi\} \models \psi$ if and only if $\Gamma \models \phi \rightarrow \psi$.

Proof. Let's try it.

Understanding Check

True or False: If $\Gamma \models A \vee B$ then $\Gamma \models A$ or $\Gamma \models B$.

Logical Equivalence

Logical Equivalence

Definition. Formulas ϕ and ψ are **logically equivalent** (written $\phi \equiv \psi$) if $\{\phi\} \models \psi$ and $\{\psi\} \models \phi$. That is, all valuations agree on ϕ and ψ .

Logical Equivalence

Definition. Formulas ϕ and ψ are **logically equivalent** (written $\phi \equiv \psi$) if $\{\phi\} \models \psi$ and $\{\psi\} \models \phi$. That is, all valuations agree on ϕ and ψ .

Example. $x \wedge y$ is logically equivalent to $y \wedge x$.

Logical Equivalence

Definition. Formulas ϕ and ψ are **logically equivalent** (written $\phi \equiv \psi$) if $\{\phi\} \models \psi$ and $\{\psi\} \models \phi$. That is, all valuations agree on ϕ and ψ .

Example. $x \wedge y$ is logically equivalent to $y \wedge x$.

Logical equivalence captures when boolean conditionals **express the same thing**.*

Logical Equivalence

Definition. Formulas ϕ and ψ are **logically equivalent** (written $\phi \equiv \psi$) if $\{\phi\} \models \psi$ and $\{\psi\} \models \phi$. That is, all valuations agree on ϕ and ψ .

Example. $x \wedge y$ is logically equivalent to $y \wedge x$.

Logical equivalence captures when boolean conditionals **express the same thing**.*

*This does not account for short-circuiting.

Understanding Check

Show that $A \wedge B \equiv \neg(\neg A \vee \neg B)$.

Standard Logical Equivalences

Distribution:

$$A \vee (B \wedge C) = (A \vee C) \wedge (B \vee C)$$

$$A \wedge (B \vee C) = (A \wedge C) \vee (B \wedge C)$$

DeMorgan's Law:

$$A \wedge B \equiv \neg(\neg A \vee \neg B)$$

$$A \vee B \equiv \neg(\neg A \wedge \neg B)$$

Double Negation Elimination: $\neg(\neg A) \equiv A$

Law of the Excluded Middle: $A \vee \neg A$

And many more...