Propositional Logic: An Introduction

Type Theory and Mechanized Reasoning

Lecture 5

`CAS CS 400`

Introduction

Administrivia

- Assignment 1 is out. Another assignment will be out on Thursday.
- The course has been approved for the semester.
- Details about the final project will be available next week.

Objectives

1. Introduce propositional logic, a simple form of logic for reasoning about Boolean connectives.

2. Use Agda to implement propositional logic.

3. Use propositional logic as a setting for learning important concepts and terms in logic.

Unicode cheatsheet

`→` is `\->`

`ℕ` is `\bN`

`×` is `\times`

`∧` is `\and`

`∨` is `\or`

`¬` is `\neg`

`_ᵖ` is `\^p`

Practice Problem

*Write a function **down** which, given a natural number **n**, returns a vector with the values **n** through **1**.*

*Write a function **up**, which return a vector with the number from **1** to **n**.*

Agda Tutorial: Interaction

Interaction: At a High Level

Interaction: At a High Level

Agda is a strongly typed language, you need to type-check your code frequently.

Interaction: At a High Level

Agda is a strongly typed language, you need to type-check your code frequently.

Agda is a functional language, which means a lot of pattern matching.

Interaction: At a High Level

Agda is a strongly typed language, you need to type-check your code frequently.

Agda is a functional language, which means a lot of pattern matching.

Agda is pure, there are no print statements, so we need to know how to compute values.

The Primary Tool: Holes

```
modulo : ℕ → ℕ → ℕ
modulo = ?
```

The Primary Tool: Holes

```
modulo : ℕ → ℕ → ℕ
modulo = ?
```

Holes are essentially typed TODO items. **To create a hole type "?".**

The Primary Tool: Holes

```
modulo : ℕ → ℕ → ℕ
modulo = ?
```

Holes are essentially typed TODO items. **To create a hole type "?".**

They allow you to build up code piece-by-piece and check code frequently.

The Primary Tool: Holes

```
modulo : ℕ → ℕ → ℕ
modulo = ?
```

Holes are essentially typed TODO items. **To create a hole type "?".**

They allow you to build up code piece-by-piece and check code frequently.

Within a hole, we can determine the types of everything in the environment.

The Primary Tool: Holes

```
modulo : ℕ → ℕ → ℕ
modulo = ?
```

Holes are essentially typed TODO items. **To create a hole type "?".**

They allow you to build up code piece-by-piece and check code frequently.

Within a hole, we can determine the types of everything in the environment.

We can even try to fill in a value (which can have it's own holes).

Standard Workflow

Standard Workflow

1. (C-c C-l) Write some Agda code with a hole, then load it.

Standard Workflow

1. (C-c C-l) Write some Agda code with a hole, then load it.

2. (C-c C-,) See what Agda wants you to fill the hole with.

Standard Workflow

1. (C–c C–l) Write some Agda code with a hole, then load it.

2. (C–c C–,) See what Agda wants you to fill the hole with.

3. (C–c C–c) Pattern match on an input if necessary.

Standard Workflow

1. (C–c C–l) Write some Agda code with a hole, then load it.

2. (C–c C–,) See what Agda wants you to fill the hole with.

3. (C–c C–c) Pattern match on an input if necessary.

4. (C–c C–SPACE) write a definition and try to fill it in.

Standard Workflow

1. (C-c C-l) Write some Agda code with a hole, then load it.
2. (C-c C-,) See what Agda wants you to fill the hole with.
3. (C-c C-c) Pattern match on an input if necessary.
4. (C-c C-SPACE) write a definition and try to fill it in.
5. Rinse and repeat.

Interaction Cheat Sheet

`C-c C-l`	`load file`
`C-c C-,`	`check type in hole`
`C-c C-c`	`pattern match within hole`
`C-c C-SPACE`	`try to fill in hole`
`C-c C-n`	`compute value of term`

demo

Propositional Logic: Motivation

What is logic?

What is logic?

Logic is a formalization of language.

What is logic?

Logic is a formalization of language.

This means we need to specify two things:

syntax	what things can I write down?
semantics	what do those things mean?

What is logic?

Logic is a formalization of language.

This means we need to specify two things:

syntax	what things can I write down?
semantics	what do those things mean?

(this is also what you need for a programming language)

What does syntax mean?

sequence of characters → sequence of lexemes → sequence of tokens → parse tree

We're not going to focus on *syntax or parsing* in this course.

We will almost always presume we already have a parse tree.

What does syntax mean?

sequence of characters → sequence of lexemes → sequence of tokens → parse tree

`[(, X, □, a, n, d, □, Y, )]`

We're not going to focus on *syntax or parsing* in this course.

We will almost always presume we already have a parse tree.

What does syntax mean?

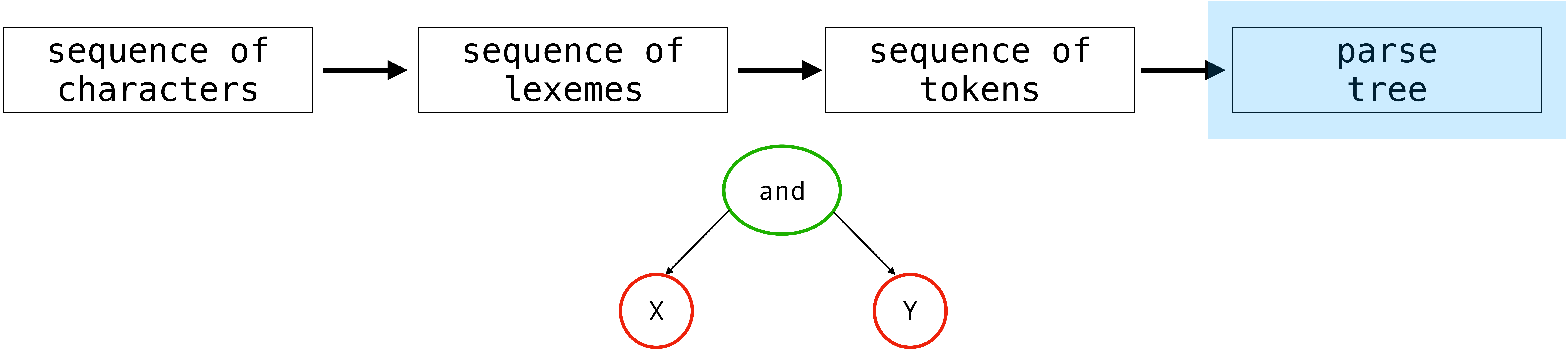
sequence of characters → sequence of lexemes → sequence of tokens → parse tree

```
[sym '(', var 'X', con 'and', var 'Y', sym ')']
```

We're not going to focus on *syntax or parsing* in this course.

We will almost always presume we already have a parse tree.

What does syntax mean?



We're not going to focus on *syntax or parsing* in this course.

We will almost always presume we already have a parse tree.

What does syntax mean?

sequence of characters → sequence of lexemes → sequence of tokens → parse tree

$$x \wedge y$$

We're not going to focus on *syntax or parsing* in this course.

We will almost always presume we already have a parse tree.

What does meaning mean?

$$\begin{array}{l} v(x) = \text{true} \\ v(y) = \text{true} \end{array} \implies v(x \wedge y) = \text{true}$$

What does meaning mean?

$$\begin{array}{l} v(x) = \text{true} \\ v(y) = \text{true} \end{array} \implies v(x \wedge y) = \text{true}$$

```
Very interesting question from a philosophical
perspective.
```

What does meaning mean?

$$\left.\begin{array}{l} v(x) = \text{true} \\ v(y) = \text{true} \end{array}\right. \implies v(x \wedge y) = \text{true}$$

Very interesting question from a philosophical perspective.

In classical logic (what we will consider today) meaning will mean ***truth***.

What does meaning mean?

$$\begin{aligned} v(x) &= \text{true} \\ v(y) &= \text{true} \end{aligned} \implies v(x \wedge y) = \text{true}$$

Very interesting question from a philosophical perspective.

In classical logic (what we will consider today) meaning will mean ***truth***.

In intuitionistic logic, meaning will be something different.

Propositions

Propositions

Some facts seem contingent on the world:

» It is raining.
» It is cold.

Propositions

Some facts seem contingent on the world:

» It is raining.
» It is cold.

Some facts seem unassailable:

» If it is raining and it is cold then it is raining.

Propositions

Some facts seem contingent on the world:

- It is raining.
- It is cold.

Some facts seem unassailable:

- If it is raining and it is cold then it is raining.

Why is this?

Propositions

Some facts seem contingent on the world:

» It is raining.
» It is cold.

Some facts seem unassailable:

» If it is raining and it is cold then it is raining.

proposition proposition proposition

Why is this?

Propositions

Some facts seem contingent on the world:

- It is raining.
- It is cold.

Some facts seem unassailable:

- $\underbrace{\texttt{If}}_{\text{connective}}\ \overbrace{\texttt{it is raining}}^{\text{proposition}}\ \underbrace{\texttt{and}}_{\text{connective}}\ \overbrace{\texttt{it is cold}}^{\text{proposition}}\ \underbrace{\texttt{then}}_{\text{connective}}\ \overbrace{\texttt{it is raining}}^{\text{proposition}}.$

Why is this?

Boolean Connectives

$$x \boxed{\land} y$$

conjunction (and)

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

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

Propositional Logic and 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?*

Propositional Logic: Syntax

Propositional Variables

$$\boxed{x} \land \boxed{y}$$

propositional variables

Propositional Variables

$$\boxed{x} \wedge \boxed{y}$$

propositional variables

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

Propositional Variables

$$\boxed{x} \wedge \boxed{y}$$

propositional variables

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

We don't care what the actual propositions are.

Propositional Variables

$$\boxed{x} \wedge \boxed{y}$$

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 was as we think of variables in algebra.*

Propositional Variables

$$\boxed{x} \wedge \boxed{y}$$

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 was as we think of variables in algebra.*

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

Syntax: In (Mathematical) English

Syntax: In (Mathematical) English

Definition. A **formula** is defined inductively as follows:

Syntax: In (Mathematical) English

Definition. A **formula** is defined inductively as follows:

- A propositional variable x is a formula.

Syntax: In (Mathematical) English

Definition. A **formula** is defined inductively as follows:

- A propositional variable x is a formula.
- If P is a formula then so it $\neg P$.*

Syntax: In (Mathematical) English

Definition. A **formula** is defined inductively as follows:

- A propositional variable x is a formula.
- If P is a formula then so it $\neg P$.*
- If P and Q are formulas then so are $P \wedge Q$ and $P \vee Q$ and $P \rightarrow Q$.

Syntax: In (Mathematical) English

Definition. A **formula** is defined inductively as follows:

- A propositional variable x is a formula.
- If P is a formula then so it $\neg P$.*
- If P and Q are formulas then so are $P \wedge Q$ and $P \vee Q$ and $P \rightarrow Q$.

*P is a *meta-variable*. It refers to an arbitrary formula. It is not the same as a propositional variable.

Examples

$$P \wedge (Q \wedge \neg Z)$$

$$A \to (B \to (\neg C \vee B))$$

$$A \wedge (A \wedge (A \wedge A))$$

Note. Parentheses are *meta-syntactical*. Remember that these formulas are shorthand for trees.

Example (More formally)

$$P \land (Q \land \neg Z)$$

Example (More formally)

$$P \wedge (Q \wedge \neg Z)$$

Z is a variable so Z is a formula.

Example (More formally)

$$P \wedge (Q \wedge \neg Z)$$

Z is a variable so Z is a formula.

Z is a formula so $\neg Z$ is a formula.

Example (More formally)

$$P \wedge (Q \wedge \neg Z)$$

Z is a variable so Z is a formula.

Z is a formula so $\neg Z$ is a formula.

Q is a variable so Q is a formula.

Example (More formally)

$$P \wedge (Q \wedge \neg Z)$$

Z is a variable so Z is a formula.

Z is a formula so $\neg Z$ is a formula.

Q is a variable so Q is a formula.

Q is a formula and $\neg Z$ is a formula, so $Q \wedge \neg Z$ is a formula.

Example (More formally)

$$P \wedge (Q \wedge \neg Z)$$

Z is a variable so Z is a formula.

Z is a formula so $\neg Z$ is a formula.

Q is a variable so Q is a formula.

Q is a formula and $\neg Z$ is a formula, so $Q \wedge \neg Z$ is a formula.

P is a variable so P is a formula.

Example (More formally)

$$P \wedge (Q \wedge \neg Z)$$

Z is a variable so Z is a formula.

Z is a formula so $\neg Z$ is a formula.

Q is a variable so Q is a formula.

Q is a formula and $\neg Z$ is a formula, so $Q \wedge \neg Z$ is a formula.

P is a variable so P is a formula.

P is a formula and $Q \wedge \neg Z$ is a formula so $P \wedge (Q \wedge \neg Z)$ is a formula.

Informal Meaning

$\neg \equiv$ "not"

$\wedge \equiv$ "and"

$\vee \equiv$ "or"

$\rightarrow \equiv$ "implies"

Informal Meaning

$\neg \equiv$ "not"

$\land \equiv$ "and"

$\lor \equiv$ "or"

$\rightarrow \equiv$ "implies"

R It's raining

C It's cold

$(R \land C) \rightarrow R$ If it's raining and cold then it's raining

Understanding Check:
English to Formula

Syntax: In Agda

```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.

Syntax: What's Next?

Remember. We haven't actually given formulas *meaning*. That is, we haven't given a semantics.

But we can ask about:

» Function on formulas
» Transformations of formulas

Example: Depth (In Mathematical English)

Definition. The **depth** of a formula is defined as the depth of its corresponding tree:

$$d(x) = 0$$
$$d(\neg P) = 1 + d(P)$$
$$d(P \square Q) = \max(d(P), d(Q)) + 1$$

where $\square$ is any one of '$\wedge$' or '$\vee$' or '$\rightarrow$'.

Example: Depth (In Agda)

```
Let's do a demo.
```

Propositional Logic: Semantics

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 $\{\mathsf{true}, \mathsf{false}\}$.

Valuations (In Mathematical English)

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

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 $\{\mathsf{true}, \mathsf{false}\}$.

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.

Valuation (In Agda)

```
Let's do a demo.
```

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.

Evaluation: In Agda

```
Let's do a demo.
```

Evaluation: In English

Evaluation: In English

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

Evaluation: In English

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

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

Evaluation: In English

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

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

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

$$\overline{v}(\neg P) = \begin{cases} \mathsf{false} & \overline{v}(P) = \mathsf{true} \\ \mathsf{true} & \texttt{otherwise} \end{cases}$$

Understanding Check: *Evaluation*

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 $\vDash \phi$) if *every* valuation v makes ϕ true, e.g.

$$x \vee \neg x$$

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

$$x \vee y$$

Understanding Check: Validity

Understanding Check:
Validity

Entailment

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

Example. $\{x \to y, y \to z\} \vDash x \to z$

Logical Equivalence

Logical Equivalence

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

Logical Equivalence

Definition. Formulas ϕ and ψ are **logically equivalent** if $\{\phi\} \vDash \psi$ and $\{\psi\} \vDash \phi$. 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** if $\{\phi\} \vDash \psi$ and $\{\psi\} \vDash \phi$. 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** if $\{\phi\} \vDash \psi$ and $\{\psi\} \vDash \phi$. 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: Logical Equivalence

Understanding Check:
Logical Equivalence

Propositional Logic: Functional Completeness

What is a connective?

What is a connective?

Consider the exclusive–or operations $P \otimes Q$.

What is a connective?

Consider the exclusive–or operations $P \otimes Q$.

Why didn't we include this in our presentation of propositional logic?

What is a connective?

Consider the exclusive-or operations $P \otimes Q$.

Why didn't we include this in our presentation of propositional logic?

Theorem. For any propositions P and Q, $P \otimes Q$ is logically equivalent to $(P \wedge Q) \vee (\neg P \vee \neg Q)$.

What is a connective?

Consider the exclusive–or operations $P \otimes Q$.

Why didn't we include this in our presentation of propositional logic?

Theorem. For any propositions P and Q, $P \otimes Q$ is logically equivalent to $(P \wedge Q) \vee (\neg P \vee \neg Q)$.

A connective is a boolean function.

Boolean Functions

Boolean Functions

Definition. An n–variate boolean function is a function of the form

$$f : \{\mathsf{true}, \mathsf{false}\}^n \to \{\mathsf{true}, \mathsf{false}\}$$

Boolean Functions

Definition. An n–variate boolean function is a function of the form

$$f : \{\mathsf{true}, \mathsf{false}\}^n \to \{\mathsf{true}, \mathsf{false}\}$$

f is **represented** by a formula ϕ on with propositional variables $x_1, \ldots, x_n$ if for any valuation v,

$$f(v(x_1), \ldots, v(x_n)) = \overline{v}(\phi)$$

Functional Completeness

Theorem. Every n–variate boolean function is represented by a formula.

Let's try to prove it...