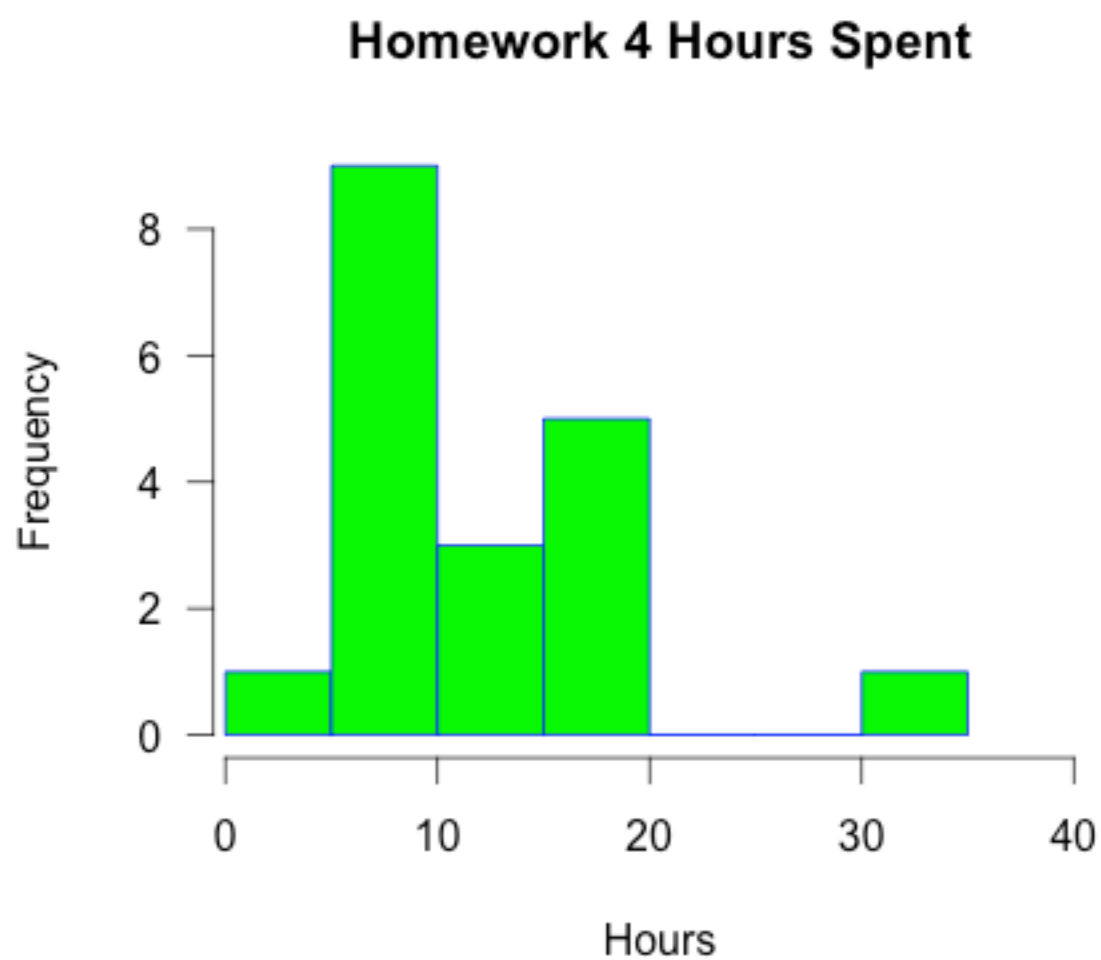
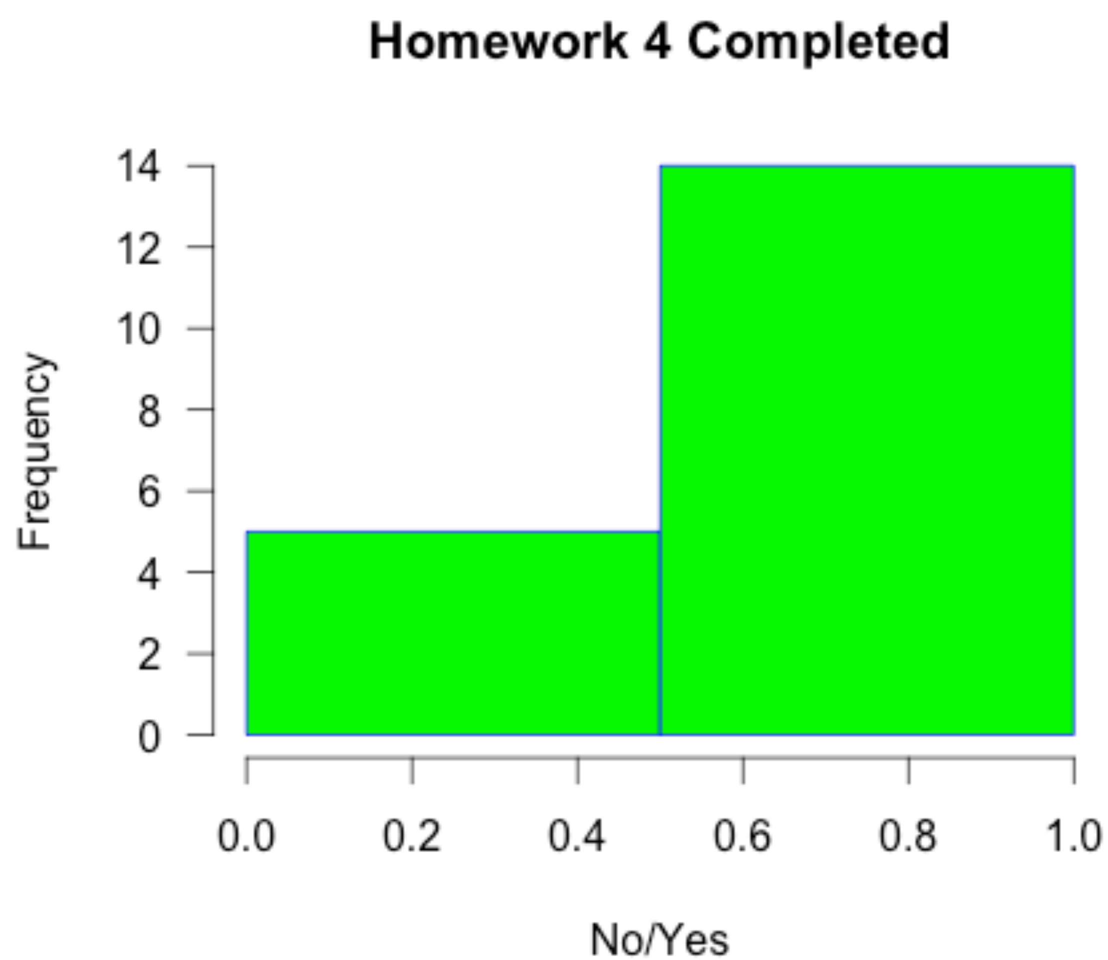
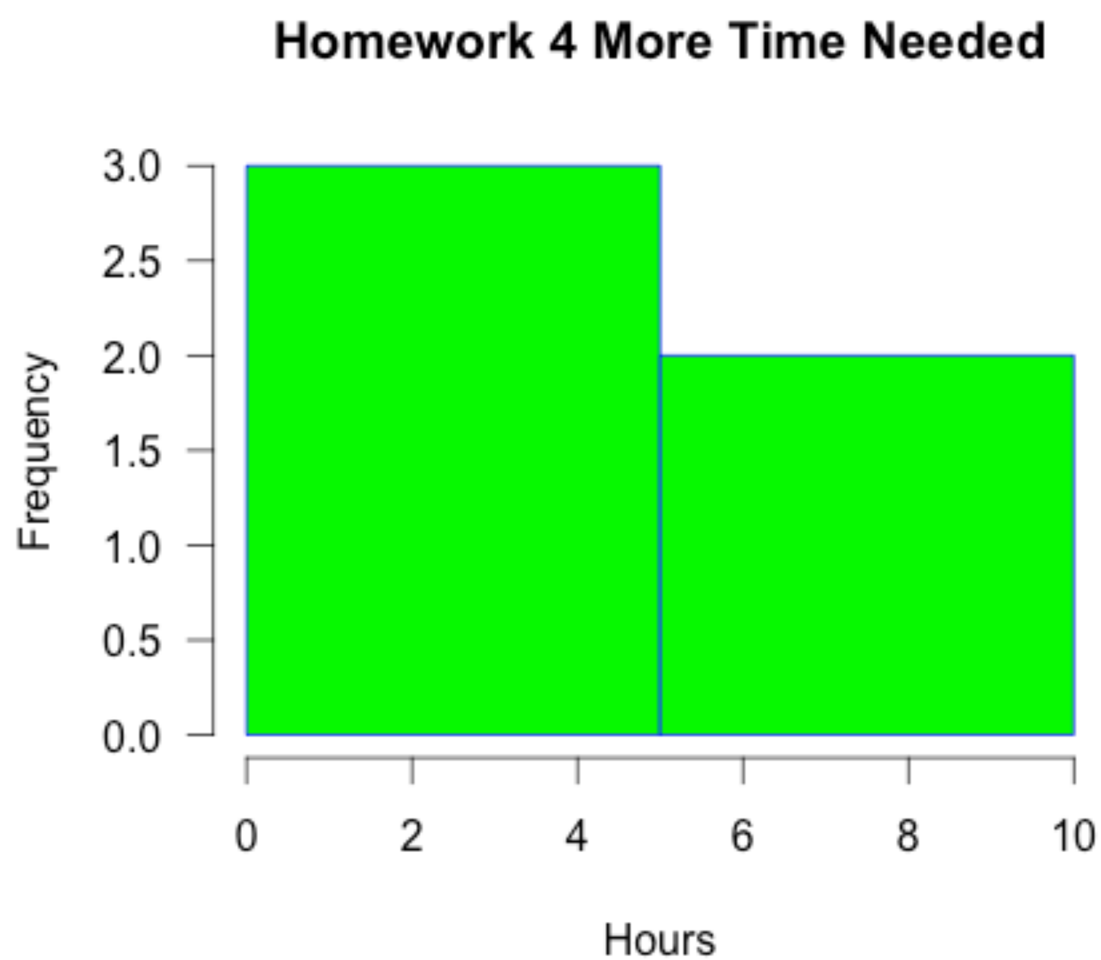
Comp 311
Functional Programming

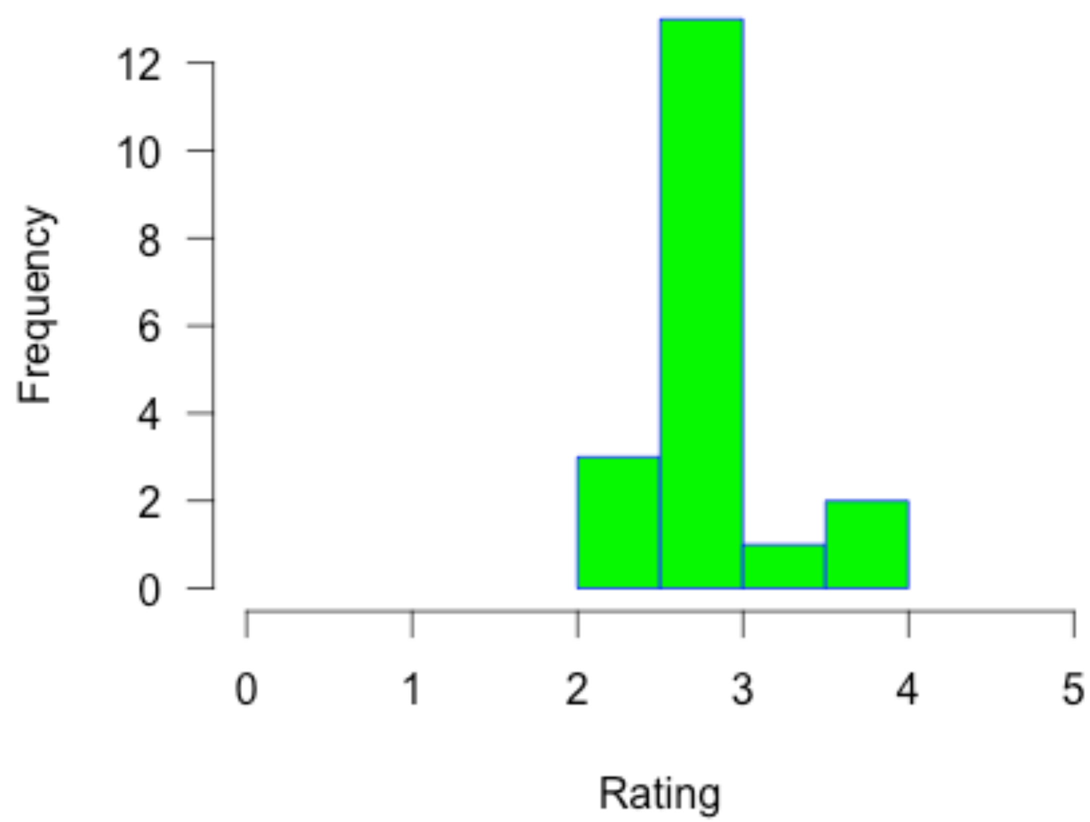
Eric Allen, PhD
Vice President, Engineering
Two Sigma Investments, LLC

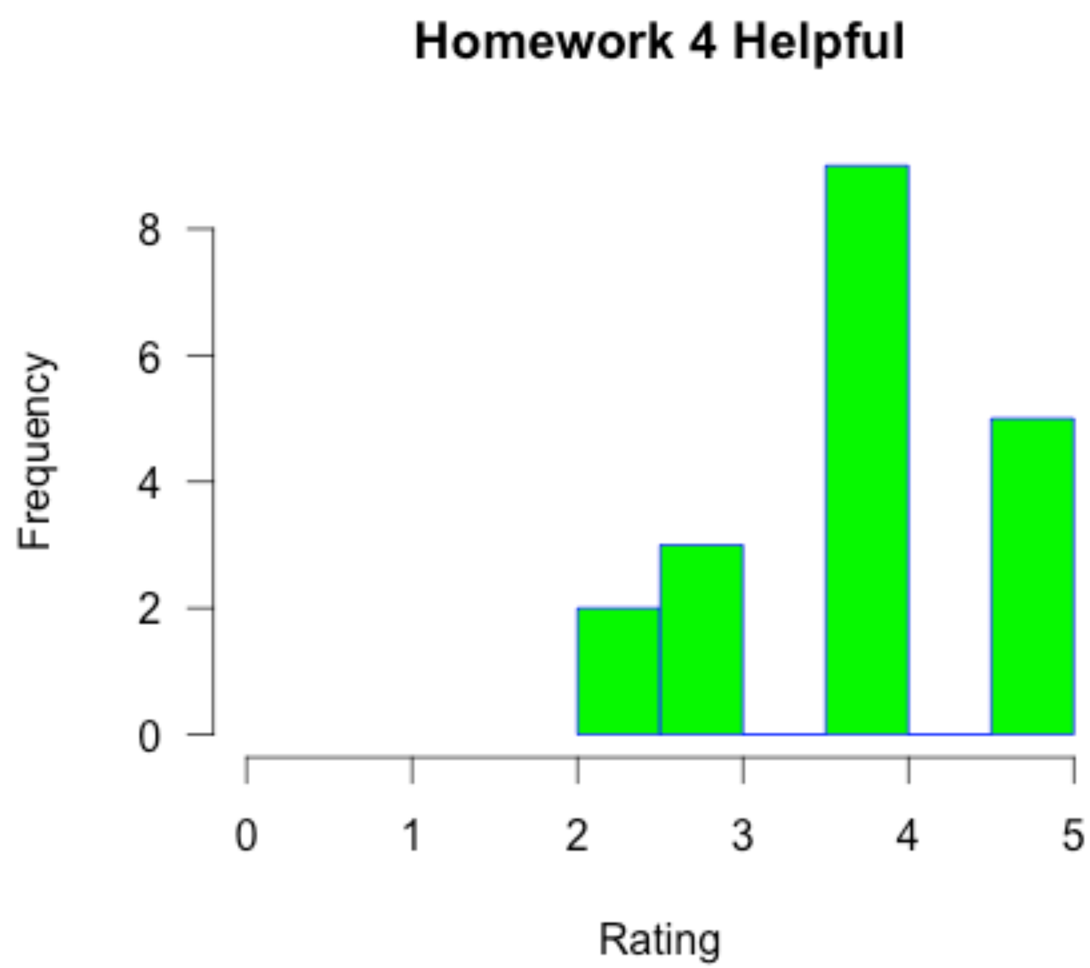

Homework 4 Hours Spent
Frequency
8
6
4
2
0
0
10
20
30
40
Hours

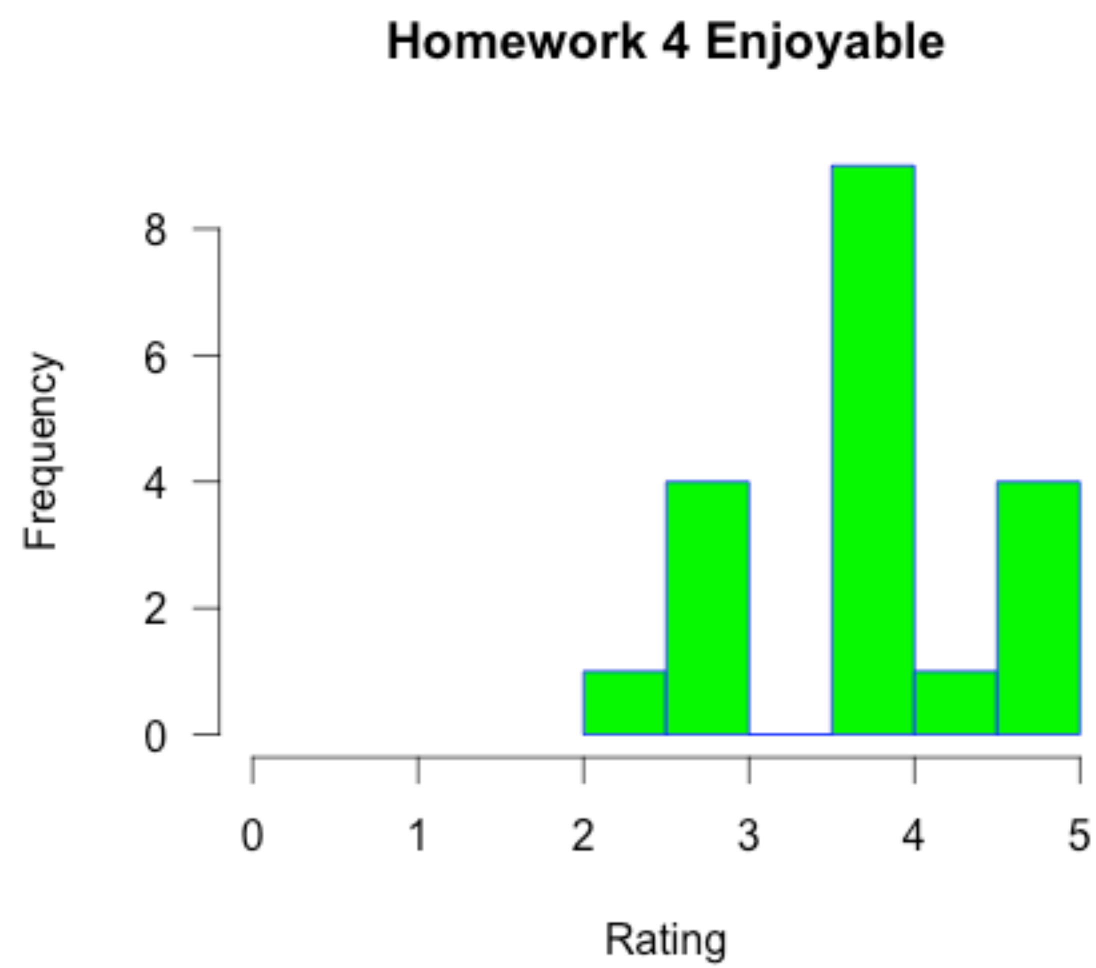

Homework 4 Completed
Frequency
0
2
4
6
8
10
12
14
0.0
0.2
0.4
0.6
0.8
1.0
No/Yes

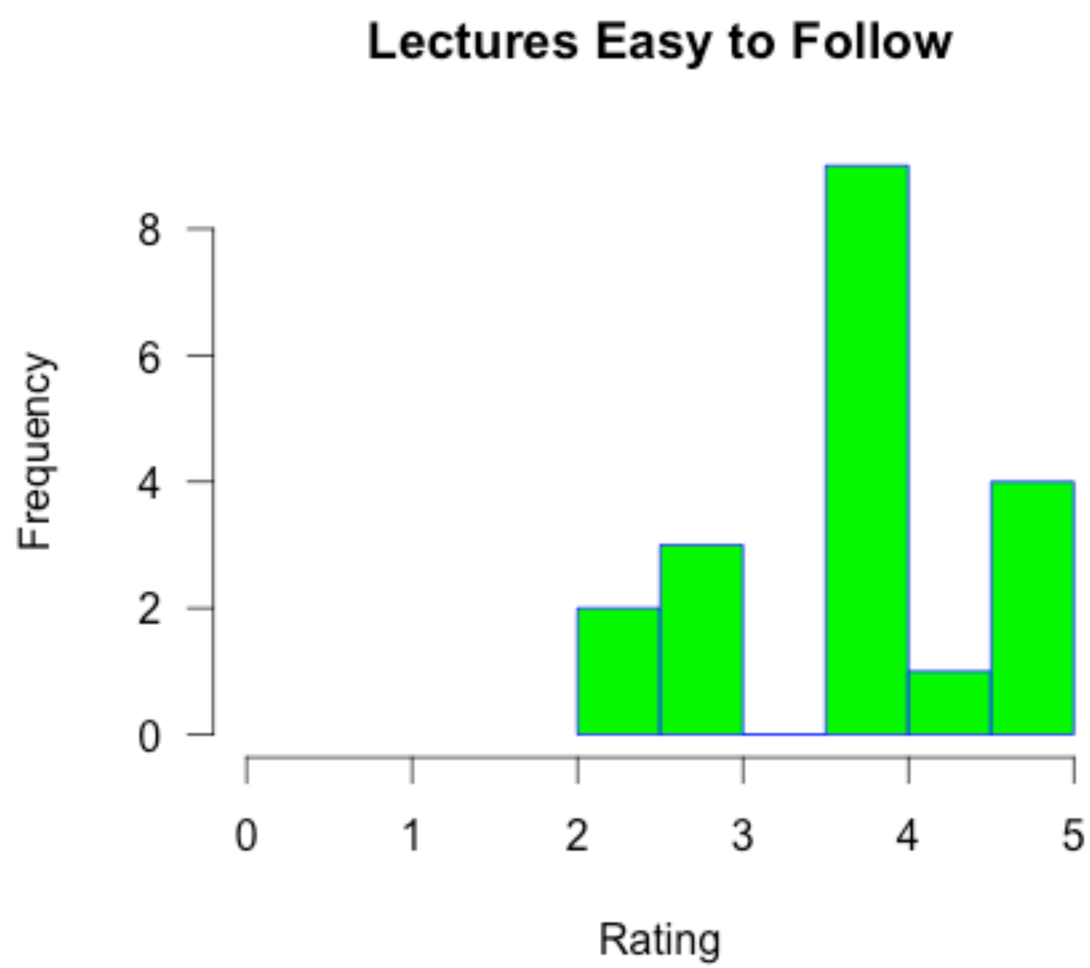

Homework 4 More Time Needed
Frequency
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
2
4
6
8
10
Hours

Homework 4 Workload




Homework 4 Helpful
Frequency
0
2
4
6
8
0
1
2
3
4
5
Rating


Homework 4 Enjoyable
Frequency
0
2
4
6
8
0
1
2
3
4
5
Rating


Lectures Easy to Follow
Frequency
0
2
4
6
8
0
1
2
3
4
5
Rating


Class Pace
Frequency
15
10
5
0
0
1
2
3
4
5
Rating


Class Enjoyable
Frequency
8
6
4
2
0
0
1
2
3
4
5
Rating

Comments and Actions

- Lectures are easy to follow but then it is difficult to know how to apply the material to new situations
  - Worksheets? Smaller homeworks?
- Not enough practice with types
  - Some dealt with in Homework 6
  - Add an additional homework with a type-heavy component

Mechanical Proof Checking

Syntax of Propositional Logic

$$
\begin{array}{rcl}
S & ::= & x \\
& | & S \wedge S \\
& | & S \vee S \\
& | & S \rightarrow S \\
& | & \neg S
\end{array}
$$

Factory Methods for Construction

```
case object Formulas {
  def evar(name: String): Formula
  def and(left: Formula, right: Formula): Formula
  def or(left: Formula, right: Formula): Formula
  def implies(left: Formula, right: Formula): Formula
  def not(body: Formula): Formula
}
```

Sequents

$$S* \vdash S$$

Sequents

- Sequents consist of two parts:
  - The *antecedents* to the left of the turnstile
  - The *consequent* to the right of the turnstile
  - Example:

$$\{p,\ \ q,\ \ \neg r, p \to r\} \vdash \neg p$$

Sequents

- When the set of antecedents consists of a single formula, we often elide the enclosing braces:

$$\{p\} \vdash p$$

- is equivalent to:

$$p \vdash p$$

Inference Rules

$$\frac{\Gamma \vdash p \quad \Delta \vdash q}{\Gamma \cup \Delta \vdash p \wedge q} \text{ AND-INTRO}$$

Inference Rules: General Form

$$\frac{Q*}{Q}$$

Inference Rules

$$\frac{\Gamma \vdash p \wedge q}{\Gamma \vdash p} \textsc{And-Elim-Left}$$

Inference Rules

$$\frac{\Gamma \vdash p \wedge q}{\Gamma \vdash q} \textsc{And-Elim-Right}$$

Inference Rules

$$\frac{\Gamma \vdash p}{\Gamma \vdash p \lor q} \textsc{Or-Intro-Left}$$

Inference Rules

$$\frac{\Gamma \vdash p}{\Gamma \vdash q \vee p}\ \textsc{Or-Intro-Right}$$

Inference Rules

$$\frac{\Gamma \vdash p \vee q \quad \Gamma' \cup \{p\} \vdash r \quad \Gamma'' \cup \{q\} \vdash r}{\Gamma \cup \Gamma' \cup \Gamma'' \vdash r} \textsc{Or-Elim}$$

Inference Rules

$$\frac{\Gamma \cup \{p\} \vdash q \quad \Gamma' \cup \{p\} \vdash \neg q}{\Gamma \cup \Gamma' \vdash \neg p}\ \textsc{Neg-Intro}$$

Inference Rules

$$\frac{\Gamma \vdash \neg\neg p}{\Gamma \vdash p} \textsc{Neg-Elim}$$

Inference Rules

$$\frac{\Gamma \cup \{p\} \vdash q}{\Gamma \vdash p \to q} \textsc{Implies-Intro}$$

Inference Rules

$$\frac{\Gamma \vdash p \rightarrow q \quad \Gamma' \vdash p}{\Gamma \cup \Gamma' \vdash q} \textsc{Implies-Elim}$$

Inference Rules

$$\frac{}{p \vdash p} \textsc{Identity}$$

Inference Rules

$$\frac{}{\Gamma \cup \{p\} \vdash p}\ \textsc{Assumption}$$

Inference Rules

$$\frac{\Gamma \vdash p}{\Gamma \cup \{q\} \vdash p}\ \textsc{Generalization}$$

Example Proof 1

$$\dfrac{\dfrac{}{p \vdash p}\ \textsc{Identity}}{\emptyset \vdash p \to p}\ \textsc{Implies-Intro}$$

Example Proof 2

$$\dfrac{\dfrac{}{p \rightarrow q \vdash p \rightarrow q}\ \textsc{Identity} \qquad \dfrac{}{p \vdash p}\ \textsc{Identity}}{\{p,\ p \rightarrow q\} \vdash q}\ \textsc{Implies-Elim}$$

Example Proof 3

$$
\dfrac{
  \dfrac{\dfrac{}{p \land \neg p \vdash p \land \neg p}\ \textsc{Identity}}{p \land \neg p \vdash p}\ \textsc{And-Elim-Left}
  \qquad
  \dfrac{\dfrac{}{p \land \neg p \vdash p \land \neg p}\ \textsc{Identity}}{p \land \neg p \vdash \neg p}\ \textsc{And-Elim-Right}
}{\emptyset \vdash \neg(p \land \neg p)}\ \textsc{Neg-Intro}
$$

Rule Application

```
case object Rules {
  def identity(p: Formula): Sequent
  def assumption(s: Sequent): Sequent
  def generalization(p: Formula)(s: Sequent): Sequent
  def andIntro(left: Sequent, right: Sequent): Sequent
  def andElimLeft(s: Sequent): Sequent
  def andElimRight(s: Sequent): Sequent
  def orIntroLeft(p: Formula)(s: Sequent): Sequent
  def orIntroRight(p: Formula)(s: Sequent): Sequent
  def orElim(s0: Sequent, s1: Sequent, s2: Sequent): Sequent
  def negIntro(p: Formula)(s0: Sequent, s1: Sequent): Sequent
  def negElim(s: Sequent): Sequent
  def impliesIntro(s: Sequent): Sequent
  def impliesElim(p: Formula)(s: Sequent): Sequent
}
```

The Curry-Howard Isomorphism

Simply Typed Expressions

```
E ::= x
    | 0 | 1 | 2…
    | true | false
    | (x:T) => E
    | E(E)
```

Simple Types

```
T ::= Int
    | Boolean
    | T => T
```

Simple Type Assertions

`E:T`

Simple Type Assertions

```
0:Int
```

Simple Type Assertions

```
true:Boolean
```

Simple Type Assertions

```
(x:Int) => x : Int => Int
```

Simple Type Assertions

```
x:Boolean
```

Assertions Within a Type Environment

$$\texttt{\{x:Boolean\}} \vdash \texttt{x:Boolean}$$

Rules for Checking the Type of an Expression

$$\frac{n \in \texttt{IntLiteral}}{\Gamma \vdash \texttt{n:Int}}\ \textsc{T-Int}$$

Rules for Checking the Type of an Expression

$$\frac{}{\Gamma \vdash \texttt{true:Boolean}} \text{ T-TRUE}$$

$$\frac{}{\Gamma \vdash \texttt{false:Boolean}} \text{ T-FALSE}$$

Rules for Checking the Type of an Expression

$$\frac{\Gamma \cup \{\texttt{x:S}\} \vdash \texttt{E:T}}{\Gamma \vdash \texttt{(x:S)=>E : S=>T}} \ \text{T-ABS}$$

Rules for Checking the Type of an Expression

$$\frac{\Gamma \vdash \mathtt{E:S=>T} \quad \Gamma \vdash \mathtt{E'}\mathtt{:S}}{\Gamma \vdash \mathtt{E(E')}\mathtt{:T}} \text{ T-App}$$

Contrast with Implies-Intro For Propositional Logic

$$\frac{\Gamma \cup \{p\} \vdash q}{\Gamma \vdash p \rightarrow q} \textsc{Implies-Intro}$$

$$\frac{\Gamma \cup \{\texttt{x:S}\} \vdash \texttt{E:T}}{\Gamma \vdash \texttt{(x:S)=>E : S=>T}} \textsc{T-Abs}$$

Contrast with Implies-Intro For Propositional Logic

$$\frac{\Gamma \cup \{p\} \vdash q}{\Gamma \vdash p \rightarrow q}\ \textsc{Implies-Intro}$$

$$\frac{\Gamma \cup \{\text{[redacted]}\ \texttt{S}\} \vdash \text{[redacted]}\ \texttt{T}}{\Gamma \vdash \text{[redacted]}\ \texttt{S=>T}}\ \textsc{T-Abs}$$

Contrast with Implies-Elim From Propositional Logic

$$\frac{\Gamma \vdash p \rightarrow q \quad \Gamma' \vdash p}{\Gamma \cup \Gamma' \vdash q} \textsc{Implies-Elim}$$

$$\frac{\Gamma \vdash \texttt{E:S=>T} \quad \Gamma \vdash \texttt{E}'\texttt{:S}}{\Gamma \vdash \texttt{E(E}'\texttt{):T}} \textsc{T-App}$$

Contrast with Implies-Elim From Propositional Logic

$$\frac{\Gamma \vdash p \rightarrow q \qquad \Gamma' \vdash p}{\Gamma \cup \Gamma' \vdash q} \; \textsc{Implies-Elim}$$

$$\frac{\Gamma \vdash \square \; \texttt{S=>T} \qquad \Gamma \vdash \square \; \texttt{S}}{\Gamma \vdash \square \; \texttt{T}} \; \textsc{T-App}$$

Types and Propositions

- We can think of the types in our simple type system as corresponding to propositions:
  - Primitive types (Boolean, Int) correspond to simple propositions (`p, q`)
  - Arrow types correspond to logic implication:

    `p -> q, (p -> (q -> r))`, etc.

Types and Propositions

- For each syntactic form of expression, there is exactly one form rule that contains that syntactic form as its result
- Example:

$$\frac{\Gamma \cup \{\texttt{x:S}\} \vdash \texttt{E:T}}{\Gamma \vdash \texttt{(x:S)=>E : S=>T}} \text{ T-}\textsc{Abs}$$

Types and Propositions

- If we wish to use type rules to prove that an expression has a specific type
  - We can start with the expression, and apply the rules backwards:

$$\dfrac{\dfrac{}{\texttt{x:T} \vdash \texttt{x:T}}\ \textsc{T-Identity}}{\emptyset \vdash \texttt{(x:T) => x : T => T}}\ \textsc{T-Abs}$$

Types and Propositions

- While working backwards with expressions, there is only one choice at each step
- Thus a well-typed expression E entirely determines the form of the proof that E:T
- But the proof of E:T in our type system is equivalent to a proof of T in propositional logic

Types and Propositions

- So, E effectively encodes a proof of type T, thought of as a proposition
- Checking the type T of an expression E is equivalent to proving the validity of T

The Curry-Howard Isomorphism

- This deep correspondence between types and logical assertions is known as the *Curry-Howard Isomorphism*
- This correspondence goes far beyond just propositional logic, extending to predicate calculus, modal logic, etc.
- This leads to the surprising result that the arrow in arrow types is really just the implication symbol from propositional logic!