An introduction to denotational semantics of logic

Samuel Mimram
December 10, 2020
I will try to introduce the concepts of semantics and its uses in logic.

The presentation may be more oriented to combinatorists in the audience: I suppose very little on your background in logic.

You are of course very welcome to ask questions.
What am I computing?

We should remember that programs are *not* computing functions.

It is just a bunch of electric currents going through wires.
What am I computing?

We should remember that programs are *not* computing functions.

It is just a bunch of electric currents going through wires.

In order to give a meaning to those we have to *interpret* them as functions.
What am I computing?

We should remember that programs are not computing functions.

It is just a bunch of electric currents going through wires.

In order to give a meaning to those we have to interpret them as functions.

Since we have that

\[
\text{PROGRAM} = \text{PROOF}
\]

(this is the Curry-Howard correspondence) we can play the same game for logics.
This is quite useful and productive:

- semantics helps to understand better the properties of logics
- logics helps to find the common structures behind the models
Part I

Boolean logic
The boolean semantics of logic

In fact, for most people, logic reduces to a semantical intuition: the boolean interpretation.

A formula is either

- \( X \): a variable in a fixed countably infinite set \( X \),
- \( A \land B \): a conjunction,
- \( A \lor B \): a disjunction,
- \( \top \): truth,
- \( \bot \): falsity,
- \( A \implies B \): an implication,
In fact, for most people, logic reduces to a semantical intuition: the boolean interpretation.

A formula is either

- \( X \): a variable in a fixed countably infinite set \( \mathcal{X} \),
- \( A \land B \): a conjunction,
- \( A \lor B \): a disjunction,
- \( \top \): truth,
- \( \bot \): falsity,
- \( A \Rightarrow B \): an implication,
- \( \neg A \): a negation (can be coded by \( A \Rightarrow \bot \)).
A *valuation* is a function $\rho : \mathcal{X} \rightarrow \{0, 1\}$. 

Given a valuation $\rho$, we can interpret any formula as a boolean by using the standard interpretation of connectives:

- $A \land B$
- $A \lor B$
- $A \Rightarrow B$
- $\neg A$

For instance, consider $\rho$ such that $\rho(X) = 1$ and $\rho(Y) = 0$. A formula is *valid* when its interpretation is true for every value given to the variables.
A **valuation** is a function $\rho : \mathcal{X} \rightarrow \{0, 1\}$.

Given a valuation $\rho$, we can interpret any formula as a boolean by using the standard interpretation of connectives:

<table>
<thead>
<tr>
<th>$A \land B$</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$A \lor B$</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$A \Rightarrow B$</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$\neg A$</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
The boolean semantics of logic

A **valuation** is a function \( \rho : \mathcal{X} \rightarrow \{0, 1\} \).

Given a valuation \( \rho \), we can interpret any formula as a boolean by using the standard interpretation of connectives:

\[
\begin{array}{c|c|c}
A \land B & 0 & 1 \\
\hline
0 & 0 & 0 \\
1 & 0 & 1 \\
\end{array}
\quad
\begin{array}{c|c|c}
A \lor B & 0 & 1 \\
\hline
0 & 0 & 1 \\
1 & 1 & 1 \\
\end{array}
\quad
\begin{array}{c|c|c}
A \rightarrow B & 0 & 1 \\
\hline
0 & 1 & 1 \\
1 & 0 & 1 \\
\end{array}
\quad
\begin{array}{c}
\neg A \\
\hline
0 \\
1 \\
\end{array}
\]

For instance, consider \( \rho \) such that \( \rho(X) = 1 \) and \( \rho(Y) = 0 \).
A valuation is a function $\rho : \mathcal{X} \to \{0, 1\}$.

Given a valuation $\rho$, we can interpret any formula as a boolean by using the standard interpretation of connectives:

<table>
<thead>
<tr>
<th>$A \land B$</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$A \lor B$</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$A \implies B$</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$\neg A$</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

For instance, consider $\rho$ such that $\rho(X) = 1$ and $\rho(Y) = 0$.

$$(X \implies Y) \implies (\neg X \lor Y)$$
The boolean semantics of logic

A **valuation** is a function $\rho : \mathcal{X} \rightarrow \{0, 1\}$.

Given a valuation $\rho$, we can interpret any formula as a boolean by using the standard interpretation of connectives:

<table>
<thead>
<tr>
<th>$A \land B$</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$A \lor B$</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$A \Rightarrow B$</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$\neg A$</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

For instance, consider $\rho$ such that $\rho(X) = 1$ and $\rho(Y) = 0$.

\[
(X \Rightarrow Y) \Rightarrow (\neg X \lor Y)
\]
A **valuation** is a function \( \rho : \mathcal{X} \to \{0, 1\} \).

Given a valuation \( \rho \), we can interpret any formula as a boolean by using the standard interpretation of connectives:

\[
\begin{array}{c|cc}
A \land B & 0 & 1 \\
\hline
0 & 0 & 0 \\
1 & 0 & 1 \\
\end{array}
\quad
\begin{array}{c|cc}
A \lor B & 0 & 1 \\
\hline
0 & 0 & 1 \\
1 & 1 & 1 \\
\end{array}
\quad
\begin{array}{c|cc}
A \Rightarrow B & 0 & 1 \\
\hline
0 & 1 & 1 \\
1 & 0 & 1 \\
\end{array}
\quad
\begin{array}{c|c}
\neg A & \\
\hline
0 & 1 \\
1 & 0 \\
\end{array}
\]

For instance, consider \( \rho \) such that \( \rho(X) = 1 \) and \( \rho(Y) = 0 \).

\[
(X \Rightarrow Y) \Rightarrow (\neg X \lor Y)
\]
A **valuation** is a function $\rho : \mathcal{X} \rightarrow \{0, 1\}$.

Given a valuation $\rho$, we can interpret any formula as a boolean by using the standard interpretation of connectives:

<table>
<thead>
<tr>
<th>$A \land B$</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$A \lor B$</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$A \Rightarrow B$</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$\neg A$</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
</table>

For instance, consider $\rho$ such that $\rho(X) = 1$ and $\rho(Y) = 0$.

$$(X \Rightarrow Y) \Rightarrow (\neg X \lor Y)$$
The boolean semantics of logic

A **valuation** is a function $\rho : \mathcal{X} \rightarrow \{0, 1\}$.

Given a valuation $\rho$, we can interpret any formula as a boolean by using the standard interpretation of connectives:

<table>
<thead>
<tr>
<th>$A \land B$</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$A \lor B$</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$A \Rightarrow B$</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$\neg A$</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

For instance, consider $\rho$ such that $\rho(X) = 1$ and $\rho(Y) = 0$.

$$(X \Rightarrow Y) \Rightarrow (\neg X \lor Y)$$
A **valuation** is a function $\rho : \mathcal{X} \rightarrow \{0, 1\}$.

Given a valuation $\rho$, we can interpret any formula as a boolean by using the standard interpretation of connectives:

\[
\begin{array}{c|c|c}
A \land B & 0 & 1 \\
\hline
0 & 0 & 0 \\
1 & 0 & 1 \\
\end{array}
\quad
\begin{array}{c|c|c}
A \lor B & 0 & 1 \\
\hline
0 & 0 & 1 \\
1 & 1 & 1 \\
\end{array}
\quad
\begin{array}{c|c|c}
A \implies B & 0 & 1 \\
\hline
0 & 1 & 1 \\
1 & 0 & 1 \\
\end{array}
\quad
\begin{array}{c}
\neg A \\
\hline
0 \\
1 \\
\end{array}
\]

For instance, consider $\rho$ such that $\rho(X) = 1$ and $\rho(Y) = 0$.

\[
(X \implies Y) \implies (\neg X \lor Y)
\]

\[
\underbrace{1}_{\text{1}}
\]

A formula is **valid** when its interpretation is true for every value given to the variables.
The boolean semantics of logic

A **valuation** is a function \( \rho : \mathcal{X} \rightarrow \{0, 1\} \).

Given a valuation \( \rho \), we can interpret any formula as a boolean by using the standard interpretation of connectives:

<table>
<thead>
<tr>
<th>( A \land B )</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( A \lor B )</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( A \Rightarrow B )</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( \neg A )</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

For instance, consider \( \rho \) such that \( \rho(X) = 1 \) and \( \rho(Y) = 0 \).

\[
(X \Rightarrow Y) \Rightarrow (\neg X \lor Y)
\]

A formula is **valid** when its interpretation is true for every value given to the variables.
Part II

Natural deduction
Formalizing logic

People started to formalize the rules of logic:

- we can show meta-theoretic properties of logical systems,
- we can reason on proofs,
- we can build bridges to other things.

Some important advances are

1880: Frege's *Begriffsschrift*
1900: Hilbert's quest for foundations of mathematics
1920: Brouwer's intuitionism
1930: Gentzen's natural deduction
People started to formalize the rules of logic:

- we can show meta-theoretic properties of logical systems,
- we can reason on proofs,
- we can build bridges to other things.

Some important advances are

- 1880: Frege’s *Begriffsschrift*
People started to formalize the rules of logic:

- we can show meta-theoretic properties of logical systems,
- we can reason on proofs,
- we can build bridges to other things.

Some important advances are

- 1880: Frege’s *Begriffsschrift*
- 1990: Hilbert’s quest for foundations of mathematics (following paradoxes such as Russell’s)
People started to formalize the rules of logic:

- we can show meta-theoretic properties of logical systems,
- we can reason on proofs,
- we can build bridges to other things.

Some important advances are

- 1880: Frege’s *Begriffsschrift*
- 1990: Hilbert’s quest for foundations of mathematics
- 1920: Brouwer’s intuitionism
Formalizing logic

People started to formalize the rules of logic:

- we can show meta-theoretic properties of logical systems,
- we can reason on proofs,
- we can build bridges to other things.

Some important advances are

- 1880: Frege’s *Begriffsschrift*
- 1990: Hilbert’s quest for foundations of mathematics
- 1920: Brouwer’s intuitionism
- 1930: Gentzen’s natural deduction
Let’s shift from provability to proofs.
A context $\Gamma$ is a list of formulas $A_1, \ldots, A_n$.
A context $\Gamma$ is a list of formulas $A_1, \ldots, A_n$.

A sequent $\Gamma \vdash A$ consists of a context $\Gamma$ together with a formula $A$. 
A context $\Gamma$ is a list of formulas $A_1, \ldots, A_n$.

A sequent

$\Gamma \vdash A$

consists of a context $\Gamma$ together with a formula $A$.

An inference rule

$\Gamma_1 \vdash A_1 \quad \ldots \quad \Gamma_n \vdash A_n$

specifies when I can deduce a sequent from others / what I need to show in order to prove a sequent.
Intuitionistic natural deduction (NJ)

\[ \Gamma \vdash A, \Gamma' \vdash A \ (ax) \]

\[ \begin{array}{c}
\Gamma \vdash A \Rightarrow B \\
\Gamma \vdash A \\
\hline
\Gamma \vdash B \\
\end{array} \ (\Rightarrow E) \]

\[ \begin{array}{c}
\Gamma \vdash A \Rightarrow B \\
\Gamma \vdash A \\
\hline
\Gamma \vdash A \Rightarrow B \\
\end{array} \ (\Rightarrow I) \]

\[ \begin{array}{c}
\Gamma \vdash A \land B \\
\hline
\Gamma \vdash A \\
\end{array} \ (\land E_1) \]

\[ \begin{array}{c}
\Gamma \vdash A \land B \\
\hline
\Gamma \vdash B \\
\end{array} \ (\land E_2) \]

\[ \begin{array}{c}
\Gamma \vdash A \land B \\
\Gamma \vdash A \\
\Gamma \vdash B \\
\hline
\Gamma \vdash A \land B \\
\end{array} \ (\land I) \]

\[ \begin{array}{c}
\Gamma \vdash A \lor B \\
\Gamma \vdash A \\
\Gamma \vdash B \\
\hline
\Gamma \vdash A \lor B \\
\end{array} \ (\lor I) \]

\[ \begin{array}{c}
\Gamma \vdash A \lor B \\
\hline
\Gamma \vdash C \\
\end{array} \ (\lor E) \]

\[ \begin{array}{c}
\Gamma \vdash A \lor B \\
\Gamma \vdash A \\
\Gamma \vdash B \\
\hline
\Gamma \vdash A \lor B \\
\end{array} \ (\lor I) \]

\[ \begin{array}{c}
\Gamma \vdash \bot \\
\hline
\Gamma \vdash A \\
\end{array} \ (\bot E) \]

\[ \begin{array}{c}
\Gamma \vdash \top \\
\hline
\Gamma \vdash \top \\
\end{array} \ (\top I) \]
Some remarks about the rules

Apart from the axiom, rules are either

- **elimination** rules: use a connective,
- **introduction** rules: show a connective.
Some remarks about the rules

Apart from the axiom, rules are either

- **elimination** rules: use a connective,
- **introduction** rules: show a connective.

The leaves are axiom (or $\top$ introduction) rules: all the other rules have premises.
Some remarks about the rules

Apart from the axiom, rules are either

- **elimination** rules: use a connective,
- **introduction** rules: show a connective.

The leaves are axiom (or $\top$ introduction) rules: all the other rules have premises.

The axiom is the only way to “use” a formula in the context.
Some remarks about the rules

Apart from the axiom, rules are either

- **elimination** rules: use a connective,
- **introduction** rules: show a connective.

The leaves are axiom (or $\top$ introduction) rules: all the other rules have premises.

The axiom is the only way to “use” a formula in the context.

There is no introduction of $\bot$ and elimination of $\top$. 
Some remarks about the rules

Apart from the axiom, rules are either

- **elimination** rules: use a connective,
- **introduction** rules: show a connective.

The leaves are axiom (or $\top$ introduction) rules: all the other rules have premises.

The axiom is the only way to “use” a formula in the context.

There is no introduction of $\bot$ and elimination of $\top$.

The **principal** premise is the leftmost premise of an elimination rule.
A **proof** is a tree formed with the derivation rules of NJ.
A *proof* is a tree formed with the derivation rules of NJ.

A sequent $\Gamma \vdash A$ is *provable* when it is the conclusion of some proof.
Proofs

A **proof** is a tree formed with the derivation rules of NJ.

A sequent $\Gamma \vdash A$ is **provable** when it is the conclusion of some proof.

A formula $A$ is **provable** when the sequent $\vdash A$ is.
A proof of $A \Rightarrow A$

$\vdash A \Rightarrow A$
A proof of $A \Rightarrow A$

\[
\frac{A \vdash A}{\vdash A \Rightarrow A} (\Rightarrow_1)
\]
A proof of \( A \Rightarrow A \)

\[
\begin{align*}
A & \vdash A \quad \text{(ax)} \\
\hline
A & \vdash A \\
\vdash A \Rightarrow A \quad \text{(\(\Rightarrow\)1)}
\end{align*}
\]
A proof of $A \land B \Rightarrow A \lor B$
A proof of $A \land B \Rightarrow A \lor B$

\[
\begin{align*}
A \land B & \vdash A \lor B \\
\vdash A \land B \Rightarrow A \lor B & \quad (\Rightarrow_1)
\end{align*}
\]
A proof of $A \land B \Rightarrow A \lor B$

\[
\begin{align*}
A \land B & \vdash A \\
\hline
A \land B & \vdash A \lor B \\
\vdash A \land B \Rightarrow A \lor B
\end{align*}
\]

(\lor_i)

(\Rightarrow_i)
A proof of $A \land B \Rightarrow A \lor B$
A proof of $A \land B \Rightarrow A \lor B$

\[
\begin{align*}
A \land B & \vdash A \land B \quad \text{(ax)} \\
A \land B & \vdash A \quad \text{($\land E$)} \\
A \land B & \vdash A \lor B \quad \text{($\lor I$)} \\
\vdash A \land B \Rightarrow A \lor B \quad \text{($\Rightarrow I$)}
\end{align*}
\]
Proposition (Correctness)
The boolean interpretation is correct with respect to the rules:

if $A$ is provable then $A$ is valid.
Proposition (Correctness)
The boolean interpretation is correct with respect to the rules:
if $A$ is provable then $A$ is valid.

Proof.
We say that a sequent $\Gamma \vdash A$ is valid if $A$ is true for every valuation $\rho$ which makes the formulas in $\Gamma$ true.
Correctness of the boolean interpretation

**Proposition (Correctness)**
The boolean interpretation is correct with respect to the rules:
if \( A \) is provable then \( A \) is valid.

**Proof.**
We say that a sequent \( \Gamma \vdash A \) is *valid* if \( A \) is true for every valuation \( \rho \) which makes the formulas in \( \Gamma \) true.

Show that for every deduction rule if the premises are valid then the conclusion is valid.
Proposition (Correctness)
The boolean interpretation is correct with respect to the rules:
if $A$ is provable then $A$ is valid.

Proof.
We say that a sequent $\Gamma \vdash A$ is valid if $A$ is true for every valuation $\rho$ which makes the formulas in $\Gamma$ true.

Show that for every deduction rule if the premises are valid then the conclusion is valid.

By contraposition: if $A$ is not valid then it is not provable.
An major property of a logical system is that it is **consistency**: there is at least one formula which is not provable.
Correctness of the boolean interpretation

An major property of a logical system is that it is **consistency**: there is at least one formula which is not provable.

**Lemma**

A logical system is consistent if and only if $\bot$ is not provable.

**Proof.**

Recall that the elimination rule for negation is

$$\frac{\Gamma \vdash \bot \quad \bot \vdash A}{\Gamma \vdash A} (\bot_E).$$

$\square$
An major property of a logical system is that it is **consistency**: there is at least one formula which is not provable.

**Lemma**

A logical system is consistent if and only if $\bot$ is not provable.

**Proof.**

Recall that the elimination rule for negation is

$$\Gamma \vdash \bot \quad (\bot_E)$$

**Proposition**

The system NJ is consistent.

**Proof.**

If $\bot$ was provable then, by correctness, it would be valid wrt the boolean interpretation, which it is not, by definition.
Completeness of the boolean interpretation

The **completeness** of the boolean interpretation would be the converse: every valid formula is provable.
Completeness of the boolean interpretation

The *completeness* of the boolean interpretation would be the converse: every valid formula is provable.

This is not true: essentially, we miss the fact that $\neg\neg A \Rightarrow A$, more on this later.
Part III

The Curry-Howard correspondence
Types in functional programming languages

In modern languages, everything has a type:

<table>
<thead>
<tr>
<th>expression</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

In modern languages, everything has a type:

<table>
<thead>
<tr>
<th>expression</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>int</td>
</tr>
</tbody>
</table>
In modern languages, everything has a type:

<table>
<thead>
<tr>
<th>expression</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>int</td>
</tr>
<tr>
<td>true</td>
<td>int</td>
</tr>
</tbody>
</table>
In modern languages, everything has a type:

<table>
<thead>
<tr>
<th>expression</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>int</td>
</tr>
<tr>
<td>true</td>
<td>bool</td>
</tr>
</tbody>
</table>
In modern languages, everything has a type:

<table>
<thead>
<tr>
<th>expression</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>int</td>
</tr>
<tr>
<td>true</td>
<td>bool</td>
</tr>
<tr>
<td><code>fun x -&gt; 2 * x</code></td>
<td></td>
</tr>
</tbody>
</table>
Types in functional programming languages

In modern languages, everything has a type:

<table>
<thead>
<tr>
<th>expression</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>int</td>
</tr>
<tr>
<td>true</td>
<td>bool</td>
</tr>
<tr>
<td>fun x -&gt; 2 * x</td>
<td>int -&gt; int</td>
</tr>
</tbody>
</table>
Types in functional programming languages

In modern languages, everything has a type:

<table>
<thead>
<tr>
<th>expression</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>int</td>
</tr>
<tr>
<td>true</td>
<td>bool</td>
</tr>
<tr>
<td>fun x -&gt; 2 * x</td>
<td>int -&gt; int</td>
</tr>
<tr>
<td>fun x -&gt; (2 * x, &quot;A&quot;)</td>
<td></td>
</tr>
</tbody>
</table>
## Types in functional programming languages

In modern languages, everything has a type:

<table>
<thead>
<tr>
<th>expression</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>int</td>
</tr>
<tr>
<td>true</td>
<td>bool</td>
</tr>
<tr>
<td>fun x -&gt; 2 * x</td>
<td>int -&gt; int</td>
</tr>
<tr>
<td>fun x -&gt; (2 * x, &quot;A&quot;)</td>
<td>int -&gt; int * string</td>
</tr>
</tbody>
</table>
In modern languages, everything has a type:

<table>
<thead>
<tr>
<th>expression</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>int</td>
</tr>
<tr>
<td>true</td>
<td>bool</td>
</tr>
<tr>
<td>fun x -&gt; 2 * x</td>
<td>int -&gt; int</td>
</tr>
<tr>
<td>fun x -&gt; (2 * x, &quot;A&quot;)</td>
<td>int -&gt; int * string</td>
</tr>
<tr>
<td>fun x -&gt; (x, &quot;A&quot;)</td>
<td></td>
</tr>
</tbody>
</table>

Types in functional programming languages
Types in functional programming languages

In modern languages, everything has a type:

<table>
<thead>
<tr>
<th>expression</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>int</td>
</tr>
<tr>
<td>true</td>
<td>bool</td>
</tr>
<tr>
<td>fun x -&gt; 2 * x</td>
<td>int -&gt; int</td>
</tr>
<tr>
<td>fun x -&gt; (2 * x, &quot;A&quot;)</td>
<td>int -&gt; int * string</td>
</tr>
<tr>
<td>fun x -&gt; (x, &quot;A&quot;)</td>
<td>’a -&gt; ’a * string</td>
</tr>
</tbody>
</table>
Types in functional programming languages

In modern languages, everything has a type:

<table>
<thead>
<tr>
<th>expression</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>int</td>
</tr>
<tr>
<td>true</td>
<td>bool</td>
</tr>
<tr>
<td>fun x -&gt; 2 * x</td>
<td>int -&gt; int</td>
</tr>
<tr>
<td>fun x -&gt; (2 * x, &quot;A&quot;)</td>
<td>int -&gt; int * string</td>
</tr>
<tr>
<td>fun x -&gt; (x, &quot;A&quot;)</td>
<td>'a -&gt; 'a * string</td>
</tr>
<tr>
<td>fun x y -&gt; x</td>
<td></td>
</tr>
</tbody>
</table>
In modern languages, everything has a type:

<table>
<thead>
<tr>
<th>expression</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>int</td>
</tr>
<tr>
<td>true</td>
<td>bool</td>
</tr>
<tr>
<td>fun x -&gt; 2 * x</td>
<td>int -&gt; int</td>
</tr>
<tr>
<td>fun x -&gt; (2 * x, &quot;A&quot;)</td>
<td>int -&gt; int * string</td>
</tr>
<tr>
<td>fun x -&gt; (x, &quot;A&quot;)</td>
<td>'a -&gt; 'a * string</td>
</tr>
<tr>
<td>fun x y -&gt; x</td>
<td>'a -&gt; 'b -&gt; 'a</td>
</tr>
</tbody>
</table>
## Types in functional programming languages

In modern languages, everything has a type:

<table>
<thead>
<tr>
<th>expression</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>int</td>
</tr>
<tr>
<td>true</td>
<td>bool</td>
</tr>
<tr>
<td>fun x -&gt; 2 * x</td>
<td>int -&gt; int</td>
</tr>
<tr>
<td>fun x -&gt; (2 * x, &quot;A&quot;)</td>
<td>int -&gt; int * string</td>
</tr>
<tr>
<td>fun x -&gt; (x, &quot;A&quot;)</td>
<td>'a -&gt; 'a * string</td>
</tr>
<tr>
<td>fun x y -&gt; x</td>
<td>'a -&gt; 'b -&gt; 'a</td>
</tr>
<tr>
<td>fun x -&gt; x x</td>
<td>'a -&gt; 'b -&gt; 'a</td>
</tr>
</tbody>
</table>
In modern languages, everything has a type:

<table>
<thead>
<tr>
<th>expression</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>int</td>
</tr>
<tr>
<td>true</td>
<td>bool</td>
</tr>
<tr>
<td>fun x -&gt; 2 * x</td>
<td>int -&gt; int</td>
</tr>
<tr>
<td>fun x -&gt; (2 * x, &quot;A&quot;)</td>
<td>int -&gt; int * string</td>
</tr>
<tr>
<td>fun x -&gt; (x, &quot;A&quot;)</td>
<td>'a -&gt; 'a * string</td>
</tr>
<tr>
<td>fun x y -&gt; x</td>
<td>'a -&gt; 'b -&gt; 'a</td>
</tr>
<tr>
<td>fun x -&gt; x x</td>
<td></td>
</tr>
</tbody>
</table>

Error: This expression has type 'a -> 'b but an expression was expected of type 'a The type variable 'a occurs inside 'a -> 'b
Types in functional programming languages

In modern languages, everything has a type:

<table>
<thead>
<tr>
<th>expression</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>int</td>
</tr>
<tr>
<td>true</td>
<td>bool</td>
</tr>
<tr>
<td>fun x -&gt; 2 * x</td>
<td>int -&gt; int</td>
</tr>
<tr>
<td>fun x -&gt; (2 * x, &quot;A&quot;)</td>
<td>int -&gt; int * string</td>
</tr>
<tr>
<td>fun x -&gt; (x, &quot;A&quot;)</td>
<td>'a -&gt; 'a * string</td>
</tr>
<tr>
<td>fun x y -&gt; x</td>
<td>'a -&gt; 'b -&gt; 'a</td>
</tr>
<tr>
<td>fun x -&gt; x x</td>
<td>'a -&gt; 'b</td>
</tr>
</tbody>
</table>

Namely, the type of x should be of the form 'a -> 'b with 'a = ('a -> 'b), i.e.

```
((... -> 'b) -> 'b) -> 'b
```
Types in functional programming languages

In good languages, typing is

- **static**: checked at compilation time

  - e.g. a program of type `int` will produce an integer, typically
    ```ml
    let n : int = 6 + 2
    ```
  - but the following is rejected
    ```ml
    let n : int = 3 + "a"
    ```
  - but the following is accepted
    ```ml
    let rec loop x = loop x
    let n : int = loop "a"
    ```
Types in functional programming languages

In good languages, typing is

- **static**: checked at compilation time
- **safe**: if a program of a given type produces a values then it is of the expected form
In good languages, typing is

- **static**: checked at compilation time
- **safe**: if a program of a given type produces a values then it is of the expected form
  e.g. a program of type `int` will produce an integer, typically

```ocaml
let n : int = 6 + 2
```
In good languages, typing is

- **static**: checked at compilation time
- **safe**: if a program of a given type produces a values then it is of the expected form
  
  e.g. a program of type `int` will produce an integer, typically
  
  ```
  let n : int = 6 + 2
  ```

  but the following is rejected
  
  ```
  let n : int = 3 + "a"
  ```
Types in functional programming languages

In good languages, typing is

- **static**: checked at compilation time
- **safe**: if a program of a given type produces a values then it is of the expected form
  
  e.g. a program of type `int` will produce an integer, typically
  
  ```
  let n : int = 6 + 2
  ```
  
  but the following is rejected
  
  ```
  let n : int = 3 + "a"
  ```
  
  but the following is accepted
  
  ```
  let rec loop x = loop x
  ```

  ```
  let n : int = loop "a"
  ```
Simply typed λ-calculus

For simplicity, let us consider a language where **types** are of the form

- constants (e.g. `int`, `bool`, ...),
- \( A \rightarrow B \): a function taking an \( A \) and producing a \( B \),
- \( A \times B \): a pair of an \( A \) and a \( B \),
- \( 1 \): unit.
Simply typed $\lambda$-calculus

For simplicity, let us consider a language where types are of the form

- constants (e.g. `int`, `bool`, ...),
- $A \rightarrow B$: a function taking an $A$ and producing a $B$,
- $A \times B$: a pair of an $A$ and a $B$,
- $1$: unit.

A terms $t$ (= a program) is of the form

- a constant (natural numbers, booleans, etc.)
- a variable $x$,
- $\lambda x^A.t$: the function which to $x$ associates $t$ (in OCaml: `fun (x : A) -> t`),
- $t\ u$: we apply the function $t$ to $u$,
- $\langle t, u \rangle$: a pair,
- $\pi_l(t), \pi_r(t)$: the projections,
- $\langle \rangle$: unit.
A context $\Gamma$ is a list

$$x_1 : A_1, \ldots, x_n : A_n$$

of pairs consisting of a variable $x_i$ and a type $A_i$ (all the variables we know of).
Typing rules

A context $\Gamma$ is a list

$$x_1 : A_1, \ldots, x_n : A_n$$

of pairs consisting of a variable $x_i$ and a type $A_i$ (all the variables we know of).

A sequent is a triple

$$\Gamma \vdash t : A$$

consisting of a context $\Gamma$ a term $t$ and a type $A$ (a typing judgment).
A context $\Gamma$ is a list

$$x_1 : A_1, \ldots, x_n : A_n$$

of pairs consisting of a variable $x_i$ and a type $A_i$ (all the variables we know of).

A sequent is a triple

$$\Gamma \vdash t : A$$

consisting of a context $\Gamma$ a term $t$ and a type $A$ (a typing judgment).

A term $t$ has type $A$ when $\vdash t : A$ can be derived using the typing rules.
Typing rules

\[ \Gamma, x : A, \Gamma' \vdash x : A \]

\[ \frac{\Gamma \vdash t : A \to B \quad \Gamma \vdash u : A}{\Gamma \vdash tu : B} \quad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x^A.t : A \to B} \]

\[ \frac{\Gamma \vdash t : A \times B \quad \Gamma \vdash t : A \times B}{\Gamma \vdash \pi_l(t) : A} \quad \frac{\Gamma \vdash t : A \times B \quad \Gamma \vdash t : A \times B}{\Gamma \vdash \pi_r(t) : B} \]

\[ \frac{\Gamma \vdash t : A \quad \Gamma \vdash u : B}{\Gamma \vdash \langle t, u \rangle : A \times B} \]

\[ \frac{}{\Gamma \vdash \langle \rangle : 1} \]
A typing example

⊢ λf^{A→A}.λx^A.f(f\ x) : (A → A) → A → A
A typing example

\[ f : A \rightarrow A \vdash \lambda x^A. f(f x) : A \rightarrow A \]

\[ \vdash \lambda f^A \rightarrow A. \lambda x^A. f(f x) : (A \rightarrow A) \rightarrow A \rightarrow A \] (\(\rightarrow_1\))
\[
\frac{f : A \to A, x : A}{f(f x) : A} \quad (\to_1)
\]

\[
\frac{f : A \to A}{\lambda x^A.f(f x) : A \to A} \quad (\to_1)
\]

\[
\vdash \lambda f^{A \to A}.\lambda x^A.f(f x) : (A \to A) \to A \to A
\]
A typing example

\[
\begin{align*}
\Gamma &\vdash f : A \to A \\
\Gamma &\vdash f \ x : A \\
\hline
f &: A \to A, x : A \vdash f(f \ x) : A \\
\hline
\Gamma &\vdash \lambda x^A. f(f \ x) : A \to A \\
\hline
\vdash \lambda f^A \to A. \lambda x^A. f(f \ x) : (A \to A) \to A \to A \\
\end{align*}
\]
A typing example

\[ \Gamma \vdash f : A \rightarrow A \]  

\[ \Gamma \vdash f \cdot x : A \]  

\[ f : A \rightarrow A, x : A \vdash f(f \cdot x) : A \]  

\[ \Gamma \vdash \lambda x^A.f(f \cdot x) : A \rightarrow A \]  

\[ \vdash \lambda f^{A \rightarrow A}.\lambda x^A.f(f \cdot x) : (A \rightarrow A) \rightarrow A \rightarrow A \]
A typing example

\[\begin{align*}
\Gamma \vdash f : A &\rightarrow A \\
\Gamma &\vdash f \, x : A \\
\Gamma &\vdash x : A \\
\Gamma &\vdash f \, (f \, x) : A \\
\Gamma &\vdash \lambda x. f \, x : (A \rightarrow A) \rightarrow A \\
\Gamma &\vdash \lambda (f : A \rightarrow A). \lambda x. f \, x : (A \rightarrow A) \rightarrow A \rightarrow A
\end{align*}\]
A typing example

\[
\begin{align*}
\Gamma \vdash f : A \rightarrow A & \quad \text{(ax)} \quad \Gamma \vdash x : A & \quad \text{(→E)} \\
\Gamma \vdash f : A \rightarrow A & \quad (\text{ax}) \\
\Gamma \vdash f \times : A & \quad (\text{→E}) \\
\therefore f : A \rightarrow A, x : A \vdash f(\times) : A & \quad (\text{→E}) \\
\therefore f : A \rightarrow A \vdash \lambda x^A. f(\times) : A \rightarrow A & \quad (\text{→I}) \\
\vdash \lambda f^A \rightarrow A. \lambda x^A. f(\times) : (A \rightarrow A) \rightarrow A \rightarrow A & \quad (\text{→I})
\end{align*}
\]
A typing example

$\Gamma \vdash f : A \rightarrow A$ (ax)
$\Gamma \vdash x : A$ (ax)

$\Gamma \vdash f \ x : A$ ($\rightarrow$E)

$\Gamma \vdash f : A \rightarrow A, x : A \vdash f(f \ x) : A$ ($\rightarrow$E)

$\Gamma \vdash f : A \rightarrow A, x : A \vdash \lambda x^A.f(f \ x) : A \rightarrow A$ ($\rightarrow$I)

$\vdash \lambda f^A \rightarrow A. \lambda x^A.f(f \ x) : (A \rightarrow A) \rightarrow A \rightarrow A$ ($\rightarrow$I)
A typing example

\[
\begin{align*}
 f : A \to A, x : A & \vdash x : A \\
 \underline{f : A \to A \vdash \lambda x^A.x : A \to A} & \quad (\to_1) \\
 \vdash \lambda f^{A\to A} \cdot \lambda x^A.x : (A \to A) \to A \to A & \quad (\to_1)
\end{align*}
\]
Note that depending on a term at most one rule applies:

**Proposition (Uniqueness of typing)**

*Given a term* \( t \) *such that* \( \Gamma \vdash t : A \) *and* \( \Gamma \vdash t : A' \) *are derivable then* \( A = A' \)

**Proof.**

By induction on the derivations. \( \square \)
Proposition (Uniqueness of typing)

Given a term $t$ such that $\Gamma \vdash t : A$ and $\Gamma \vdash t : A'$ are derivable then $A = A'$

and given two derivations

\[
\begin{array}{c}
\pi \\
\hline
\Gamma \vdash t : A
\end{array}
\quad
\begin{array}{c}
\pi' \\
\hline
\Gamma \vdash t : A
\end{array}
\]

we have $\pi = \pi'$.

Proof.

By induction on the derivations. \qed
The Curry-Howard correspondence

A very simple observation is that if we “erase” terms in typing rules and slightly change the notations of connectives

<table>
<thead>
<tr>
<th>typing</th>
<th>logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\to$</td>
<td>$\Rightarrow$</td>
</tr>
<tr>
<td>$\times$</td>
<td>$\land$</td>
</tr>
<tr>
<td>$1$</td>
<td>$\top$</td>
</tr>
</tbody>
</table>

we obtain the rules of logic, for instance:

$$
\frac{\Gamma \vdash t : A \quad \Gamma \vdash u : B}{\Gamma \vdash \langle t, u \rangle : A \times B} (\times I) \quad \sim \quad \frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \times B} (\land I)
$$
The Curry-Howard correspondence

\[ \Gamma, x : A, \Gamma' \vdash x : A \]

\[ \Gamma \vdash t : A \rightarrow B \quad \Gamma \vdash u : A \]
\[ \Gamma \vdash t u : B \quad (\rightarrow_E) \]

\[ \Gamma \vdash \lambda x^A.t : A \rightarrow B \]
\[ (\rightarrow_I) \]

\[ \Gamma \vdash t : A \times B \]
\[ \Gamma \vdash \pi_l(t) : A \quad (\times_\Gamma) \]
\[ \Gamma \vdash \pi_r(t) : B \quad (\times_\Gamma) \]

\[ \Gamma \vdash \langle t, u \rangle : A \times B \quad (\times_1) \]

\[ \Gamma \vdash \langle \rangle : 1 \quad (1_\Gamma) \]
The Curry-Howard correspondence

\[ \Gamma, \quad A, \Gamma' \vdash A \quad (\text{ax}) \]

\[
\begin{align*}
\Gamma \vdash A \Rightarrow B & \quad \Gamma \vdash A & \quad \Gamma, \quad A \vdash B & \quad (\Rightarrow E) \\
\Gamma \vdash B & & \Gamma \vdash A \Rightarrow B & \quad (\Rightarrow I)
\end{align*}
\]

\[
\begin{align*}
\Gamma \vdash A \land B & \quad \Gamma \vdash A \land B & \quad \Gamma \vdash A \land B & \quad (\land E) \\
\Gamma \vdash A & & \Gamma \vdash A \land B & \quad (\land I) \\
\Gamma \vdash B & & \Gamma \vdash B & \quad (\land I)
\end{align*}
\]

\[
\Gamma \vdash () & \quad (\top I)
\]
The Curry-Howard correspondence

Theorem
There is a bijection between given a context $\Gamma$ and a type $A$

i. terms $t$ such that $\Gamma \vdash t : A$ is derivable,

ii. typing derivations $\Gamma \vdash t : A$ for some $t$,

iii. proofs of $\Gamma \vdash A$. 

This extends to richer fragments (disjunctions, quantifications, etc.).
The Curry-Howard correspondence

Theorem
There is a bijection between given a context $\Gamma$ and a type $A$

- terms $t$ such that $\Gamma \vdash t : A$ is derivable,
- typing derivations $\Gamma \vdash t : A$ for some $t$,
- proofs of $\Gamma \vdash A$.

This extends to richer fragments (disjunctions, quantifications, etc.).
The Curry-Howard correspondence

\[
\lambda f^{A \to A}, \lambda x^A. f(f x)
\]
The Curry-Howard correspondence

\[
\begin{align*}
\Gamma & \vdash f : A \to A \\
\Gamma & \vdash x : A \\
\Gamma & \vdash f : A \to A \\
\Gamma & \vdash f x : A \\
\Gamma & \vdash x : A \\
\Gamma & \vdash f x : A \\
\Gamma & \vdash \lambda x : A. f (f x) : A \to A \\
\Gamma & \vdash \lambda f : A \to A. \lambda x : A. f (f x) : (A \to A) \to A \to A
\end{align*}
\]
The Curry-Howard correspondence
Part IV

Semantics of propositional logic
This suggests that the interpretation of formulas as booleans is very poor.

We would rather like to interpret formulas as sets, typically

\[
\llbracket \text{int} \rrbracket = \mathbb{N}
\]
The set-theoretic interpretation

This suggests that the interpretation of formulas as booleans is very poor.

We would rather like to interpret formulas as sets, typically

\[ \text{[int]} = \mathbb{N} \]

Suppose fixed an interpretation of base types as above, we extend the interpretation as

- \([A \Rightarrow B] = [A] \rightarrow [B]\) is the set of functions from \([A]\) to \([B]\),
- \([A \land B] = [A] \times [B]\) is the cartesian product of \([A]\) and \([B]\),
- \([\top] = \{\star\}\) is the set with one element.
We extend the interpretation to contexts

\[ \Gamma = A_1, \ldots, A_n \]

by

\[ \llbracket \Gamma \rrbracket = \llbracket A_1 \rrbracket \times \ldots \times \llbracket A_n \rrbracket \]
The set-theoretic interpretation

We extend the interpretation to contexts

$$\Gamma = A_1, \ldots, A_n$$

by

$$[\Gamma] = [A_1] \times \ldots \times [A_n]$$

Finally, we extend the interpretation to proofs by

$$[\Gamma \vdash t : A] \in [\Gamma] \rightarrow [A]$$

by induction on their derivation.
For instance, suppose that our proof ends with

\[ \begin{align*}
\Gamma & \vdash t : A \\
\Gamma & \vdash u : B \\
\Gamma & \vdash \langle t, u \rangle : A \times B
\end{align*} \] 

(×₁)
The set-theoretic interpretation

For instance, suppose that our proof ends with

\[
\frac{\Gamma \vdash t : A \quad \Gamma \vdash u : B}{\Gamma \vdash \langle t, u \rangle : A \times B} (\times_1)
\]

By induction hypothesis we have

\[
[\Gamma \vdash t : A] \in [\Gamma] \rightarrow [A] \quad [\Gamma \vdash u : B] \in [\Gamma] \rightarrow [B]
\]
The set-theoretic interpretation

For instance, suppose that our proof ends with

\[
\frac{\Gamma \vdash t : A \quad \Gamma \vdash u : B}{\Gamma \vdash \langle t, u \rangle : A \times B} (\times_1)
\]

By induction hypothesis we have

\[
[\Gamma \vdash t : A] \in [[\Gamma]] \rightarrow [[A]] \quad [\Gamma \vdash u : B] \in [[\Gamma]] \rightarrow [[B]]
\]

and we define

\[
[[\Gamma \vdash \langle t, u \rangle : A \times B]] : [[\Gamma]] \rightarrow [[A]] \times [[B]]
\]

\[
g \mapsto ([[\Gamma \vdash t : A]](g), [[\Gamma \vdash u : B]](g))
\]
The set-theoretic interpretation

For instance, suppose that our proof ends with

\[ \Gamma \vdash t : A \rightarrow B \quad \Gamma \vdash u : A \]

\[ \Gamma \vdash t \; u : B \] \hspace{1cm} (\rightarrow_E)
The set-theoretic interpretation

For instance, suppose that our proof ends with

\[
\begin{align*}
\Gamma \vdash t : A \rightarrow B & \quad \Gamma \vdash u : A \\
\hline
\Gamma \vdash t \ u : B \quad \text{(→}_{E})
\end{align*}
\]

By induction hypothesis we have

\[
\begin{align*}
[[\Gamma \vdash t : A \rightarrow B]] & \in [[\Gamma]] \rightarrow ([[A]] \rightarrow [[B]]) & \quad [[\Gamma \vdash u : A]] & \in [[\Gamma]] \rightarrow [[A]]
\end{align*}
\]

Other cases are "similar".
The set-theoretic interpretation

For instance, suppose that our proof ends with

\[
\frac{\Gamma \vdash t : A \rightarrow B \quad \Gamma \vdash u : A}{\Gamma \vdash t \ u : B}\text{ (→E)}
\]

By induction hypothesis we have

\[
[\Gamma \vdash t : A \rightarrow B] \in [\Gamma] \rightarrow ([A] \rightarrow [B]) \quad [\Gamma \vdash u : A] \in [\Gamma] \rightarrow [A]
\]

and we define

\[
[\Gamma \vdash t \ u : B] : [\Gamma] \rightarrow [B]
\]

\[
g \mapsto [\Gamma \vdash t : A \rightarrow B](g)([\Gamma \vdash u : A](g))
\]
The set-theoretic interpretation

For instance, suppose that our proof ends with

\[
\Gamma \vdash t : A \to B \quad \Gamma \vdash u : A \\
\Gamma \vdash t \ u : B
\]

By induction hypothesis we have

\[
[\Gamma \vdash t : A \to B] \in [\Gamma] \to ([A] \to [B]) \quad \quad [\Gamma \vdash u : A] \in [\Gamma] \to [A]
\]

and we define

\[
[\Gamma \vdash t \ u : B] : [\Gamma] \to [B] \\
\quad g \mapsto [\Gamma \vdash t : A \to B](g)([\Gamma \vdash u : A](g))
\]

Other cases are “similar”.
The set-theoretic interpretation

This can further be extended to other connectives by setting

\[ [A \lor B] = [A] \sqcup [B] \]

and

\[ [\bot] = \emptyset \]

and

\[ [\neg A] = [A \Rightarrow \bot] = [A] \rightarrow \emptyset \]
We recover the previous interpretation by considering whether a set is empty or not:

\[
\begin{array}{c|cc}
A \Rightarrow B & \emptyset & 1 \\
\hline
\emptyset & 1 & 1 \\
1 & \emptyset & 1 \\
\end{array}
\quad
\begin{array}{c|cc}
A \land B & \emptyset & 1 \\
\hline
\emptyset & 0 & 0 \\
1 & 1 & 1 \\
\end{array}
\quad
\begin{array}{c|cc}
A \lor B & \emptyset & 1 \\
\hline
\emptyset & 0 & 1 \\
1 & 1 & 1 \\
\end{array}
\quad
\begin{array}{c|c}
\neg A & 1 \\
\hline
\emptyset & 1 \\
1 & \emptyset \\
\end{array}
\]

We have shifted from provability to proofs!
The interpretation of negation is

\[ [\neg A] = [A \Rightarrow \bot] = [A] \rightarrow \emptyset \]
The interpretation of negation is

\[ [\neg A] = [A \Rightarrow \bot] = [A] \rightarrow \emptyset \]

Thus,

\[ [\neg A] = \begin{cases} \emptyset & \text{if } A \neq \emptyset, \\ \{\star\} & \text{if } A = \emptyset, \end{cases} \]

We thus understand why we cannot expect to have a proof of \( \neg\neg A \Rightarrow A \) in general! And that doubly negated formulas behave as in the boolean interpretation.
Classical logic

The interpretation of negation is

\[ [-A] = [A \Rightarrow \bot] = [A] \rightarrow \emptyset \]

Thus,

\[ [-A] = \begin{cases} 
\emptyset & \text{if } A \neq \emptyset, \\
\{\star\} & \text{if } A = \emptyset, 
\end{cases} \]

and

\[ [-[-A]] = \begin{cases} 
\emptyset & \text{if } [A] = \emptyset, \\
\{\star\} & \text{if } [A] \neq \emptyset. 
\end{cases} \]
The interpretation of negation is

\[ [\neg A] = [A \Rightarrow \bot] = [A] \rightarrow \emptyset \]

Thus,

\[ [\neg A] = \begin{cases} \emptyset & \text{if } A \neq \emptyset, \\ \{\star\} & \text{if } A = \emptyset, \end{cases} \quad \text{and} \quad [\neg\neg A] = \begin{cases} \emptyset & \text{if } [A] = \emptyset, \\ \{\star\} & \text{if } [A] \neq \emptyset. \end{cases} \]

We thus understand why we cannot expect to have a proof of

\[ \neg\neg A \Rightarrow A \]

in general!
The interpretation of negation is

\[ [\neg A] = [A \Rightarrow \bot] = [A] \Rightarrow \emptyset \]

Thus,

\[ [\neg A] = \begin{cases} \emptyset & \text{if } A \neq \emptyset, \\ \{\star\} & \text{if } A = \emptyset, \end{cases} \]

and

\[ [\neg \neg A] = \begin{cases} \emptyset & \text{if } [A] = \emptyset, \\ \{\star\} & \text{if } [A] \neq \emptyset. \end{cases} \]

We thus understand why we cannot expect to have a proof of

\[ \neg \neg A \Rightarrow A \]

in general! And that doubly negated formulas behave as in the boolean interpretation.
If we add the rule

\[
\Gamma \vdash \neg
\neg A \\
\frac{\Gamma \vdash \neg A}{\Gamma \vdash A} (\neg E)
\]

we obtain classical logic.

**Theorem**

*The boolean interpretation is correct and complete for classical logic.*
Part V

Dynamics
What makes programs interesting is that one can execute them.
What makes programs interesting is that one can execute them.

In \( \lambda \)-calculus, execution is called \( \beta \)-\textit{reduction} and consists in the rule

\[
(\lambda x. t) u \rightarrow_\beta t[x := u]
\]

which can be applied anywhere in a term.
Executing programs

What makes programs interesting is that one can execute them.

In λ-calculus, execution is called β-reduction and consists in the rule

\[(\lambda x. t) u \rightarrow_\beta t[x := u]\]

which can be applied anywhere in a term.

For instance, if we define

\[
\text{double} = \lambda x. x + x
\]

we have

\[
\text{double } 5 = (\lambda x. x + x) 5 \rightarrow_\beta 5 + 5
\]
We should also add rules for products:

\[ (\lambda x.t) \ u \rightarrow^\beta t[x := u] \]

\[ \pi_l \langle t, u \rangle \rightarrow^\beta t \]

\[ \pi_r \langle t, u \rangle \rightarrow^\beta u \]

A redex is a subterm which can reduce.
One of the most important properties of typing systems is called **subject reduction**:

**Theorem**

*If* $\Gamma \vdash t : A$ *is derivable and* $t \rightarrow_\beta t'$ *then* $\Gamma \vdash t' : A$ *is also derivable.*

It means that if a term is check to have type `int`, it will never reduce to "a".
One of the most important properties of typing systems is called **subject reduction**:

**Theorem**

If $\Gamma \vdash t : A$ is derivable and $t \rightarrow^\beta t'$ then $\Gamma \vdash t' : A$ is also derivable.

**Proof.**

Transform the proof of $\Gamma \vdash t : A$ into a proof of $\Gamma \vdash t' : A$.

It means that if a term is check to have type `int`, it will never reduce to "a".
Subject reduction: proof

Suppose that $\Gamma \vdash t : A$ and $t \rightarrow_{\beta} t'$ using the rule

$$\pi_1\langle u, v \rangle \rightarrow_{\beta} u$$
Subject reduction: proof

Suppose that $\Gamma \vdash t : A$ and $t \rightarrow_{\beta} t'$ using the rule

$$\pi_l(u, v) \rightarrow_{\beta} u$$

This means that the typing derivation of $t$ will contain a subproof of the form

$$\Gamma \vdash \pi_l(u, v) : A$$
Subject reduction: proof

Suppose that $\Gamma \vdash t : A$ and $t \rightarrow_\beta t'$ using the rule

$$\pi_1\langle u, v \rangle \rightarrow_\beta u$$

This means that the typing derivation of $t$ will contain a subproof of the form

$$\Gamma \vdash \langle u, v \rangle : A \times B \quad \frac{\quad \Gamma \vdash \pi_1\langle u, v \rangle : A}{\quad \Gamma \vdash \pi_1\langle u, v \rangle : A} \quad (\times_1^E)$$
Subject reduction: proof

Suppose that $\Gamma \vdash t : A$ and $t \rightarrow^\beta t'$ using the rule

$$\pi_1\langle u, v \rangle \rightarrow^\beta u$$

This means that the typing derivation of $t$ will contain a subproof of the form

\[
\begin{array}{c}
\vdash u : A \\
\vdash v : B \\
\hline
\vdash \langle u, v \rangle : A \times B \\
\end{array}
\]

$$(\times_1)$$

\[
\begin{array}{c}
\vdash \pi_1\langle u, v \rangle : A \\
\end{array}
\]

$$(\times_E)$$
Subject reduction: proof

Suppose that \( \Gamma \vdash t : A \) and \( t \overset{\beta}{\longrightarrow} t' \) using the rule

\[
\pi_1\langle u, v \rangle \overset{\beta}{\longrightarrow} u
\]

This means that the typing derivation of \( t \) will contain a subproof of the form

\[
\vdots \\
\Gamma \vdash u : A \\
\Gamma \vdash v : B \\
\hline
\Gamma \vdash \langle u, v \rangle : A \times B \\
\hline
\Gamma \vdash \pi_1\langle u, v \rangle : A \\
\hline
\vdots
\]

\((\times_l)\)  \((\times_E)\)  \(\rightsquigarrow\)

\[
\Gamma \vdash u : A
\]
Suppose that $\Gamma \vdash t : A$ and $t \rightarrow_{\beta} t'$ using the rule

$$\pi_r\langle u, v \rangle \rightarrow_{\beta} v$$

This means that the typing derivation of $t$ will contain a subproof of the form

\[
\begin{array}{c}
\vdots \\
\Gamma \vdash u : A \\
\vdash v : B \\
\vdash \langle u, v \rangle : A \times B \\
\vdash \pi_r\langle u, v \rangle : A \\
\end{array}
\]

\[
\begin{array}{c}
\vdash \langle u, v \rangle : A \\
\vdash \pi_r\langle u, v \rangle : A \\
\vdash \pi_r\langle u, v \rangle : A \\
\vdash \pi_r\langle u, v \rangle : A \\
\end{array}
\]

\[
\begin{array}{c}
\vdash \pi_r\langle u, v \rangle : A \\
\vdash \pi_r\langle u, v \rangle : A \\
\vdash \pi_r\langle u, v \rangle : A \\
\vdash \pi_r\langle u, v \rangle : A \\
\end{array}
\]

There is a “similar” transformation in the case where the rule is

\[
(\lambda x. t) u \rightarrow_{\beta} t[x := u]
\]
Suppose that $\Gamma \vdash t : A$ and $t \rightarrow_\beta t'$ using the rule

$$\pi_r\langle u, v \rangle \rightarrow_\beta v$$

This means that the typing derivation of $t$ will contain a subproof of the form

$$\vdots (\times_1) \Gamma \vdash \langle u, v \rangle : A \times B \vdots (\times_\pi\Gamma) \Gamma \vdash \pi_r\langle u, v \rangle : A \vdots$$

There is a “similar” transformation in the case where the rule is

$$(\lambda x. t)u \rightarrow_\beta t[x := u]$$
By the Curry-Howard correspondence, we can interpret these operations as proof transformations:

\[
\begin{align*}
\Gamma \vdash u : A & \quad \Gamma \vdash v : B \\
\Gamma \vdash \langle u, v \rangle : A \times B & \quad (\times_I) \\
\Gamma \vdash \pi_1 \langle u, v \rangle : A & \quad (\times_E) \\
\end{align*}
\]

Such a situation is called a **cut** (= introduction and elimination of a connective) and this transformation is called **cut elimination**.
By the Curry-Howard correspondence, we can interpret these operations as proof transformations:

\[
\begin{array}{c}
\vdash A \\
\vdash B \\
\vdash A \times B \\
\vdash A
\end{array}
\]

Such a situation is called a **cut** (\(=\) introduction and elimination of a connective) and this transformation is called **cut elimination**.
In the case of $\Rightarrow$, it relies on the fact that the following rule is admissible:

\[
\frac{\Gamma \vdash A \quad \Gamma, A \vdash B}{\Gamma \vdash B} \quad \text{(cut)}
\]

which corresponds to preservation of typing under substitutions.
In the case of \( \Rightarrow \), it relies on the fact that the following rule is admissible:

\[
\frac{\Gamma \vdash A \quad \Gamma, A \vdash B}{\Gamma \vdash B} \quad \text{(cut)}
\]

which corresponds to preservation of typing under substitutions.

Note that this rule has the following particular case:

\[
\frac{A \vdash B \quad B \vdash C}{A \vdash C} \quad \text{(cut)}
\]
The major theorem of proof theory:

**Theorem (Strong normalization)**

The reduction of any typable term \( \Gamma \vdash t : A \) eventually stops.
The major theorem of proof theory:

**Theorem (Strong normalization)**
The reduction of any typable term $\Gamma \vdash t : A$ eventually stops.

**Theorem (Cut elimination)**
If a sequent $\Gamma \vdash A$ is provable then it admits a proof without cut.
The major theorem of proof theory:

**Theorem (Strong normalization)**
*The reduction of any typable term* $\Gamma \vdash t : A$ *eventually stops.*

**Theorem (Cut elimination)**
*If a sequent* $\Gamma \vdash A$ *is provable then it admits a proof without cut.*

**Proof.**
Apply previous theorem through Curry-Howard.
The major theorem of proof theory:

**Theorem (Strong normalization)**

The reduction of any typable term $\Gamma \vdash t : A$ eventually stops.

**Theorem (Cut elimination)**

If a sequent $\Gamma \vdash A$ is provable then it admits a proof without cut.

**Proof.**

Apply previous theorem through Curry-Howard.

There are wonderful consequences of this (coherence, improvability of $\neg\neg A \Rightarrow A$, etc.) but this will be for another time.
The set-theoretic semantics is correct, in the sense that it is invariant under reduction (= cut elimination): given a derivation of $\Gamma \vdash t : A$ if $t \rightarrow_\beta t'$ then

$$\llbracket \Gamma \vdash t : A \rrbracket = \llbracket \Gamma \vdash t' : A \rrbracket$$

This can be seen as a modularity principle: the behavior of the whole program can be determined from its components only (and not the interactions they can have).
Definition
A semantics of a programming language is **denotational** when it is invariant under reduction (and “co-reduction” / extensionality).
**Definition**

A semantics of a programming language is **denotational** when it is invariant under reduction (and “co-reduction” / extensionality).

There are many alternatives to plain sets and functions:

- we might want to model more features: fixpoints \([\textbf{while}]\) (domains), equality (spaces / HoTT), etc.
- we might want to capture more precisely the language: the interactive behavior (game semantics), etc.
- we might want to remove all functions which are not interpretations of programs.
The “co-cuts” correspond to \( \eta \)-reduction which express some form of extensionality

\[
\begin{align*}
\lambda x. tx & \rightarrow_\eta t \\
\langle \pi_l t, \pi_r t \rangle & \rightarrow_\eta t
\end{align*}
\]
The “co-cuts” correspond to $\eta$-reduction which express some form of extensionality

$$\lambda x. tx \rightarrow_\eta t \quad \langle \pi_l t, \pi_r t \rangle \rightarrow_\eta t$$

On the typing side:

\[
\begin{align*}
\vdash t : A \times B \\
\vdash \pi_l t : A \\
\vdash \langle \pi_l t, \pi_r t \rangle : A \times B \\
\vdash \pi_r t : B
\end{align*}
\]
Part VI

Categorical semantics
Categorical axiomatization

Instead of checking each time that the model is denotational, people have studied categorical axiomatizations.

We define algebraic structures on categories which ensure that

- **operations**: there is a canonical interpretation of proofs in those categories,
- **axioms**: the interpretation of proofs is invariant under reduction.
Categories

A category $C$ consists of

- a set of objects $A, B, \ldots$
- a set of morphisms $C(A, B)$ for every objects $A$ and $B$
- a composition operations and identities

such that

- composition is associative: $h \circ (g \circ f) = (h \circ g) \circ f$
- identities are neutral elements: $id \circ f = f = f \circ id.$

Typically:

- the category $\text{Set}$ of sets and functions,
- $\text{Top}$, $\text{Vect}$, ...
The identity will correspond to the interpretation of

\[
\frac{}{A \vdash A} \quad \text{(ax)}
\]

and composition to

\[
\frac{A \vdash B \quad B \vdash C}{A \vdash C} \quad \text{(cut)}
\]
The identity will correspond to the interpretation of

\[ \frac{}{A \vdash A} \]  
(ax)

and composition to

\[ \frac{A \vdash B \quad B \vdash C}{A \vdash C} \]  
(cut)

The axioms of categories will ensure that the interpretation is invariant under cut-elimination.
In order to interpret conjunction, we need for every objects $A$ and $B$, an object $A \times B$:

$$[A \land B] = [A] \times [B]$$
Cartesian categories

In order to interpret conjunction, we need for every objects $A$ and $B$, an object $A \times B$:

$$[A \land B] = [A] \times [B]$$

We also need “projection” morphisms:

$$\pi_{A,B} : A \times B \to A$$
$$\pi'_{A,B} : A \times B \to B$$

which interpret

$$\frac{A \land B \vdash A \land B}{A \land B \vdash A} \quad (\land^l_E)$$
$$\frac{A \land B \vdash A \land B}{A \land B \vdash B} \quad (\land^r_E)$$
Cartesian categories

Given morphisms $f : C \to A$ and $g : C \to B$, we need a morphism $\langle f, g \rangle : C \to A \times B$:

$$\Gamma \vdash A \quad \Gamma \vdash B \quad \left(\land_i\right)$$

$$\Gamma \vdash A \land B$$
Given morphisms $f : C \to A$ and $g : C \to B$, we need a morphism $\langle f, g \rangle : C \to A \times B$:

$$
\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \land B} \quad (\land_I)
$$

Moreover, invariance under cut elimination

$$
\vdots \\
\frac{\Gamma \vdash A}{\Gamma \vdash A \land B} \quad (\land_I)
\frac{\Gamma \vdash A \land B}{\Gamma \vdash A} \quad (\land_E) \\
\vdots
\rightsquigarrow
\frac{\Gamma \vdash A}{\Gamma \vdash A}
$$

will impose

$$
\pi \circ \langle f, g \rangle = f
$$
Cartesian categories

Given morphisms $f : C \to A$ and $g : C \to B$, we need a morphism $\langle f, g \rangle : C \to A \times B$:

$$
\Gamma \vdash A \quad \Gamma \vdash B \\
\Gamma \vdash A \land B \quad (\land_1)
$$

Moreover, invariance under cut elimination

$$
\vdots \\
\Gamma \vdash A \\
\Gamma \vdash A \land B \quad (\land_1) \\
\Gamma \vdash A \quad (\land_E) \\
\vdots \\n\vdots 
$$

will impose

$$
\pi \circ \langle f, g \rangle = f \\
\pi' \circ \langle f, g \rangle = g
$$
Given morphisms \( f : C \to A \) and \( g : C \to B \), we need a morphism \( \langle f, g \rangle : C \to A \times B \):

\[
\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \land B} \quad (\land_1)
\]

Moreover, invariance under cut elimination

\[
\vdash \frac{\vdash \Gamma \vdash A \land B \quad \vdash \Gamma \vdash A}{\vdash \Gamma \vdash A} \quad (\land_1) \quad \rightsquigarrow \quad \vdash \frac{\vdash \Gamma \vdash A \land B \quad \vdash \Gamma \vdash A}{\vdash \Gamma \vdash A} \quad (\land_E)
\]

will impose

\[
\pi \circ \langle f, g \rangle = f
\]

\[
\pi' \circ \langle f, g \rangle = g
\]

\[
\langle \pi \circ f, \pi' \circ f \rangle = f
\]
A **cartesian product** of $A$ and $B$ in a category $C$ is
In order to interpret our fragment of logic we need a cartesian closed category:

- a category
In order to interpret our fragment of logic we need a **cartesian closed category**:

- a *category*
- with *cartesian products*
Cartesian closed categories

In order to interpret our fragment of logic we need a **cartesian closed category**:

- a *category*
- with *cartesian products*
- with a *terminal object* $1$ to interpret $\top$:

\[ A \quad \vdash \quad 1 \]
In order to interpret our fragment of logic we need a **cartesian closed category**:

- a *category*
- with *cartesian products*
- with a *terminal object* $1$ to interpret $\top$:

$$A \rightarrow 1$$

- with an *exponential closure* to interpret $\Rightarrow$:

$$\mathcal{C}(A \times B, C) \cong \mathcal{C}(A, B \rightarrow C)$$
Theorem (Soundness)
We have a denotational semantics of our logic in every cartesian closed category.
Theorem (Soundness)
We have a denotational semantics of our logic in every cartesian closed category.

Theorem (Completeness)
Every categorical model of our logic has to be cartesian closed.
**Theorem (Soundness)**
*We have a denotational semantics of our logic in every cartesian closed category.*

**Theorem (Completeness)**
*Every categorical model of our logic has to be cartesian closed.*
Part VII

Linear logic
Reasoning about resources

A λ-term can use its argument many times:

\[ \lambda x.x + x \]

including zero times:

\[ \lambda x.0 \]
Reasoning about resources

A \( \lambda \)-term can use its argument many times:

\[
\lambda x. x + x
\]

including zero times:

\[
\lambda x. 0
\]

**Linear logic**: we would like to distinguish between variables which can be used exactly once or not in order to

- take resources in account,
- take complexity in account,
- have a fine grained understanding of logic.
Let's try to modify the rules for the \( \lambda \)-calculus in order to capture **linear \( \lambda \)-terms** only, where each variable is used **exactly once**.
Let's try to modify the rules for the λ-calculus in order to capture linear λ-terms only, where each variable is used exactly once.

\[ \lambda x.\lambda y.(x, y) \quad \lambda x.\lambda y.xy \quad \lambda x.\lambda y.x \quad \lambda x.\lambda y.xyy \]
Let's try to modify the rules for the \( \lambda \)-calculus in order to capture \textbf{linear} \( \lambda \)-\textbf{terms} only, where each variable is used \underline{exactly once}.

\begin{align*}
\lambda x.\lambda y.\langle x, y \rangle & \quad \lambda x.\lambda y.yx & \quad \lambda x.\lambda y.x & \quad \lambda x.\lambda y.yyy \\
\text{linear} & & & 
\end{align*}
Let's try to modify the rules for the \( \lambda \)-calculus in order to capture linear \( \lambda \)-terms only, where each variable is used exactly once.

\[
\begin{align*}
\lambda x.\lambda y.\langle x, y \rangle & \quad \lambda x.\lambda y.\langle x, y \rangle & \lambda x.\lambda y.x & \quad \lambda x.\lambda y.xyy \\
\text{linear} & \quad \text{linear} & \text{linear} & \text{not linear}
\end{align*}
\]
Let's try to modify the rules for the $\lambda$-calculus in order to capture **linear $\lambda$-terms** only, where each variable is used **exactly once**.

$\lambda x.\lambda y.\langle x, y \rangle$
- linear

$\lambda x.\lambda y.yx$
- linear

$\lambda x.\lambda y.x$
- not linear

$\lambda x.\lambda y.xyy$
Let's try to modify the rules for the $\lambda$-calculus in order to capture linear $\lambda$-terms only, where each variable is used exactly once.

\[
\begin{align*}
\lambda x.\lambda y.\langle x, y \rangle & \quad \text{linear} \\
\lambda x.\lambda y.yx & \quad \text{linear} \\
\lambda x.\lambda y.x & \quad \text{not linear} \\
\lambda x.\lambda y.xyy & \quad \text{not linear}
\end{align*}
\]
Let's try to modify the rules for the \( \lambda \)-calculus in order to capture \textbf{linear \( \lambda \)-terms} only, where each variable is used \textbf{exactly once}.

\[
\begin{align*}
\lambda x. \lambda y. \langle x, y \rangle & \quad \lambda x. \lambda y. yx & \quad \lambda x. \lambda y. x & \quad \lambda x. \lambda y. xyy \\
\text{linear} & \quad \text{linear} & \quad \text{not linear} & \quad \text{not linear}
\end{align*}
\]

The main idea is that in sequents, the context \( \Gamma \) maintains the variables we are aware of, which can be thought of as resources that we should handle carefully.
A rule is **admissible** when if we can derive the premises then we can derive the conclusion.
A rule is **admissible** when if we can derive the premises then we can derive the conclusion.

**Lemma**
*The following contraction rule is admissible:*

\[
\Gamma, A, A \vdash B \\
\Gamma, A \vdash B
\]
Lemma
The following contraction rule is admissible:

\[
\frac{\Gamma, A, A \vdash B}{\Gamma, A \vdash B}
\]

Categorically,

\[
A \times A \xrightarrow{f} B
\]
Lemma

*The following contraction rule is admissible:*

\[ \Gamma, A, A \vdash B \]

\[ \Gamma, A \vdash B \]

Categorically,

\[ A \times A \xrightarrow{f} B \]

\[ A \]
Lemma

The following contraction rule is admissible:

\[ \frac{\Gamma, A, A \vdash B}{\Gamma, A \vdash B} \]

Categorically,
Contraction

Lemma

The following contraction rule is admissible:

\[
\frac{\Gamma, A, A \vdash B}{\Gamma, A \vdash B}
\]

Categorically,

\[
\begin{array}{c}
A \\
\xrightarrow{\delta} A \times A \\
\xrightarrow{\pi'} A
\end{array}
\xrightarrow{f}
\begin{array}{c}
A \\
\xrightarrow{id_A} A
\end{array}
\]

In sets: \( \delta(x) = (x, x) \).

This means that from \((x, y) \mapsto f(x, y)\), we can construct \(x \mapsto f(x, x)\)!
Lemma
The following contraction rule is admissible:

\[
\frac{\Gamma, A, A \vdash B}{\Gamma, A \vdash B}
\]

Categorically,

\[
\begin{array}{ccc}
A & \xrightarrow{\delta} & A \times A \\
\downarrow{\pi} & & \rightarrow_{f} & \downarrow{\pi'} & B \\
& & \downarrow{\pi'} & & \\
A & \xrightarrow{id_{A}} & A
\end{array}
\]

Closely related to the fact that we have the proof

\[
\frac{A \vdash A}{A \vdash A} \quad \frac{A \vdash A}{A \vdash A} \quad \frac{A \vdash A}{A \vdash A \land A} \quad (\land I).
\]
In linear logic, the traditional *additive* conjunction is noted \( \& \)

\[
\begin{align*}
\Gamma \vdash A \& B & \quad \frac{\Gamma \vdash A \& B}{\Gamma \vdash A} (\&_E) \\
\Gamma \vdash A \& B & \quad \frac{\Gamma \vdash A \& B}{\Gamma \vdash B} (\&_E) \\
\Gamma \vdash A \quad \Gamma \vdash B & \quad \frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \& B} (\&_I)
\end{align*}
\]
In linear logic, the traditional *additive* conjunction is noted $\&$

\[
\frac{\Gamma \vdash A \& B}{\Gamma \vdash A} (\&_E) \quad \frac{\Gamma \vdash A \& B}{\Gamma \vdash B} (\&_E) \quad \frac{\Gamma \vdash A}{\Gamma \vdash A \& B} (\&_I)
\]

It allows deducing

\[
\frac{A \vdash A}{A \vdash A \& A} (\&_I) \quad \frac{A \vdash A}{A \vdash A} (ax)
\]

\[
\frac{A \vdash A}{A \vdash A \& A} (ax)
\]
In linear logic, the traditional *additive* conjunction is noted \&

\[
\begin{align*}
& \frac{}{\Gamma \vdash \mathbf{A} \& \mathbf{B}} \quad (\&_E^l) \\
& \frac{}{\Gamma \vdash \mathbf{A}} \quad (\&_E^r) \\
& \frac{}{\Gamma \vdash \mathbf{A} \& \mathbf{B}} \quad (\&_1)
\end{align*}
\]

It allows deducing

\[
\begin{align*}
& \frac{}{\mathbf{A} \vdash \mathbf{A}} \quad (\text{ax}) \\
& \frac{}{\mathbf{A} \vdash \mathbf{A}} \quad (\text{ax}) \\
& \frac{}{\mathbf{A} \vdash \mathbf{A} \& \mathbf{A}} \quad (\&_1)
\end{align*}
\]

We want to replace it by a *multiplicative* conjunction \(\otimes\) which does not allow this

\[
\begin{align*}
& \frac{}{\Gamma \vdash \mathbf{A}} \\
& \frac{}{\Gamma' \vdash \mathbf{B}} \\
& \frac{}{\Gamma, \Gamma' \vdash \mathbf{A} \otimes \mathbf{B}} \quad (\otimes_1)
\end{align*}
\]
In linear logic, the traditional *additive* conjunction is noted &

\[\frac{\Gamma \vdash A \& B}{\Gamma \vdash A} (\&_E)\]

\[\frac{\Gamma \vdash A \& B}{\Gamma \vdash B} (\&_E)\]

\[\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \& B} (\&_I)\]

It allows deducing

\[\frac{A \vdash A}{A \vdash A \& A} (\&_I)\]

We want to replace it by a *multiplicative* conjunction \(\otimes\) which does not allow this

\[\frac{\Gamma \vdash A \otimes B}{\Gamma \vdash A} (\otimes_E)\]

\[\frac{\Gamma', A, B \vdash C}{\Gamma, \Gamma' \vdash C} (\otimes_E)\]

\[\frac{\Gamma \vdash A}{\Gamma, \Gamma' \vdash A \otimes B} (\otimes_1)\]

\[\frac{\Gamma' \vdash B}{\Gamma, \Gamma' \vdash A \otimes B} (\otimes_1)\]
Similarly, we change the rules for implication to

\[
\Gamma \vdash A \rightarrow B \quad \Gamma' \vdash A \\
\underline{\quad \Gamma, \Gamma' \vdash B \quad} \quad (\rightarrow_E) \\
\Gamma, A \vdash B \\
\underline{\quad \Gamma \vdash A \rightarrow B \quad} \quad (\rightarrow_I)
\]
We replace the axiom rule

\[
\Gamma, A \vdash A \quad \text{(ax)}
\]

by

\[
A \vdash A \quad \text{(ax)}
\]
We replace the axiom rule

\[ \Gamma, A \vdash A \] (ax)

by

\[ A \vdash A \] (ax)

Similarly, we replace

\[ \Gamma \vdash \top \] (\top_I)

by

\[ \vdash 1 \] (1_I)
We also need to allow exchanging hypothesis in the context

\[
\Gamma, B, A, \Gamma' \vdash C
\]

\[
\Gamma, A, B, \Gamma' \vdash C
\]
The simply-typed linear λ-calculus

**Theorem**

The λ-terms typable with $\otimes$, $1$ and $\multimap$ are precisely the λ-terms which

- are typable in the previous sense and
- linear.
The simply-typed linear $\lambda$-calculus

The models of this logic are symmetric monoidal closed categories:

- **monoidal**: there is an object $1$, and objects and morphisms equipped with a binary operation $\otimes$ together with isomorphisms

\[ \alpha_{A,B,C} : (A \otimes B) \otimes C \simeq A \otimes (B \otimes C) \quad \lambda_A : 1 \otimes A \simeq A \]
\[ \rho_A : A \otimes 1 \simeq A \]

satisfying axioms
The simply-typed linear $\lambda$-calculus

The models of this logic are symmetric monoidal closed categories:

- **monoidal**: there is an object $1$, and objects and morphisms equipped with a binary operation $\otimes$ together with isomorphisms

  $$\alpha_{A,B,C} : (A \otimes B) \otimes C \simeq A \otimes (B \otimes C)$$

  $$\lambda_A : 1 \otimes A \simeq A$$

  $$\rho_A : A \otimes 1 \simeq A$$

  satisfying axioms

- **symmetric**:

  $$\gamma_{A,B} : B \otimes A \to A \otimes B$$
The simply-typed linear $\lambda$-calculus

The models of this logic are symmetric monoidal closed categories:

- **monoidal**: there is an object $1$, and objects and morphisms equipped with a binary operation $\otimes$ together with isomorphisms

  $\alpha_{A,B,C} : (A \otimes B) \otimes C \simeq A \otimes (B \otimes C)$

  $\lambda_A : 1 \otimes A \simeq A$

  $\rho_A : A \otimes 1 \simeq A$

  satisfying axioms

- **symmetric**: 

  $\gamma_{A,B} : B \otimes A \rightarrow A \otimes B$

- **closed**: there is a closure

  $\text{Hom}(A \otimes B, C) \simeq \text{Hom}(A, B \rightarrow C)$
Every typing derivation of a linear \( \lambda \)-term is a valid non-linear one: in terms of models, we have that every cartesian closed category is a symmetric monoidal closed category.
The simply-typed linear $\lambda$-calculus

Every typing derivation of a linear $\lambda$-term is a valid non-linear one: in terms of models, we have that every cartesian closed category is a symmetric monoidal closed category.

Conversely, a symmetric monoidal closed category is a cartesian closed one when we have the ability of duplicating and erasing objects:
The simply-typed linear $\lambda$-calculus

Every typing derivation of a linear $\lambda$-term is a valid non-linear one: in terms of models, we have that every cartesian closed category is a symmetric monoidal closed category.

Conversely, a symmetric monoidal closed category is a cartesian closed one when we have the ability of duplicating and erasing objects:

**Theorem**

A monoidal category is cartesian precisely when for every object $A$ is equipped with

$$\delta_A : A \to A \otimes A \quad \varepsilon_1 : A \to 1$$

compatible with other morphisms and coassociative, counital and cocommutative:

$$A \xrightarrow{\delta_A} A \otimes A \xrightarrow{\delta_A \otimes A} (A \otimes A) \otimes A \xrightarrow{\sim} A \otimes (A \otimes A) \xleftarrow{A \otimes \delta_A} A \otimes (A \otimes A) \xrightarrow{\sim} 1 \otimes A \xleftarrow{\varepsilon_A \otimes A} A \otimes A \xrightarrow{A \otimes \varepsilon_A} A \otimes 1$$
Linear logic builds on these observations and incorporates both worlds:

<table>
<thead>
<tr>
<th>conjunction</th>
<th>multiplicative</th>
<th>additive</th>
</tr>
</thead>
<tbody>
<tr>
<td>⊗</td>
<td>⊙</td>
<td>&amp;</td>
</tr>
</tbody>
</table>

It can be shown that we can define $A \multimap B = A^* \otimes B$ (akin $A \Rightarrow B = \neg A \lor B$ in classical logic).
**Linear logic** builds on these observations and incorporates both worlds:

<table>
<thead>
<tr>
<th></th>
<th>conjunction</th>
<th>disjunction</th>
</tr>
</thead>
<tbody>
<tr>
<td>multiplicative</td>
<td>⊗</td>
<td>⨿</td>
</tr>
<tr>
<td>additive</td>
<td>&amp;</td>
<td>⊕</td>
</tr>
</tbody>
</table>

It can be shown that we can define $A \multimap B = A^* \otimes B$ (akin to $A \Rightarrow B = \neg A \lor B$ in classical logic).
**Linear logic** builds on these observations and incorporates both worlds:

<table>
<thead>
<tr>
<th></th>
<th>conjunction</th>
<th>disjunction</th>
</tr>
</thead>
<tbody>
<tr>
<td>multiplicative</td>
<td>$\otimes$</td>
<td>$\wp$</td>
</tr>
<tr>
<td>additive</td>
<td>$&amp;$</td>
<td>$\oplus$</td>
</tr>
</tbody>
</table>

We write $A^*$ for the **dual** of a formula:

$$(A \otimes B)^* = A^* \wp B^* \quad (A \wp B)^* = A^* \otimes B^* \quad (A \& B)^* = A^* \oplus B^* \quad \ldots$$
Linear logic builds on these observations and incorporates both worlds:

<table>
<thead>
<tr>
<th></th>
<th>conjunction</th>
<th>disjunction</th>
</tr>
</thead>
<tbody>
<tr>
<td>multiplicative</td>
<td>$\otimes$</td>
<td>$\multimap$</td>
</tr>
<tr>
<td>additive</td>
<td>$&amp;$</td>
<td>$\oplus$</td>
</tr>
</tbody>
</table>

We write $A^*$ for the dual of a formula:

$$(A \otimes B)^* = A^* \multimap B^* \quad (A \multimap B)^* = A^* \otimes B^* \quad (A \& B)^* = A^* \oplus B^* \quad \ldots$$

It can be shown that we can define

$$A \multimap B = A^* \multimap B$$

(akin $A \Rightarrow B = \neg A \lor B$ in classical logic).
The exponential

In order to relate both worlds, linear logic introduces a last connective the exponential $!$. We will see the formal rules later on (if we have time), but we should have maps $!A \multimap A$.

$$!A \equiv \bigotimes_{n=0} A \otimes n / \sim$$
In order to relate both worlds, linear logic introduces a last connective the **exponential** $!$ (and its dual $?$).
The exponential

In order to relate both worlds, linear logic introduces a last connective the exponential $!$ (and its dual $?$).

The intuition is that $!A$ is an $A$ which an be used any number of times, something like

$$!A = \bigotimes_{n=0} A^\otimes n / \sim$$

we will see the formal rules later on (if we have time), but we should have maps

$$!A \rightarrow A \quad \quad \quad !A \rightarrow !!A$$
The exponential

The main properties of exponential are that

- it turns additives into multiplicatives:

\[ !(A \& B) \dashv \vdash !A \otimes !B \]
The exponential

The main properties of exponential are that

- it turns additives into multiplicatives:

\[ !(A \land B) \quad \dashv \vdash \quad !A \otimes !B \]

- we can recover the usual implication as

\[ A \Rightarrow B \quad = \quad !A \twoheadrightarrow B \]
Part VIII

The relational model
In order to gain intuition, we can build the following model of the logic we have so far, in the category \textbf{Rel} of sets and relations.

We interpret $A$ as a set:

- $[A \otimes B] = [A \& B] = [A \imply B] = [A] \times [B]
- [A \& B] = [A \oplus B] = [A] \sqcup [B]$
The relational model

In order to gain intuition, we can build the following model of the logic we have so far, in the category $\text{Rel}$ of sets and relations.

We interpret $A$ as a set:

- $[A \otimes B] = [A \triangleright B] = [A \rightarrow B] = [A] \times [B]$ 
- $[A \& B] = [A \oplus B] = [A] \sqcup [B]$ 

A proof $\pi$ of $A \vdash B$ is interpreted as a relation between $[A]$ and $[B]$:

$$[\pi] \subseteq [A] \times [B]$$

(intuitively, $(a, b) \in [\pi]$ means that $b$ is a possible result for $a$)
The relational model: axiom

The axiom

\[
A \vdash A \quad \text{(ax)}
\]

is interpreted as the identity relation:

\[
[A \vdash A] = \{(a, a) \mid a \in [A]\} \subseteq [A] \times [A]
\]
For the cut rule

\[
\begin{array}{c}
\rho \\
A \vdash B
\end{array} \quad \begin{array}{c}
\sigma \\
B \vdash C
\end{array}
\begin{array}{c}
\hline \\
A \vdash C
\end{array}
\text{(cut)}
\]
The relational model: cut

For the cut rule

\[
\frac{\rho}{A \vdash B} \quad \frac{\sigma}{B \vdash C} \quad (\text{cut})
\]

\[\frac{A \vdash C}{A \vdash C}\]

given the respective interpretations

\[R \subseteq [A] \times [B]\]
\[S \subseteq [B] \times [B]\]

of \(\rho\) and \(\sigma\),
For the cut rule

\[
\frac{\rho}{A \vdash B} \quad \frac{\sigma}{B \vdash C} \quad (\text{cut})
\]

given the respective interpretations

\[R \subseteq [A] \times [B] \quad S \subseteq [B] \times [B]\]

of \(\rho\) and \(\sigma\), we define the interpretation of the proof as

\[S \circ R = \{(a, c) \in [A] \times [C] \mid (a, b) \in R \text{ and } (b, c) \in S\}\]
We thus interpret proof in the category \( \text{Rel} \) with
- sets as objects,
- relations as morphisms,
- compositions and identities defined as above.
The relational model

We thus interpret proof in the category $\text{Rel}$ with

- sets as objects,
- relations as morphisms,
- compositions and identities defined as above.

When we interpret

$[A \otimes B] = [A] \times [B]$ \hspace{1cm} [A & B] = [A] \sqcup [B]$

The cartesian product (in terms of the categorical property) is the second one, the first only induces a monoidal structure.
The **exponential** \(!A\) is interpreted as

\[
[!A] = \mathcal{M}_{\text{fin}}([A])
\]

the set of finite multisets of elements of \([A]\), i.e. lists of elements of \([A]\) considered up to permutation.
The exponential $!A$ is interpreted as

$$[!A] = \mathcal{M}_{\text{fin}}([A])$$

the set of finite multisets of elements of $[A]$, i.e. lists of elements of $[A]$ considered up to permutation.

For instance, we can check that we have the required isomorphism

$$[!(A \& B)] \simeq [!A \otimes !B]$$
Enriching over sets

This model is not the most informative (every connective is interpreted as its dual, it is not fully abstract, etc.).

A refined variant of this it can be obtained as follows.

The interpretation of a proof $\pi$ of $A \vdash B$ is

$$[\pi] \subseteq [A] \times [B]$$

and $(a, b) \in [\pi]$ means that $b$ is a possible result for $a$, but we do not track why!
A relation

\[ R \subseteq X \times Y \]

can alternatively be seen as a function

\[(X \times Y) \to \{0, 1\}\]
A relation

\[ R \subseteq X \times Y \]

can alternatively be seen as a function

\[(X \times Y) \to \{0, 1\}\]

We should get a better behaved model by switching to functions

\[(X \times Y) \to \text{Set}\]
We can extend previous constructions in order to get a “model” of linear logic.
Enriching over sets

We can extend previous constructions in order to get a “model” of linear logic.

Given two “relations”

\[ R : X \times Y \to \text{Set} \quad \quad \quad \quad S : Y \times Z \to \text{Set} \]

their composite is given by

\[ S \circ R(x, z) = \bigsqcup_{y \in Y} R(x, y) \times S(y, z) \]

it is not strictly associative but only associative up to isomorphism.
Enriching over sets

We can extend previous constructions in order to get a “model” of linear logic.

Given two “relations”

\[ R : X \times Y \rightarrow \text{Set} \quad \text{and} \quad S : Y \times Z \rightarrow \text{Set} \]

their composite is given by

\[ S \circ R(x, z) = \bigsqcup_{y \in Y} R(x, y) \times S(y, z) \]

it is not strictly associative but only associative up to isomorphism.

We thus get a (bi)category of sets and “generalized relations”, which corresponds to Girard’s model of normal functors or Kock’s polynomial functors.
In this model, the functions

\[
A \Rightarrow B = !A \to B
\]

are interpreted as “relations” \( R \) in

\[
(\mathcal{M}_{\text{fin}}[A] \times [B]) \to \text{Set}
\]
In this model, the functions

\[ A \Rightarrow B = \! A \rightarrow B \]

are interpreted as “relations” \( R \) in

\[ (\mathcal{M}_{\text{fin}}[A] \times [B]) \rightarrow \text{Set} \]

An element of

\[ R(a_1a_2\ldots a_n, b) \]

can be interpreted as an operation

and composition corresponds to the expected composition of trees.
Generalized species

The exponential is interpreted as

\[
!A = \mathcal{M}_{\text{fin}}([A]) = \{a_1 \ldots a_n \in A^* \mid n \in \mathbb{N}\}/\sim
\]

where we identify

\[
a_1 \ldots a_n \sim a_{\sigma(1)} \ldots a_{\sigma(n)}
\]

for every permutation \(\sigma \in \Sigma_n\).
The exponential is interpreted as

\[ !A = \mathcal{M}_{\text{fin}}([A]) = \{ a_1 \ldots a_n \in A^* \mid n \in \mathbb{N} \} / \sim \]

where we identify

\[ a_1 \ldots a_n \sim a_{\sigma(1)} \ldots a_{\sigma(n)} \]

for every permutation \( \sigma \in \Sigma_n \).

We would like to avoid quotienting and keep this action explicit.
This suggests another generalization of the model [Fiore, Gambino, Hyland]:

- we model objects as groupoids,
- morphisms are functors (= profunctors)

\[(X \times Y) \to \text{Set}\]

- the exponential \(!X\) is the free symmetric monoidal category on \(X\).
In particular, morphisms $!1 \rightarrow 1$

- where functions $\mathbb{N} \rightarrow \text{Set}$ in previous model,
- are now functors $\text{Bij} \rightarrow \text{Set}$.

In this sense morphisms $!A \rightarrow B$ are generalized species.

This suggests studying categorical structures present in this category, as well as further generalizations of it.
Questions?
Part IX

Rules for linear logic
Sequent calculus

In order to formulate the usual rules of linear logic, we perform two changes:

- we use sequent calculus instead of natural deduction: we change

\[
\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \& B} (\&_R)
\]

to

\[
\frac{\Gamma, A \vdash \Delta}{\Gamma, A \& B \vdash C} (\&_L) \quad \frac{\Gamma, B \vdash \Delta}{\Gamma, A \& B \vdash C} (\&_L') \quad \frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \& B} (\&_R)
\]
In order to formulate the usual rules of linear logic, we perform two changes:

- we use sequent calculus instead of natural deduction: we change

  \[ \frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \& B} (\&_R) \]

  to

  \[ \frac{\Gamma, A \vdash \Delta}{\Gamma, A \& B \vdash C} (\&_L) \quad \frac{\Gamma, B \vdash \Delta}{\Gamma, A \& B \vdash C} (\&'_L) \]

  \[ \frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \& B} (\&_R) \]

- we shift to classical logic by allowing multiple sequents on the right: sequents are of the form

  \[ \Gamma \vdash \Delta \]
Linear logic: categorical rules

Categorical rules:

\[
\begin{align*}
\Gamma & \vdash \Gamma \quad \text{(ax)} \\
\Gamma & \vdash A, \Delta \\
\Gamma', A & \vdash B, \Delta' \\
\Gamma, \Gamma' & \vdash B, \Delta, \Delta' \\
\end{align*}
\]

Structural rules:

\[
\begin{align*}
\Gamma, B, A, \Gamma' & \vdash \Delta \\
\Gamma, A, B, \Gamma' & \vdash \Delta \\
\Gamma & \vdash \Delta, B, A, \Delta' \\
\Gamma & \vdash \Delta, A, B, \Delta' \\
\end{align*}
\]
Linear logic: multiplicative rules

Multiplicative conjunction:

\[
\frac{\Gamma, A, B \vdash \Delta}{\Gamma, A \otimes B \vdash \Delta} \quad (\otimes_L) \\
\frac{\Gamma \vdash A, \Delta \quad \Gamma' \vdash B, \Delta}{\Gamma, \Gamma' \vdash A \otimes B, \Delta, \Delta'} \quad (\otimes_R)
\]

Multiplicative truth:

\[
\frac{\Gamma \vdash A}{\Gamma, 1 \vdash A} \quad (1_L) \\
\vdash 1 \quad (1_R)
\]

Multiplicative disjunction:

\[
\frac{\Gamma, A \vdash \Delta \quad \Gamma', B \vdash \Delta'}{\Gamma, \Gamma', A \parr B \vdash \Delta, \Delta'} \quad (\parr_L) \\
\frac{\Gamma \vdash A, B, \Delta}{\Gamma \vdash A \parr B, \Delta} \quad (\parr_R)
\]

Multiplicative falsity:

\[
\vdash \bot \quad (\bot_L) \\
\frac{\Gamma \vdash \Delta}{\Gamma \vdash \bot, \Delta} \quad (\bot_R)
\]
Linear logic: additives

Additive conjunction:

\[
\begin{align*}
\Gamma, A \vdash \Delta & \quad \Gamma, B \vdash \Delta \\
\frac{}{\Gamma, A \& B \vdash \Delta} (\&_L) & \quad \frac{}{\Gamma, A \& B \vdash \Delta} (\&_R)
\end{align*}
\]

Additive truth:

\[
\frac{}{\Gamma \vdash \top, \Delta} (\top_R)
\]

Additive disjunction:

\[
\begin{align*}
\Gamma, A \vdash \Delta & \quad \Gamma, B \vdash \Delta \\
\frac{}{\Gamma, A \oplus B \vdash \Delta} (\oplus_L) & \quad \frac{}{\Gamma \vdash A \oplus B, \Delta} (\oplus^R) & \quad \frac{}{\Gamma \vdash A \oplus B, \Delta} (\oplus^R)
\end{align*}
\]

Additive falsity:

\[
\frac{}{\Gamma, 0 \vdash \Delta} (0_L)
\]
Linear logic: exponentials

Bang:

\[
\frac{\Gamma, A \vdash \Delta}{\Gamma, !A \vdash \Delta} \quad \frac{\Gamma \vdash \Delta}{\Gamma, !A \vdash \Delta} \quad \frac{\Gamma, !A, !A \vdash \Delta}{\Gamma, !A \vdash \Delta} \quad \frac{!\Gamma \vdash A, ?\Delta}{!\Gamma \vdash !A, ?\Delta}
\]

Maybe:

\[
\frac{\Gamma \vdash A, \Delta}{\Gamma \vdash ?A, \Delta} \quad \frac{\Gamma \vdash \Delta}{\Gamma \vdash ?A, \Delta} \quad \frac{\Gamma \vdash ?A, ?A, \Delta}{\Gamma \vdash ?A, \Delta} \quad \frac{!\Gamma, A \vdash ?\Delta}{!\Gamma, ?A \vdash ?\Delta}
\]

(dereliction / weakening / contraction / promotion)