CS 4110

Programming Languages & Logics

Lecture 28 Propositions as Types

7 November 2016

Propositions as Types

Logics = Type Systems

Let's start with *constructive logic*, where the rules work like functions that take smaller proofs and generate larger proofs.

Let's start with *constructive logic*, where the rules work like functions that take smaller proofs and generate larger proofs.

Here's a rule from *natural deduction*, a constructive logic invented by logician Gerhard Gentzen in 1935:

$$\frac{\phi \quad \psi}{\phi \wedge \psi} \wedge \text{-intro}$$

Given a proof of ϕ and a proof of ψ , it lets you *construct* a proof of $\phi \wedge \psi$.

In natural deduction, we define the set of true formulas ("theorems") inductively.

In natural deduction, we define the set of true formulas ("theorems") inductively.

Φ

We'll start with a grammar for formulas:

$$\begin{array}{c} ::= & \top \\ & \mid & \bot \\ & \mid & X \\ & \mid & \phi \land \psi \\ & \mid & \phi \lor \psi \\ & \mid & \phi \to \psi \\ & \mid & \neg \phi \\ & \mid & \forall x. \phi \end{array}$$

where X ranges over Boolean variables and $\neg \phi$ is an abbreviation for $\phi \rightarrow \bot$.

Let's define a judgment that that a formula is true given a set of assumptions Γ :

 $\Gamma \vdash \phi$

where Γ is just a list of formulas.

Let's define a judgment that that a formula is true given a set of assumptions Γ :

 $\mathsf{\Gamma}\vdash\phi$

where Γ is just a list of formulas.

If $\vdash \phi$ (with no assumptions), we say ϕ is a *theorem*.

Examples:

• $\vdash A \land B \rightarrow A$

Let's define a judgment that that a formula is true given a set of assumptions Γ :

 $\mathsf{\Gamma}\vdash\phi$

where Γ is just a list of formulas.

If $\vdash \phi$ (with no assumptions), we say ϕ is a *theorem*.

Examples:

- $\vdash A \land B \rightarrow A$
- $\vdash \neg (A \land B) \rightarrow \neg A \lor \neg B$

Let's define a judgment that that a formula is true given a set of assumptions Γ :

 $\mathsf{\Gamma}\vdash\phi$

where Γ is just a list of formulas.

If $\vdash \phi$ (with no assumptions), we say ϕ is a *theorem*.

Examples:

- $\vdash A \land B \rightarrow A$
- $\vdash \neg (A \land B) \rightarrow \neg A \lor \neg B$
- $A, B, C \vdash B$

Let's write the rules for our judgment:

$$\frac{\Gamma \vdash \phi \quad \Gamma \vdash \psi}{\Gamma \vdash \phi \land \psi} \land \text{-intro}$$

Let's write the rules for our judgment:

$$\frac{\Gamma \vdash \phi \quad \Gamma \vdash \psi}{\Gamma \vdash \phi \land \psi} \land \text{-intro}$$

$$\frac{\Gamma \vdash \phi \land \psi}{\Gamma \vdash \phi} \land \text{-Elim} \qquad \qquad \frac{\Gamma \vdash \phi \land \psi}{\Gamma \vdash \psi} \land \text{-Elim} 2$$

Let's write the rules for our judgment:

$$\frac{\Gamma \vdash \phi \quad \Gamma \vdash \psi}{\Gamma \vdash \phi \land \psi} \land \text{-intro}$$

$$\frac{\Gamma \vdash \phi \land \psi}{\Gamma \vdash \phi} \land \text{-Elim1} \qquad \qquad \frac{\Gamma \vdash \phi \land \psi}{\Gamma \vdash \psi} \land \text{-Elim2}$$

$$\frac{\Gamma, \phi \vdash \psi}{\Gamma \vdash \phi \to \psi} \to \text{-intro}$$

Let's write the rules for our judgment:

$$\frac{\Gamma \vdash \phi \quad \Gamma \vdash \psi}{\Gamma \vdash \phi \land \psi} \land \text{-intro}$$

$$\frac{\Gamma \vdash \phi \land \psi}{\Gamma \vdash \phi} \land \text{-ELIM1} \qquad \qquad \frac{\Gamma \vdash \phi \land \psi}{\Gamma \vdash \psi} \land \text{-ELIM2}$$

$$\frac{\Gamma, \phi \vdash \psi}{\Gamma \vdash \phi \to \psi} \to \text{-intro}$$

...and so on.

$$\frac{\Gamma, \phi \vdash \psi}{\Gamma \vdash \phi \rightarrow \psi} \xrightarrow{} -\text{INTRO} \qquad \frac{\Gamma \vdash \phi \rightarrow \psi \quad \Gamma \vdash \phi}{\Gamma \vdash \psi} \rightarrow -\text{ELIM}$$

$$\frac{\Gamma \vdash \phi \quad \Gamma \vdash \psi}{\Gamma \vdash \phi \land \psi} \wedge -\text{INTRO} \qquad \frac{\Gamma \vdash \phi \land \psi}{\Gamma \vdash \phi} \wedge -\text{ELIM1} \qquad \frac{\Gamma \vdash \phi \land \psi}{\Gamma \vdash \psi} \wedge -\text{ELIM2}$$

$$\frac{\Gamma \vdash \phi}{\Gamma \vdash \phi \lor \psi} \vee -\text{INTRO1} \qquad \frac{\Gamma \vdash \psi}{\Gamma \vdash \phi \lor \psi} \vee -\text{INTRO2}$$

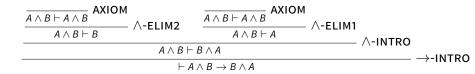
$$\frac{\Gamma \vdash \phi \lor \psi \quad \Gamma \vdash \phi \rightarrow \chi \quad \Gamma \vdash \psi \rightarrow \chi}{\Gamma \vdash \chi} \vee -\text{ELIM}$$

$$\frac{\Gamma, P \vdash \phi}{\Gamma \vdash \forall P. \phi} \forall -\text{INTRO} \qquad \frac{\Gamma \vdash \forall P. \phi}{\Gamma \vdash \phi \{\psi/P\}} \forall -\text{ELIM}$$

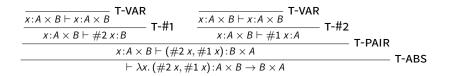
Let's try a proof! Here's a proof that $A \land B \rightarrow B \land A$ is a theorem.



Let's try a proof! Here's a proof that $A \land B \rightarrow B \land A$ is a theorem.



Doesn't this look a little... familiar?



Every natural deduction proof tree has a corresponding type tree in System F with product and sum types! And vice-versa!

	Type Systems	Formal Logic
τ	Туре	ϕ Formula
τ	is inhabited	ϕ is a theorem
е	Well-typed expression	π Proof

A program with a given type acts as a *witness* that the type's corresponding formula is true.

Every type rule in System F with product and sum types corresponds 1-1 with a proof rule in natural deduction:

Type Systems		Formal Logic	
\rightarrow	Function	\rightarrow	Implication
\times	Product	\wedge	Conjunction
+	Sum	\vee	Disjunction
\forall	Universal	\forall	Quantifier

You can even add existential types to correspond to existential quantification. It still works!

Every type rule in System F with product and sum types corresponds 1-1 with a proof rule in natural deduction:

Type Systems		Formal Logic	
\rightarrow	Function	\rightarrow	Implication
\times	Product	\wedge	Conjunction
+	Sum	\vee	Disjunction
\forall	Universal	\forall	Quantifier

You can even add existential types to correspond to existential quantification. It still works!

Is this a coincidence? Natural deduction was invented by a German logician in 1935. Types for the λ -calculus were invented by Church at Princeton in 1940.

Propositions as Types Through the Ages

Natural Deduction

Gentzen (1935)

Type Schemes Hindley (1969)

System F Girard (1972)

Modal Logic Lewis (1910)

Classical-Intuitionistic Embedding Gödel (1933)

- $\Leftrightarrow \quad \textbf{Typed } \lambda \textbf{-Calculus} \\ \text{Church (1940)} \\$
- ↔ ML's Type System Milner (1975)
- $\Leftrightarrow \quad \begin{array}{l} \textbf{Polymorphic } \lambda \textbf{-Calculus} \\ \text{Reynolds (1974)} \end{array}$

↔ Monads Kleisli (1965), Moggi (1987)

⇔ Continuation Passing Style Reynolds (1972)

Term Assignment

This all means that we have a new way of proving theorems: writing programs!

This all means that we have a new way of proving theorems: writing programs!

- To prove a formula ϕ :
- 1. Convert the ϕ into its corresponding type τ .
- 2. Find some program v that has the type τ .
- 3. Realize that the existence of v implies a type tree for $\vdash v:\tau$, which implies a proof tree for $\vdash \phi$.

Negation and Continuations

Let's explore one extension. We'd like to use this rule from classical logic:

 $\frac{\Gamma \vdash \phi}{\Gamma \vdash \neg \neg \phi}$

but there's no obvious correspondence in System F.

Negation and Continuations

Let's explore one extension. We'd like to use this rule from classical logic:

 $\frac{\Gamma \vdash \phi}{\Gamma \vdash \neg \neg \phi}$

but there's no obvious correspondence in System F.

Recall that $\neg \phi$ is shorthand for $\phi \rightarrow \bot$. So $\neg \neg \phi$ corresponds to the System F function type $(\tau \rightarrow \bot) \rightarrow \bot$.

So what we need is a way to take any program of any type τ and turn it into a program of type $(\tau \rightarrow \bot) \rightarrow \bot$.

Negation and Continuations

Let's explore one extension. We'd like to use this rule from classical logic:

 $\frac{\Gamma \vdash \phi}{\Gamma \vdash \neg \neg \phi}$

but there's no obvious correspondence in System F.

Recall that $\neg \phi$ is shorthand for $\phi \rightarrow \bot$. So $\neg \neg \phi$ corresponds to the System F function type $(\tau \rightarrow \bot) \rightarrow \bot$.

So what we need is a way to take any program of any type τ and turn it into a program of type $(\tau \rightarrow \bot) \rightarrow \bot$.

Shockingly, that's exactly what the CPS transform does! A τ becomes a function that takes a continuation $\tau \to \bot$ and invokes it, producing \bot .