Dependent Types for Natural Language Semantics

Marek Zawadowski

University of Warsaw

NASSLLI
CMU, Pittsburgh
June 25, 2018

Propositional logic - syntax

1. Language
   1. propositional variables $p, q, r, \ldots$
   2. propositional constants $\bot, \top$
   3. connectives $\wedge, \vee, \rightarrow$
2. Propositional formulas (recursive definition):
   least set $\mathcal{F}$ containing variables, constants, $\phi \wedge \psi$, $\phi \vee \psi$, $\phi \rightarrow \psi$ whenever it contains ϕ and ψ.

Propositional logic - meaning of the formulas

1. Truth conditions (leading to the classical logic)
    1. Satisfaction: when a propositional formula is true, given an assignment (i.e. truth values of the variables occurring in it)?
    2. Tautology = formula is true, under any assignment.
2. Provability (leading to the intuitionistic logic)
    1. what will count as a proof/justification of a formula, given proofs/justifications of the variables occurring in it?
    2. Tautology = a formula that has a proof no matter what are the (sets of) proofs of the variables occurring in it.

Notation

$$a : A$$

may be interpreted as

1. a is an element of a collection/set A;
2. a is a term of a type A;
3. a is a proof of a formula/proposition A.

Propositions as types.

Propositional logic - meaning

Provability/justification/evidence:

1. a proof of $\phi \wedge \psi$ consists of a pair $\langle x, y \rangle$ where x is a proof of ϕ and y is a proof of ψ;
2. a proof of $\phi \vee \psi$ consists of a pair $\langle a, y \rangle$ where a is a choice of a formula ϕ or ψ and y is a proof of that formula;
3. a proof of $\phi \to \psi$ is a function/construction that transforms proofs of ϕ to proofs of ψ;
4. there is no proof of $\bot$;
5. $\top$ has a proof.

Note that the formula $p \vee (p \to \bot)$ is NOT a tautology in the above sense.

Individual elements - first order logic

We extend the language L adding

1. individual variables (Var) and individual constants ($Const$) to talk about individual elements;
2. predicates that can express properties of individual elements $Pred$;
3. function symbols Fun to denote operations on individual elements;
4. quantifiers: $\exists_x$, $\forall_x$;
5. generalized quantifiers: $\exists_x^\omega$, **most**, **few**, **many**.

And we repeat a recursive definition of a first order formula over the language L.

First order logic - model-theoretic approach

Before we can define the notion of satisfaction, we need to define the notion of an L-structure to interpret the symbols of language L:

1. U - a universe (a set);
2. an element of U for every constant of L;
3. a relation on U for every predicate of L;
4. a function for every function symbol of L.

Now we can define in the usual (here called model-theoretic) way

1. the notion of satisfaction;
2. the notion of tautology.

First order logic - many types?

- The idea of having just one universe in first order models originated from G. Frege and is widely adopted in mathematics as it fits well the mathematical/logical practice.
- But there is no problem to have more than just one type of elements. This is common practice in programming languages. Different kinds/types/sorts of elements are stored in different types for economy reasons.
- When we decide to have many types, we need to take care of the types of variables X, Y Z. Thus we need to consider contexts to keep track of them

$$x, x' : X, y : Y, z, z' : Z$$

- ... and consider formulas/expressions only in contexts, to keep track of the typing of variables:

$$x, x' : X, y : Y, z, z' : Z \vdash P(x, y, z')$$

- How do individual elements come into a proof-theoretic interpretation? What would play a role of an L-structure?
- A universe U? OK, but we can have here also many types $X, Y, Z \ldots$, as well.
- A predicate, say P on X, should provide for each x in X a collection $P(x)$ of proofs that the property P holds of x. We can think of it as 'a family of collections' $\{P(x)\}_{x \in X}$ or more concisely as a map

$$\begin{array}{c} P \\ \downarrow{\scriptstyle \pi} \\ X \end{array}$$

so that $P(x)$ is a fiber $\pi^{-1}(x)$ of map π over element x.

First order logic - dependent types

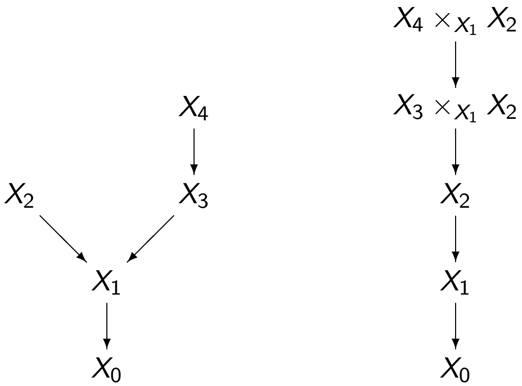
... and we can iterate this dependence relation

$$\begin{array}{c} X_n \\ \downarrow \\ X_{n-1} \\ \vdots \\ X_2 \\ \downarrow \\ X_1 \\ \downarrow \\ X_0 \end{array}$$

We think of X_{i+1} as a family of (dependent) types (fibers) indexed by the 'elements' of type X_i.

First order logic: more on dependent types

... dependence does not need to be a linear relation (left diagram)



but it can be linearized (right diagram) if we want to...

First order logic - notation for contexts with dependent types

Notation for contexts with dependent types

In the notation

$$x_0 : X_0, x_1 : X_1(x_0), x_2 : X_2(x_1, x_0)$$

all the variables types depend on are explicitly indicated.

First order logic - proof-theoretic semantics

The statement

$$x_0 : X_0, x_1 : X_1(x_0), x_2 : X_2(x_1, x_0) \vdash P(x_0, x_1, x_2) : \mathit{type}$$

should be read that P is a type depending on variables x_0, x_1, x_2.

First order logic - proof-theoretic interpretation of quantifiers

Existential quantification
If we have already built a type in context

$$x_0 : X_0, x_1 : X_1(x_0) \vdash P(x_0, x_1) : \mathit{type}$$

then we can form a type

$$x_0 : X_0 \vdash \Sigma_{x_1 : X_1(x_0)} P(x_0, x_1) : \mathit{type}$$

It should be interpreted as the collection of proofs such that in the fiber over x_0 in X_0 we have proofs why $\exists_{x_1 : X_1(x_0)} P(x_0, x_1)$.

First order logic - proof-theoretic interpretation of quantifiers

Universal quantification Similarly, having

$$x_0 : X_0, x_1 : X_1(x_0) \vdash P(x_0, x_1) : type$$

we can form a type

$$x_0 : X_0 \vdash \Pi_{x_1 : X_1(x_0)} P(x_0, x_1) : type$$

It should be interpreted as the collection of proofs such that in the fiber over x_0 in X_0 we have proofs why $\forall_{x_1 : X_1(x_0)} P(x_0, x_1)$.
NB. This is more a quantification of proofs than individual elements. But for both universal and existential quantifiers they agree, in a sense, with the intuitive meaning of quantifiers.

First order logic - proof-theoretic interpretation of quantifiers - iteration

Iterated quantification

We can have many quantifiers in a formula but we need to respect the dependencies.

If with have a type in context

$$x_0 : X_0, x_1 : X_1(x_0) \vdash P(x_0, x_1) : \mathit{type}$$

we can form types

$$x_0 : X_0 \vdash \Pi_{x_1 : X_1(x_0)} P(x_0, x_1) : \mathit{type}$$

$$\vdash \Sigma_{x_0 : X_0} \Pi_{x_1 : X_1(x_0)} P(x_0, x_1) : \mathit{type}$$

but we can't form a formula

$$\vdash \Sigma_{x_1 : X_1(x_0)} \Pi_{x_0 : X_0} P(x_0, x_1) : \mathit{type} \quad \mathit{WRONG}!$$

First order logic - proof-theoretic meaning of a formula

Proof-theoretic meaning
A sentence is a tautology iff it has a proof iff the corresponding type is inhabited.

If we interpret types as sets and dependent types as functions, say $\pi : B \to A$, then

- type $\Sigma_{a:A}B(a)$ can be interpreted as the sum of the fibers of the function π i.e. the domain of π

$$\Sigma_{a:A}B(a) = \coprod_{a\in A} B(a) = B;$$

- type $\Pi_{a:A}B(a)$ can be interpreted as the set of functions $s : A \to B$ such that $\pi \circ s = id_A$ i.e. the product of fibers of π

$$\Pi_{a:A}B(a) = \prod_{a\in A} B(a).$$

First order logic: model-theoretic interpretation with dependent types

Can we combine dependent types with model-theoretic interpretation? Why not?
Do we want that? Yes: there are dependent type constructions that natural languages make use of (e.g. anaphora).
Thus we can consider then formulas of first order logic in the context with dependent types like

$$x_0 : X_0, x_1 : X_1(x_0) \vdash P(x_0, x_1) : \mathit{formula}$$

$$x_0 : X_0 \vdash \forall_{x_1 : X_1(x_0)}(P(x_0, x_1) \wedge Q(x_0, x_1)) : \mathit{formula}$$

but we can't form a formula

$$\vdash \exists_{x_1 : X_1(x_0)} \forall_{x_0 : X_0} P(x_0, x_1) : \mathit{formula} \quad \mathit{WRONG}!$$

First order logic: model-theoretic interpretation with dependent types

What do we get:

1. predicates on dependent types;
2. quantifications along fibers;
3. generalized quantifications as easy as $\exists$ and $\forall$.

Dependent types - tracing the identity

Types (applications and identity) vs predicates (applications).

Thank You for Your Attention!