Extending First Order Logic to lists and recursion

COMP 527 Logic and Computations Final Project

Anthony Beaudry (261056660)
Claire Yang (260898597)
Tommy Zhou (260913606)

April 24, 2025

1 Motivation and Revisiting First Order Logic

In class, we introduced First Order Logic (FOL) as an extension of Propositional Logic (PL)[1].

To start, we defined terms, propositions, and types as follows:

$$\begin{array}{ll}
\text{Terms} & t := x \mid f(t_1, \ldots, t_n) \\
\text{Types} & \tau \\
\text{Propositions} & A := \top \mid \bot \mid A_1 \wedge A_2 \mid A_1 \vee A_2 \mid A_1 \supset A_2 \mid \forall x : \tau . A(x) \mid \exists x : \tau . A(x)
\end{array}$$

We extended the grammar of logical formulas with universal quantification, written as $\forall x : \tau . A(x)$, and existential quantification $\exists x : \tau . A(x)$. In class, we mainly focused on the introduction and elimination rules for these quantifiers. Below are the inference rules:

$$\dfrac{\begin{array}{c}\overline{a : \tau} \\ \vdots \\ A(a)\ \mathsf{true}\end{array}}{\forall x : \tau . A(x)\ \mathsf{true}}\ \forall I^a \qquad \dfrac{\forall x : \tau . A(x)\ \mathsf{true} \quad t : \tau}{A(t)\ \mathsf{true}}\ \forall E$$

$$\dfrac{A(t)\ \mathsf{true} \quad t : \tau}{\exists x : \tau . A(x)\ \mathsf{true}}\ \exists I \qquad \dfrac{\exists x : \tau . A(x)\ \mathsf{true} \qquad \begin{array}{c}\overline{a : \tau} \quad \overline{A(a)\ \mathsf{true}}\,u \\ \vdots \\ C\ \mathsf{true}\end{array}}{C\ \mathsf{true}}\ \exists E^{a,u}$$

We also modified the proof system's context to include types:

$$\Gamma := \cdot \mid \Gamma, u : A\ \mathsf{true} \mid \Gamma, a : \tau$$

With these definitions, we can now prove some interesting statements in first-order logic, such as:

$$\exists x \in \tau.(P(x) \supset A) \supset (\forall x \in \tau.P(x)) \supset A \ \mathsf{true}$$

We won't show the full proof derivation here, but this serves as a review of the FOL system we studied in class.

2 Extending FOL with Lists

2.1 Natural Numbers

In the previous section, we worked with FOL while keeping $t : \tau$ abstract. To support lists in FOL, we need to define new terms that belong to a specific type. We'll construct lists step by step, starting with simpler terms such as natural numbers.

We can define natural numbers in the FOL system as follows[1]:

$$\frac{}{0 : \ \mathsf{nat}}\,N_0 \qquad \frac{t : \ \mathsf{nat}}{\mathsf{suc}\ t : \ \mathsf{nat}}\,N_{suc}$$

We observe that natural numbers form an inductive type, meaning every natural number is either 0 or of the form $\mathsf{suc}\ n$ for some natural number n. By studying natural numbers, we can gain a better understanding of both lists and recursion.

In the following sections, we will define the list type using similar principles. Initially, we will keep the definition of lists orthogonal to that of natural numbers—that is, we won't rely on natural numbers to construct or operate on lists. However, we will later compare the two systems and show how they relate. In fact, we will be able to prove that natural numbers can be embedded into the list system, and that they can also be defined using lists.

2.2 Defining Lists

At this stage, our definition of lists is purely syntactic: we introduce a new type constructor and primitive constants that allow us to build list values. We are not yet introducing any principles of recursion or induction—those will come later when we begin reasoning about lists.

$$\text{Types} := \tau \mid \ \mathsf{list}\ \tau$$

Let us introduce a new type constructor $\mathsf{list}\ \tau$ for any type τ. Elements of type $\mathsf{list}\ \tau$ are finite sequences of elements of type τ. All other formation rules remain unchanged. We define lists using two constructors, which we treat as introduction rules:

1. nil : the empty list
2. $\mathsf{cons}\ (h, t)$: constructs a new list by prepending an element h of type τ to a list t of type $\mathsf{list}\ \tau$

$$\frac{}{\textsf{nil} : \textsf{list}\ \tau}\ \textsf{list}\ I_{nil} \qquad \frac{h : \tau \quad t : \textsf{list}\ \tau}{\textsf{cons}\ (h, t) : \textsf{list}\ \tau}\ \textsf{list}\ I_{cons}$$

We can observe the structural similarity between these rules and the rules for natural numbers, we will show their relation later.

- nil corresponds to 0
- cons corresponds to suc

2.3 Reasoning About Lists

One of the main goals after introducing these new definitions is to prove a property $A(l)$ true where l has type list τ. To achieve this, we use the rule of generalization, which allows us to prove a property for all lists of type list τ by showing that it holds for both the empty list and the cons case. REF REQUIRED

1. Base case: $A(\textsf{nil})\ \textsf{true}$

   We establish that the property holds for the empty list.

2. **Step case:** For any list t of type list τ, we assume $A(t)$ true and show that $A(\textsf{cons}\ (h, t))$ true for any element h of type τ.

   We can represent this reasoning using the elimination rule we are familiar with:

$$\frac{l : \textsf{list}\ \tau \quad A(\textsf{nil})\ \textsf{true} \quad \begin{array}{c} \dfrac{}{t : \textsf{list}\ \tau}\ t \quad \dfrac{}{h : \tau}\ h \quad \dfrac{}{A(t)\ \textsf{true}}\ ih \\ \vdots \\ A(\textsf{cons}\ (h, t))\ \textsf{true} \end{array}}{A(t)\ \textsf{true}}\ \textsf{list}\ E^{h,t,ih}$$

2.4 Rules with Explicit Context

If we keep the context of the proof system the same as FOL, we can rewrite the inference rules for lists as follows:

$$\frac{}{\textsf{nil} : \textsf{list}\ \tau}\ \textsf{list}\ I_{nil} \qquad \frac{\Gamma \vdash h : \tau \quad \Gamma \vdash t : \textsf{list}\ \tau}{\Gamma \vdash \textsf{cons}\ (h, t) : \textsf{list}\ \tau}\ \textsf{list}\ Icons$$

$$\frac{\Gamma \vdash l : \textsf{list}\ \tau \quad \Gamma \vdash A(\textsf{nil})\ \textsf{true} \quad \Gamma, t : \textsf{list}\ \tau, h : \tau, A(t)\ \textsf{true} \vdash A(\textsf{cons}\ (h, t))\ \textsf{true}}{A(t)\ \textsf{true}}\ \textsf{list}\ E^{h,t,ih}$$

3 Extending FOL with Recursion

When extending the FOL system with lists, a natural next step is to define recursion. Lists themselves are inherently recursive: a list is either empty, or built by adding an element to the front of another list. This recursive structure is not only reflected in the way we define lists syntactically, but also in how we reason about them through the elimination rule. The elimination rule for lists mirrors the logic of a recursive function—it breaks a list into its base case (the empty list) and its inductive case (a head and an existing list), allowing us to define operations by handling each case separately. This connection between the structure of lists and recursive reasoning forms the basis for introducing recursion into our extended FOL system.

3.1 Relation to Gödel's System T

When discussing recursion in the context of natural numbers, an important existing reference is Gödel's System T[2].

Gödel's System T provides a general mechanism called primitive recursion, which serves as a foundation for defining functions over inductively defined types. Primitive recursion captures the essential inductive nature of natural numbers, and can be viewed as an implicit termination proof for every program expressed in the language[3].

Our list type is intentionally designed to mirror the structure of natural numbers in System T. Both are inductively defined: natural numbers begin with 0 and use suc to build larger values, while lists begin with nil and use cons to construct longer sequences. Because of this parallel, we can adapt the recursion principles from System T to work with lists. Specifically, we define a list recursor that handles two cases—one for the empty list (nil), and one for a list with a head and tail (cons). This approach allows us to define recursive functions over lists in a way that is both terminating and logically sound. By building on the well-established foundation of System T, we ensure our extended system remains consistent while gaining additional expressive power.

3.2 Defining Recursion

Now that we have all the necessary tools, we can define recursion. In this project, we will focus on list-based recursion. To do this, we introduce a new term constructor rec, which represents a recursive function. We now extend our previous definition of terms as follows:

$$\text{Terms} \qquad t := x \mid f(t_1, \ldots, t_n) \mid \mathsf{rec}\,(l, M_{\mathsf{nil}}, h, t, M_{\mathsf{cons}})$$

The term $\mathsf{rec}\,(l, M_{\mathsf{nil}}, h, t, M_{\mathsf{cons}})$ is a recursive function that have 5 parameters:

- l: the list we are going to apply the recursion on with type $\mathsf{list}\ \tau$
- M_{nil} : the function we are going to apply when the list is empty with term $f(\mathsf{nil}) \to M_{\mathsf{nil}}$
- h: the new element added to the list using $\mathsf{cons}\,(h, t)$ in the inductive case, with type τ
- t: the list t merged with the new element using $\mathsf{cons}\,(h, t)$ in the inductive case, with type $\mathsf{list}\ \tau$
- M_{cons} : the inductive function we are going to apply when the list is not empty, with term $f(\mathsf{cons}\,(h, t)) \to M_{\mathsf{cons}}$

If we view the definition of recursion as a function, we can derive the following structure:

$$\begin{array}{l} \textsf{rec}\ l\ \textsf{with} \\ \quad \mid f(\ \textsf{nil}\ ) \to M_{\textsf{nil}} \\ \quad \mid f(\ \textsf{cons}\ (h,t)) \to M_{\textsf{cons}} \end{array}$$

4 Exploring the System

In this section, we will show the interesting and important properties of our newly constructed system. In general, we will show that the system is terminating, sound, and complete. We will also show what we can prove with the system, and how we can embed natural numbers into lists.

4.1 Equality Relation on List

$$\frac{}{\textsf{nil}\ =\ \textsf{nil}\ \ \textsf{true}} = I_{nil} \qquad \frac{h:\tau \quad \Gamma,\ t1:\ \textsf{list}\ \tau,\ t2:\ \textsf{list}\ \tau \vdash t1 = t2\ \textsf{true}}{\textsf{cons}\ (h,t1) =\ \textsf{cons}\ (h,t2)\ \textsf{true}} = I_{cons}$$

$$\frac{\Gamma, h:\tau, t:\ \textsf{list}\ \tau \vdash\ \textsf{cons}\ (h,t) =\ \textsf{nil}\ \ \textsf{true}}{C\ \textsf{true}} = E_{cons,nil} \qquad \frac{\Gamma, h:\tau, t:\ \textsf{list}\ \tau \vdash\ \textsf{nil}\ =\ \textsf{cons}\ (h,t)\ \textsf{true}}{C\ \textsf{true}} = E_{nil,cons}$$

$$\frac{\Gamma, h:\tau,\ t1:\ \textsf{list}\ \tau,\ t2:\ \textsf{list}\ \tau \vdash\ \textsf{cons}\ (h,t1) =\ \textsf{cons}\ (h,t2)\ \textsf{true}}{t1 = t2\ \textsf{true}} = E_{cons,cons}$$

4.2 Concatenation Function for List

As a function for lists, one natural use is to concatenate lists together which can be used to link many constructs.

$$append(t1 : list\ \tau,\ t2 : list\ \tau) =\ \textsf{rec}\ (t1, f(nil) = t2, h, t, f(cons(h,t)) = cons(h, f(t))) : list\ \tau$$

Theorem. $append(t,\ \textsf{nil}\ ) = t \ \ \forall t \in\ \textsf{list}\ \tau$

Proof Intuition: We use structural induction on the list t. In the base case (t = nil), this can be easily shown to be true by Introduction nil rule. Then, we apply induction hypothesis to t' in the step case when t = cons (h, t').

4.3 Termination for Recursion

A critical property of our recursion construct is that all well-typed recursive functions terminate. This is because:

- Each recursive call is made on a structurally smaller term (the tail of the list).
- Lists are finite and have a well-founded structure.

This ensures that the recursion will eventually reach the base case (the empty list) and terminate.

4.4 Local Soundness

Since like all inference rules we have seen before, the rules for lists allow us to introduce and extract information. It it important to check if the rules we have defined are sound and complete. We first show the local soundness:

- Base Case: We established $A(\ \mathsf{nil}\ )\ \mathsf{true}$, it is easy to show that there is a detour for the conclusion if we first introduce then eliminate the empty list.

$$\dfrac{\dfrac{}{\mathsf{nil}\ :\ \mathsf{list}\ \tau}\ \mathsf{list}\ I_{nil} \qquad \begin{array}{c}\mathcal{E}\\ A(\ \mathsf{nil}\ )\ \mathsf{true}\end{array} \qquad \begin{array}{c}\overline{t:\ \mathsf{list}\ \tau}^{\,t}\ \ \overline{h:\tau}^{\,h}\ \ \overline{A(t)\ \mathsf{true}}^{\,ih}\\ \mathcal{F}\\ A(\ \mathsf{cons}\ (h,t))\ \mathsf{true}\end{array}}{A(\ \mathsf{nil}\ )\ \mathsf{true}}\ \mathsf{list}\ E^{h,t,ih} \Rightarrow \qquad \begin{array}{c}\mathcal{E}\\ A(\ \mathsf{nil}\ )\ \mathsf{true}\end{array}$$

- Step Case: We should show the cases for list $\mathsf{cons}\ (h', t')$, the idea is slightly different from the inference rules we have seen before. This time we are unable to directly eliminate the list, instead, the soundness reduction is also completed recursively. We can substitute the list with a new list t' and a new element h', while t' have less element, this will bring us back to the base case since the list is finite. The concept of soundness in step case is entirely different from previous soundness proofs, the idea is extracted from the soundness of natural number in Frank Pfenning's lecture notes[4].

$$\dfrac{\dfrac{\begin{array}{c}\mathcal{D}_1\\ h':\tau\end{array} \qquad \begin{array}{c}\mathcal{D}_2\\ t':\ \mathsf{list}\ \tau\end{array}}{\mathsf{cons}\ (h',t'):\ \mathsf{list}\ \tau}\ \mathsf{list}\ I_{cons} \qquad \begin{array}{c}\mathcal{E}\\ A(\ \mathsf{nil}\ )\ \mathsf{true}\end{array} \qquad \begin{array}{c}\overline{t:\ \mathsf{list}\ \tau}^{\,t}\ \ \overline{h:\tau}^{\,h}\ \ \overline{A(t)\ \mathsf{true}}^{\,ih}\\ \mathcal{F}\\ A(\ \mathsf{cons}\ (h,t))\ \mathsf{true}\end{array}}{A(\ \mathsf{cons}\ (h',t'))\ \mathsf{true}}\ \mathsf{list}\ E^{h,t,ih}$$

$$\Rightarrow \qquad \begin{array}{c}[h'/h]\ [t'/t]\ \mathcal{F}\\ A(\ \mathsf{cons}\ (h',t'))\ \mathsf{true}\end{array}$$

4.5 Local Completeness for List

Now we have defined recursion, we can show the completeness of the list system, an important step to show the completeness of the system is to consider the information derived from the elimination rule $A(\ \mathsf{cons}\ (h,t)\ \mathsf{true})$ as an recursion function[4]. With this idea, we build the completeness proof from $l:\ \mathsf{list}\ \tau$:

$$\begin{array}{c}\mathcal{D}\\ l:\ \mathsf{list}\ \tau\end{array} \quad \Rightarrow$$

$$\dfrac{\begin{array}{c}\mathcal{D}\\ l : \mathsf{list}\ \tau\end{array} \quad \dfrac{}{\mathsf{nil}\ :\ \mathsf{list}\ \tau}\,\mathsf{list}\ I_{nil} \quad \dfrac{\dfrac{}{h : \tau}\,h \quad \dfrac{}{t :\ \mathsf{list}\ \tau}\,t}{\mathsf{cons}\ (h, t) :\ \mathsf{list}\ \tau}\,\mathsf{list}\ I_{cons}}{A :\ \mathsf{rec}\ (l, M_{\mathsf{nil}}, h, t, M_{\mathsf{cons}})}\,\mathsf{list}\ E^{h,t,ih}$$

4.6 Embedding Natural Numbers into Lists

Intuitively, we can view natural numbers as a special case of lists. We can represent a natural number n as a unary - a list of n identical elements. For example, the number 3 can be represented as the list $\mathsf{cons}\ (h,\ \mathsf{cons}\ (h,\ \mathsf{cons}\ (h,\ \mathsf{nil}\ )))$. This representation allows us to use the same recursive principles we defined for lists to reason about natural numbers.

We can apply what we have learned in class to define a transformation more formally. Consider the function $\mathcal{C}_h$, where the transformation function is C, and we choose $h : \tau$ be the single unique element in the list. The function $\mathcal{C}_h$ is defined as follows:

$$\begin{array}{rr} \text{Base Case:} & \mathcal{C}(0) = \mathsf{nil} \\ \text{Inductive Case:} & \mathcal{C}(\ \mathsf{suc}\ n) = \mathsf{cons}\ (h, \mathcal{C}(n)) \end{array}$$

The structure is preserved with base case where $0 :$ `nat` maps to nil, and in the inductive case, the successor operation corresponds to consing an element to the front of a list. And for all induction functions reasoning with natural number n and its euqivlant list $\mathcal{C}_h(n)$, the operation on `suc` n is equivalent to the operation on $\mathsf{cons}\ (h, \mathcal{C}_h(n))$, where the property of h is ignored. We will not cover the complete proof like what we did with embedding $\mathcal{IL}$ here.

References

[1] B. Pientka, “First-order logic - an extended discussion,” in *Logic and Computation.*

[2] V. K. Gödel, “Über eine bisher noch nicht benützte erweiterung des finiten standpunktes,” *Dialectica*, vol. 12, no. 3-4, pp. 280–287, 1958. DOI: `https://doi.org/10.1111/j.1746-8361.1958.tb01464.x`. eprint: `https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1746-8361.1958.tb01464.x`. [Online]. Available: `https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1746-8361.1958.tb01464.x`.

[3] J.-Y. Girard, “Une extension de ˇlinterpretation de gödel a ˇlanalyse, et son application a ˇlelimination des coupures dans ˇlanalyse et la theorie des types,” in *Proceedings of the Second Scandinavian Logic Symposium*, ser. Studies in Logic and the Foundations of Mathematics, J. Fenstad, Ed., vol. 63, Elsevier, 1971, pp. 63–92. DOI: `https://doi.org/10.1016/S0049-237X(08)70843-7`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S0049237X08708437`.

[4] F. Pfenning, “Lecture notes on natural numbers,” in *Constructive Logic.* [Online]. Available: `https://www.cs.cmu.edu/~fp/courses/15317-s23/lectures/11-nat.pdf`.