Extending FOL with Lists and Recursion

Anthony Beaudry
Claire Yang
Tommy Zhou

April 24, 2025

Introduction

- First-Order Logic (FOL) is a foundational system in logic which allows us to quantify over data and unordered sets.
- The standard FOL [1] cannot represent:
  - Data structures like `list`, `tree`
  - Recursive functions like `append`, `length`
  - Structural induction or recursive reasoning
- These are essential for reasoning about programs and computations.
- Our goal is to extend FOL with lists and recursion, inspired by System T[2].

Our Goal

- Extend FOL with:
  - Inductive list types (`nil`, `cons`) inspired by natural numbers <**empty citation**>
  - A structural recursion operator, inspired by System T
  - Natural deduction rules over lists
  - Recursion operator defined over list structure
- Implement the system and type-checked in Beluga.
  - Formalized the logic and recursion as LF types
  - Defined inductive schemas for lists
  - Proved example theorems like `append(l, nil) = l`

Defining and Reasoning About Lists

List Type in Logic

- We define an inductive type: `list`
  - `nil` : the empty list
  - `cons(h, t)` : adds head h to list t
- Mirrors natural numbers:
  - `nil` $\leftrightarrow$ 0
  - `cons` $\leftrightarrow$ `suc`

Reasoning with Lists

- Extend natural deduction with list rules
- Elimination rule for structural induction:
  - Base case: prove for `nil`
  - Inductive case: prove for `cons(h, t)` assuming it holds for t
- Enables proofs like: `append(l, nil) = l`

Defining Recursion Over Lists

Why Recursion?

- Many functions like `append`, `length` are naturally recursive.
- To reason about them, logic needs a recursion mechanism.
- We adapt the idea of primitive recursion from Gödel's System T.

List Recursor

- Recursor form:

$$\mathrm{rec}(l, M_{\mathtt{nil}}, h, t, M_{\mathtt{cons}})$$

- Matches on the list:
  - If $l = \mathtt{nil}$, return $M_{\mathtt{nil}}$
  - If $l = \mathtt{cons}(h, t)$, apply $M_{\mathtt{cons}}$ recursively
- Implemented as `list_rec` in Beluga
- Always terminates: recursion follows list structure

Induction and Equality

List Induction Schema

- Form: $\lambda l.\, P(l)$
- Prove $P(l)$ for all lists l by:
  - Base case: prove $P(\texttt{nil})$
  - Step case: assume $P(t)$ and prove $P(\texttt{cons}(h, t))$

Used to Prove Equalities Like:

- $\texttt{append}(l, \texttt{nil}) = l$
- $\texttt{length}(\texttt{append}(l_1, l_2)) = \texttt{add}(\texttt{length}(l_1), \texttt{length}(l_2))$

Example: Append Identity

- Theorem: $\forall l.\, \texttt{append}(l, \texttt{nil}) = l$
- Base case: $\texttt{append}(\texttt{nil}, \texttt{nil}) = \texttt{nil}$
- Step case: assume $\texttt{append}(t, \texttt{nil}) = t$ then $\texttt{append}(\texttt{cons}(h, t), \texttt{nil}) = \texttt{cons}(h, t)$ use eqList_cons to construct equality

System Check

- **Termination**:
  - Recursion only unfolds on strictly smaller lists.
  - All lists are finite: recursion always reaches `nil`.
- **Local Soundness**: Elimination + immediate re-introduction = detour.
- **Local Completeness**: Recursor handles `nil` and `cons(h, t)` cases and all list shapes covered.

Summary

In this project we have completed:

- Extended FOL with inductive lists + recursion.
- Defined natural deduction rules and recursor.
- Proved examples like `append`, `length`.
- All formalized and type-checked in Beluga.

This allows us to formally reason about recursive functions like `append` and `length` inside logic.

References I

[1] G. Dowek, "Gödel's system T as a precursor of modern type theory," working paper or preprint, 2006. [Online]. Available: https://inria.hal.science/hal-04046289.

[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.