A Mechanized Proof in Coq of the Type Soundness of Core L^3

Milestone 2

Yawar Raza

Review

- Type Soundness
    - “Is this type system *really* detecting all the type-errors a program can have?”
    - Counterexamples can be hard to find. Let’s prove it mathematically instead!
- Mechanization
    - “Is this hand-written proof *really* correct and free of mistakes?”
    - Errors in proves can be hard to find. Let’s have the computer check the proof!

Section 1

Representing Relations

Predicates and Relations

- Propositions can be parameterized
  - Unary propositions: predicates. Whether a value has a particular property.
    - **snowing**(c): it is snowing in city c
    - **raining**(c): it is raining in city c
  - N-ary propositions: relations. Whether a value is related in other values in a particular way.
    - **weather**(w, c): the weather is w in city c
    - **typed**(Γ, e, τ): expression e has type τ in environment Γ Actually written as $\Gamma \vdash e : \tau$

Inference Rules

- A closed set of rules that defines a predicate or relation.
  - $\forall x, l.\ \textbf{in}(x, \textsf{cons}\ x\ l)$
  - $\forall x, y, l.\ \textbf{in}(x, l) \Rightarrow \textbf{in}(x, \textsf{cons}\ y\ l)$
- Example: Prove $\textbf{in}(3, \textsf{cons}\ 5\ (\textsf{cons}\ 3\ (\textsf{cons}\ 7\ \textsf{nil})))$
  - Second rule: $\textbf{in}(3, \textsf{cons}\ 3\ (\textsf{cons}\ 7\ \textsf{nil})) \Rightarrow \textsf{in}(3, \textsf{cons}\ 5\ (\textsf{cons}\ 3\ (\textsf{cons}\ 7\ \textsf{nil})))$
  - First rule: $\textbf{in}(3, \textsf{cons}\ 3\ (\textsf{cons}\ 7\ \textsf{nil}))$
- Because the rules are *closed*, we know that:
  - $\textbf{in}(x, \textsf{nil})$ is never true, no matter what x is.
  - $\textbf{in}(3, \textsf{cons}\ 5\ l)$ can only be proven by the second rule, so we know $\textbf{in}(3, l)$.

Disjoint Union: Definition 1

- **disjoint_union**(s_1, s_2, s_u) says that s_u is the disjoint union of s_1 and s_2
- Define $s[x] :=$ true if $x \in s$, false if $x \notin s$
- Simple logical formula: **disjoint_union**$(s_1, s_2, s_u) :=$

$$\begin{aligned}\forall x.\ &(s_1[x] = \mathsf{true} \wedge s_2[x] = \mathsf{false} \wedge s_3[x] = \mathsf{true}) \vee \\ &(s_1[x] = \mathsf{false} \wedge s_2[x] = \mathsf{true} \wedge s_3[x] = \mathsf{true}) \vee \\ &(s_1[x] = \mathsf{false} \wedge s_2[x] = \mathsf{false} \wedge s_3[x] = \mathsf{false})\end{aligned}$$

Disjoint Union: Definition 2

- Define helper relation **nand**(b_1, b_2, b_3):
  - **nand**(true, false, true)
  - **nand**(false, false, true)
  - **nand**(false, false, false)
- **disjoint_union**$(s_1, s_2, s_u) := \forall x.$ **nand**$(s_1[x], s_2[x], s_u[x])$

Disjoint Union: Definition 3

- Instead define **disjoint**(s_1, s_2), similarly to last slide's definition.
- Then define **disjoint_union** with the single inference rule:
  - **disjoint**$(s_1, s_2) \Rightarrow$ **disjoint_union**$(s_1, s_2, union(s_1, s_2))$

Disjoint Union: Definition 4

- Build up the sets, rather than stating a property about set membership.
    - $\textbf{disjoint\_union}(\mathit{empty}, \mathit{empty}, \mathit{empty})$
    - $\forall x, s_1, s_2, s_u.$
      $x \notin s_u \wedge \textbf{disjoint\_union}(s_1, s_2, s_u) \Rightarrow$
      $\textbf{disjoint\_union}(\mathit{add}(x, s_1), s_2, \mathit{add}(x, s_u))$
    - $\forall x, s_1, s_2, s_u.$
      $x \notin s_u \wedge \textbf{disjoint\_union}(s_1, s_2, s_u) \Rightarrow$
      $\textbf{disjoint\_union}(s_1, \mathit{add}(x, s_2), \mathit{add}(x, s_u))$

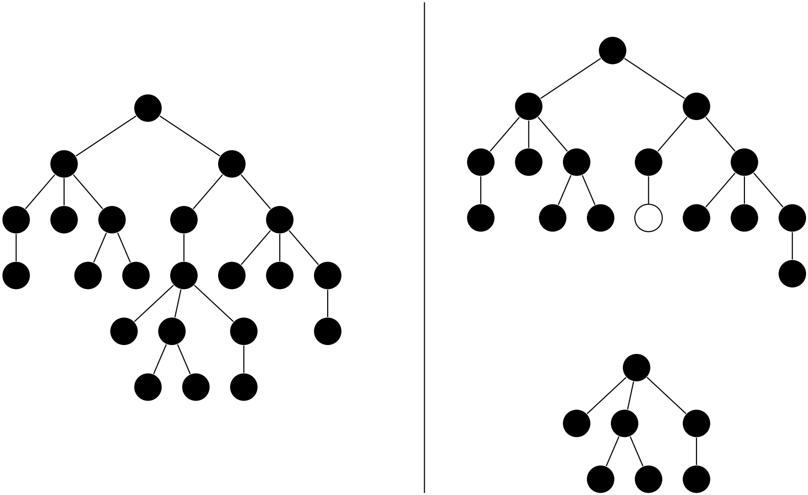
Section 2

Automation

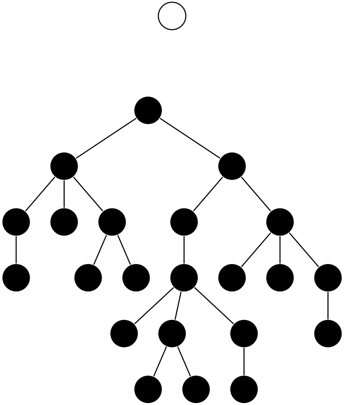
What Does Automation Do?

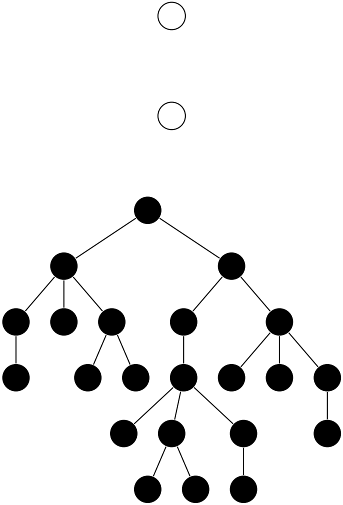
- Let’s prove $\textbf{in}(3, [5, 7, 4, 6, 3, 9])$.
  - Rule 2: $\textbf{in}(3, [7, 4, 6, 3, 9]) \Rightarrow \textbf{in}(3, [5, 7, 4, 6, 3, 9])$
  - Rule 2: $\textbf{in}(3, [4, 6, 3, 9]) \Rightarrow \textbf{in}(3, [7, 4, 6, 3, 9])$
  - Rule 2: $\textbf{in}(3, [6, 3, 9]) \Rightarrow \textbf{in}(3, [4, 6, 3, 9])$
  - Rule 2: $\textbf{in}(3, [3, 9]) \Rightarrow \textbf{in}(3, [6, 3, 9])$
  - Rule 1: $\textbf{in}(3, [3, 9])$
- Essentially, we just compared each element to the target in order.
  - We used Rule 2 if there was no match.
  - We used Rule 1 if there was a match.
- Sounds easy enough for a computer to do by itself.
- Simply put, we run a *search algorithm* to find the steps of the proof.
  - But search algorithms can infinite loop sometimes...

What I’m Searching For

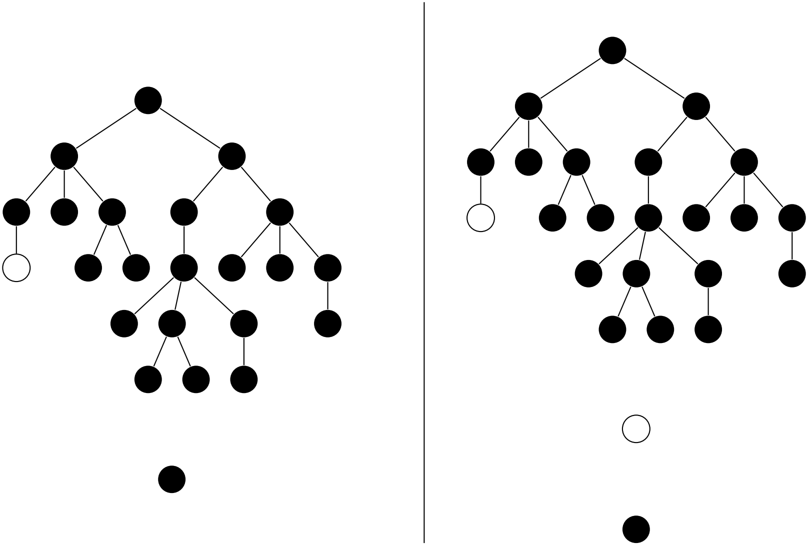


Top to Bottom

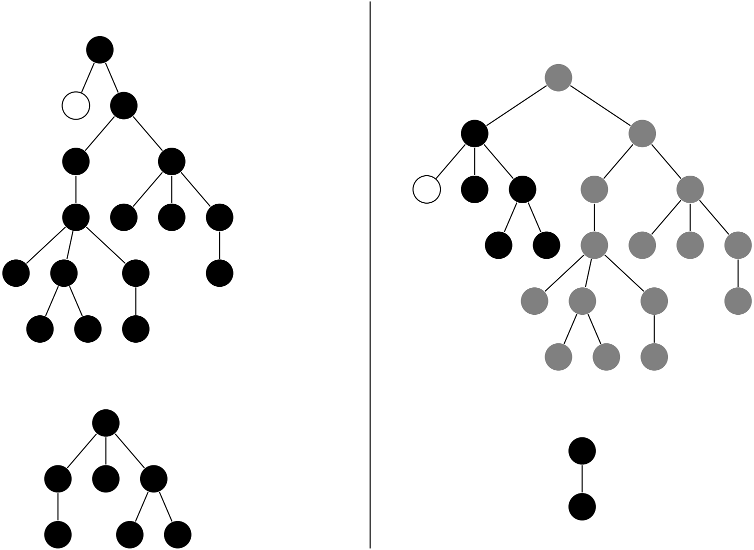




Bottom to Top



Top to Bottom, Skipping the Root



Milestones

- Complete
  - Mechanized the syntax, the operational semantics, and the type system.
- Milestone 3
  - Refactoring representations.
  - Mechanizing the semantic interpretations.
  - Working on the simple cases of the proof.
  - Figuring out what lemmas are needed for implementing the proof.