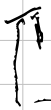


$$\begin{array}{c}
 (x, \boxed{\text{Int}}) \text{ is in } \Gamma(x, \boxed{\text{Int}}) \quad l \in \mathbb{Z} \\
 \hline
 \frac{\Gamma(x, \boxed{\text{Int}}) \vdash x :: \boxed{\text{Int}} \quad \Gamma(x, \boxed{\phantom{\text{Int}}}) \vdash l :: \boxed{\text{Int}}}{\Gamma(x, \boxed{\text{Int}}) \vdash x + l :: \boxed{\text{Int}}} \quad \begin{array}{l} \text{[T-Var]} \\ \text{[T-Num]} \end{array} \\
 \hline
 \frac{\Gamma(x, \boxed{\text{Int}}) \vdash x + l :: \boxed{\text{Int}}}{\Gamma \vdash \lambda x \rightarrow \underbrace{x + l} :: \boxed{\text{Int}} \rightarrow \boxed{\text{Int}}} \quad \begin{array}{l} \text{[T-Add]} \\ \text{[T-Lam]} \end{array}
 \end{array}$$

To do constraint-based type inference:

- whenever you need to guess a type, don't!
 - Just use a fresh type variable.
- whenever a rule imposes a constraint, try to find the right substitution for the free type variables to satisfy the constraint.



→ this step is called unification!

unification means:

may contain type variables

given two types T_1 and T_2 ,
find a substitution that makes them equal.

This substitution is called a unifier of T_1 and T_2 .
maps type variables to types.

<u>T_1</u>	<u>T_2</u>	<u>unifier</u>
"a"	Int	[("a", Int)]
"a" → "a"	Int → Int	[("a", Int)]
"a" → Int	Int → "b"	[("a", Int), ("b", Int)] (or [("a", Int), ("b", Int), ("c", Bool)])
"a"	"a"	[]
Int	Int	[]
Int	Int → Int	can't unify.
"a"	"a" → "a"	can't unify.
Int	"a" → "a"	can't unify.
"a" → Int → Int	"b" → "c"	[("a", "b") ("c", Int → Int)] or [("b", "a") ("c", Int → Int)]