Judgments, Inference Rules, and Inductive Definitions

Stephen A. Edwards

Columbia University

Spring 2023
There are various symbols

Symbols may be identical, even when drawn slightly differently

Other symbols are distinct

Symbols arranged in a horizontal sequence are "words, " "strings, " or "expressions"

Symbols may represent values, operations, or relationships

Some symbols are treated as variables that represent other symbols

The meaning of an expression with variables depends on the variables' values
There are various symbols.
Symbols may be identical, even when drawn slightly differently.

Symbols arranged in a horizontal sequence are "words," "strings," or "expressions."
Symbols may represent values, operations, or relationships.
Some symbols are treated as variables that represent other symbols.
The meaning of an expression with variables depends on the variables' values.
There are various symbols
Symbols may be identical, even when drawn slightly differently
Other symbols are distinct
There are various symbols
Symbols may be identical, even when drawn slightly differently
Other symbols are distinct
Symbols arranged in a horizontal sequence are “words,” “strings,” or “expressions”
There are various symbols
Symbols may be identical, even when drawn slightly differently
Other symbols are distinct
Symbols arranged in a horizontal sequence are “words,” “strings,” or “expressions”
Symbols may represent values, operations, or relationships
There are various symbols
Symbols may be identical, even when drawn slightly differently
Other symbols are distinct
Symbols arranged in a horizontal sequence are “words,” “strings,” or “expressions”
Symbols may represent values, operations, or relationships
Some symbols are treated as variables that represent other symbols
There are various symbols
Symbols may be identical, even when drawn slightly differently
Other symbols are distinct
Symbols arranged in a horizontal sequence are “words,” “strings,” or “expressions”
Symbols may represent values, operations, or relationships
Some symbols are treated as variables that represent other symbols
The meaning of an expression with variables depends on the variables’ values
A judgment is an assertion about one or more things, typically membership in a set.

- $0 \in \mathbb{N}$: 0 is a member of the set of natural numbers
- $n \text{ nat}$: $n$ is a member of the set of natural numbers
- $1 + 2 \text{ expr}$: $1 + 2$ is in the set of expressions
- $\tau \text{ type}$: $\tau$ is in the set of types
- $e : \tau$: Expression $e$ has type $\tau$
- $\text{sum}(n_1, n_2, n_3)$: Adding $n_1$ and $n_2$ gives $n_3$
- $n_1 + n_2 = n_3$: Adding $n_1$ and $n_2$ gives $n_3$

Prefix; infix; and suffix syntax
Inference Rule

Premises: Judgments $\rightarrow$ $I_1 \, I_2 \, \ldots \, I_k$

Conclusion: A Judgment $\rightarrow$ $I$

"If all the premises hold, the conclusion follows"

The Natural Numbers Defined Inductively by Two Inference Rules (Peano)

Axiom $\rightarrow$ $\emptyset \in \mathbb{N}$

<table>
<thead>
<tr>
<th>Judgments:</th>
<th>$a \in \mathbb{N}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variables:</td>
<td>$a \leftarrow$ Sequences of symbols</td>
</tr>
<tr>
<td>Symbols:</td>
<td>$0 \quad \text{succ}(\ )$</td>
</tr>
</tbody>
</table>

| $\emptyset$ | 0 |
| $\text{succ}(\emptyset)$ | 1 |
| $\text{succ}(\text{succ}(\emptyset))$ | 2 |
| $\text{succ}(\text{succ}(\text{succ}(\emptyset)))$ | 3 |
| $\text{succ}(\text{succ}(\text{succ}(\text{succ}(\emptyset))))$ | 4 |
Inference Rule

Premises: Judgments → $J_1 \quad J_2 \quad \ldots \quad J_k$
Conclusion: A Judgment → $\overline{J}$

"If all the premises hold, the conclusion follows"

The Natural Numbers Defined Inductively by Two Inference Rules (Peano)

$\begin{align*}
\frac{\text{zero}}{0 \in \mathbb{N}} \\
\frac{a \in \mathbb{N}}{\text{successor}} \\
\text{succ}(a) \in \mathbb{N}
\end{align*}$

Judgments: $a \in \mathbb{N}$

Variables: $a \leftarrow \text{Sequences of symbols}$

Symbols: $0 \quad \text{succ( )}$

\[\begin{array}{ll}
0 & 0 \\
\text{succ}(0) & 1 \\
\text{succ(succ}(0)) & 2 \\
\text{succ(succ(succ}(0))) & 3 \\
\text{succ(succ(succ(succ}(0)))) & 4 \\
\end{array}\]
Inference Rule

Premises: Judgments → $J_1 \ J_2 \ \cdots \ J_k$ Rule-Name
Conclusion: A Judgment → $J$ 

“If all the premises hold, the conclusion follows”

The Natural Numbers Defined Inductively by Two Inference Rules (Peano)

$\begin{align*}
0 \in \mathbb{N} & \quad \text{zero} \\
\text{succ}(a) \in \mathbb{N} & \quad \text{successor}
\end{align*}$

Technically a scheme

Scheme: pattern with variables: replacing $a$ consistently gives a rule

$\begin{align*}
\text{succ}(\text{true}) \in \mathbb{N} \\
\text{succ}(\text{succ}(\text{true})) \in \mathbb{N}
\end{align*}$

Which are variables? Values constrained? Variable scope: a single rule

Consistent replacement only:

$\begin{align*}
\text{succ(bar)} \in \mathbb{N}
\end{align*}$
is not a rule
Inference Rule

Premises: Judgments → \( J_1 \quad J_2 \quad \ldots \quad J_k \) → Conclusion: A Judgment → \( J \)

"If all the premises hold, the conclusion follows"

The Natural Numbers Defined Inductively by Two Inference Rules (Peano)

\[
\begin{align*}
0 & \in \mathbb{N} \quad \text{zero} \\
\text{succ}(a) & \in \mathbb{N} \quad \text{successor}
\end{align*}
\]

Is \( \text{succ}(\text{succ}(\text{succ}(0))) \) a.k.a. 3 a natural number? A forward derivation

\[
\begin{align*}
0 & \in \mathbb{N} \quad \text{zero}
\end{align*}
\]
Inference Rule

Premises: Judgments $\rightarrow$ $J_1 J_2 \ldots J_k$ Rule-Name
Conclusion: A Judgment $\rightarrow$ $J$

“If all the premises hold, the conclusion follows”

The Natural Numbers Defined Inductively by Two Inference Rules (Peano)

\[ \begin{align*}
\text{zero} & : 0 \in \mathbb{N} \\
\text{successor} & : \text{If } a \in \mathbb{N} \text{ then } \text{succ}(a) \in \mathbb{N}
\end{align*} \]

Is $\text{succ}(\text{succ}(\text{succ}(0)))$ a.k.a. 3 a natural number? A forward derivation

\[ \begin{align*}
\text{zero} & : 0 \in \mathbb{N} \\
\text{successor} & : \text{If } a \in \mathbb{N} \text{ then } \text{succ}(a) \in \mathbb{N}
\end{align*} \]
Inference Rule

Premises: Judgments → $J_1 \ J_2 \ \ldots \ J_k$ → Conclusion: A Judgment $\vdash J$

“If all the premises hold, the conclusion follows”

The Natural Numbers Defined Inductively by Two Inference Rules (Peano)

<table>
<thead>
<tr>
<th>Rule</th>
<th>Premise</th>
<th>Conclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>zero</td>
<td>$\theta \in \mathbb{N}$</td>
<td>$\text{successor}$ $a \in \mathbb{N}$</td>
</tr>
<tr>
<td>successor</td>
<td>$\text{successor} \ (a) \in \mathbb{N}$</td>
<td>$\text{successor}$ $\text{successor} \ (\theta) \in \mathbb{N}$</td>
</tr>
</tbody>
</table>

Is $\text{successor} \ (\text{successor} \ (\text{successor} \ (\theta)))$ a.k.a. 3 a natural number? A forward derivation

<table>
<thead>
<tr>
<th>Rule</th>
<th>Premise</th>
<th>Conclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>zero</td>
<td>$\theta \in \mathbb{N}$</td>
<td>$\text{successor}$ $\text{successor} \ (\theta) \in \mathbb{N}$</td>
</tr>
<tr>
<td>successor</td>
<td>$\text{successor} \ (\theta) \in \mathbb{N}$</td>
<td>$\text{successor} \ (\text{successor} \ (\text{successor} \ (\theta))) \in \mathbb{N}$</td>
</tr>
</tbody>
</table>
Inference Rule

Premises: Judgments → $J_1, J_2, \ldots, J_k$ Rule-Name
Conclusion: A Judgment → $J$

“If all the premises hold, the conclusion follows”

The Natural Numbers Defined Inductively by Two Inference Rules (Peano)

0 ∈ N zero

a ∈ N successor

succ(a) ∈ N successor

Is succ(succ(succ(0))) a.k.a. 3 a natural number? A forward derivation

0 ∈ N zero

succ(0) ∈ N successor

succ(succ(0)) ∈ N successor

succ(succ(succ(0))) ∈ N successor
The Natural Numbers

\[
\begin{align*}
\text{zero} & \quad 0 \in \mathbb{N} \\
\text{successor} & \quad a \in \mathbb{N} \implies \text{succ}(a) \in \mathbb{N}
\end{align*}
\]

zeroIsN :: String -> Bool

successorIsN :: String -> Bool

String is inefficient, but let’s focus on correctness first
The Natural Numbers

\[
\begin{align*}
\text{zero} & \in \mathbb{N} \\
\text{successor} & \in \mathbb{N}
\end{align*}
\]

\[
\begin{align*}
\text{zeroIsN} :: \text{String} & \rightarrow \text{Bool} \\
\text{zeroIsN } "0" & = \text{True} \\
\text{zeroIsN } _ & = \text{False} \quad -- \text{Default case}
\end{align*}
\]

\[
\begin{align*}
\text{successorIsN} :: \text{String} & \rightarrow \text{Bool}
\end{align*}
\]
The Natural Numbers

\[
\begin{align*}
\text{zero} & \in \mathbb{N} \\
as & \in \mathbb{N} \\
\text{successor} & (a) \in \mathbb{N}
\end{align*}
\]

**import** Data.List (stripPrefix) -- stripPrefix :: String -> String -> Maybe String

**zeroIsN** :: String -> Bool

zeroIsN "0" = True

zeroIsN _ = False -- Default case

**successorIsN** :: String -> Bool  -- Construct a Reverse Derivation

successorIsN s = case stripPrefix "succ(" s of
  Just aa@(_:_) -> last aa == ')' &&
  let a = init aa in
  zeroIsN a || successorIsN a -- Try both
  _ -> False

-- Of the form succ(...)?
-- Prohibit the empty string
-- Get all but last character
The Natural Numbers

\[
\begin{align*}
\text{zero} &\in \mathbb{N} \\
\text{successor} &\in \text{succ}(a) \in \mathbb{N}
\end{align*}
\]

import Data.List (stripPrefix)

zeroIsN :: String -> Bool
zeroIsN "0" = True
zeroIsN _ = False

successorIsN :: String -> Bool
successorIsN s = case match "succ(" ")" s of
    Just a -> zeroIsN a || successorIsN a
    _ -> False

match :: String -> String -> String -> Maybe String -- Helper function
match pre suff s = do a' <- stripPrefix pre s
                   reverse <$> stripPrefix (reverse suff) (reverse a')

-- Stops at Nothing
The Natural Numbers

\[
\begin{align*}
0 & \in \mathbb{N} \\
a & \in \mathbb{N} \\
\text{succ}(a) & \in \mathbb{N}
\end{align*}
\]

import Data.List (stripPrefix)

isNat :: String -> Bool -- Merge the two rules
isNat "0" = True -- zero rule
isNat s = case match "succ( " ")" s of -- successor rule
    Just a -> isNat a -- Only one thing to check
    Nothing -> False

match :: String -> String -> String -> Maybe String
match pre suff s = do a' <- stripPrefix pre s
    reverse <$> stripPrefix (reverse suff) (reverse a')
The Natural Numbers

\[
\begin{align*}
\text{zero} & \in \mathbb{N} \\
\text{successor} & \in \mathbb{N} \\
\end{align*}
\]

data Nat = Zero | Succ Nat  -- Algebraic data type: either “Zero” or “Succ n”

zeroIsN :: Nat -> Bool
zeroIsN Zero = True
zeroIsN _ = False

successorIsN :: Nat -> Bool
successorIsN (Succ a) = zeroIsN a || successorIsN a -- Try both
successorIsN _ = False
The Natural Numbers

\[
\begin{align*}
\text{zero} & \quad \text{successor} \\
\emptyset \in \mathbb{N} & \quad a \in \mathbb{N} \\
\text{succ}(a) \in \mathbb{N} 
\end{align*}
\]

\textbf{data} Nat = Zero | Succ Nat

isNat :: Nat -> Bool

isNat Zero = True -- zero rule

isNat (Succ a) = isNat a -- successor rule

isNat is trivial; Haskell’s type system enforces it for us
Equality of Natural Numbers as an Inductive Definition

\[
\begin{align*}
  \text{eq(0, 0)} & \quad \text{equalzero} \\
  \text{eq(succ(a), succ(b))} & \quad \text{equal}
\end{align*}
\]

Judgements: \(eq(n_1, n_2)\) “\(n_1\) and \(n_2\) are equal” ← a relation/a set of pairs

Variables: \(a\, \ b\)

Symbols: \(0\, \ \text{succ}(\quad )\)

Is 3 = 3? A reverse derivation

\[
\text{eq(succ(succ(succ(0))), succ(succ(succ(0))))}
\]
Equality of Natural Numbers as an Inductive Definition

\[
\begin{align*}
\text{equalzero:} & \quad \text{eq}(0, 0) \\
\text{equal:} & \quad \frac{\text{eq}(a, b)}{\text{eq}(\text{succ}(a), \text{succ}(b))}
\end{align*}
\]

Judgements: \( \text{eq}(n_1, n_2) \) “\( n_1 \) and \( n_2 \) are equal”

Variables: \( a \quad b \)

Symbols: \( 0 \quad \text{succ}(\quad) \)

Is 3 = 3? A reverse derivation

\[
\begin{align*}
\text{eq}(\quad, \quad) & \quad \frac{\text{eq}(\text{succ}(\text{succ}(\text{succ}(0))), \text{succ}(\text{succ}(\text{succ}(0))))}{\text{equal}} \\
\text{eq}(\quad, \quad) & \quad \frac{\text{eq}(\text{succ}(\text{succ}(0)), \text{succ}(\text{succ}(0)))){\text{equal}} \\
\text{eq}(\quad, \quad) & \quad \frac{\text{eq}(0, \text{succ}(0))}{\text{equal}} \\
\end{align*}
\]

We are stuck: neither rule applies, so 1 \( \neq \) 2
Equality of Natural Numbers as an Inductive Definition

\[\begin{align*}
\text{equalzero} & : \text{eq}(0, 0) \\
\text{equal} & : \frac{\text{eq}(a, b)}{\text{eq}(\text{succ}(a), \text{succ}(b))}
\end{align*}\]

Judgements: \(\text{eq}(n_1, n_2)\) “\(n_1\) and \(n_2\) are equal”

Variables: \(a\), \(b\)

Symbols: \(0\), \(\text{succ}(\ )\)

Is 3 = 3? A reverse derivation

\[\begin{align*}
\text{eq}(\text{succ}(0), \text{succ}(0)) \quad &\text{equal} \\
\text{eq}(\text{succ}(\text{succ}(0)), \text{succ}(\text{succ}(0))) &\text{equal} \\
\text{eq}(\text{succ}(\text{succ}(\text{succ}(0))), \text{succ}(\text{succ}(\text{succ}(0)))) &\text{equal}
\end{align*}\]
Equality of Natural Numbers as an Inductive Definition

\[
\begin{align*}
\text{equalzero} & : \text{eq}(\emptyset, \emptyset) \\
\text{equal} & : \text{eq}(\text{succ}(a), \text{succ}(b)) \Rightarrow \text{eq}(a, b)
\end{align*}
\]

Judgements: \( \text{eq}(n_1, n_2) \) "\( n_1 \) and \( n_2 \) are equal"
Variables: \( a \quad b \)
Symbols: \( \emptyset \quad \text{succ}() \)

Is \( 3 = 3? \) A reverse derivation

\[
\begin{align*}
\text{eq}(0, 0) & \\
\text{eq}(\text{succ}(0), \text{succ}(0)) & \\
\text{eq}(\text{succ}(\text{succ}(0)), \text{succ}(\text{succ}(0))) & \\
\text{eq}(\text{succ}(\text{succ}(\text{succ}(0))), \text{succ}(\text{succ}(\text{succ}(0)))) &
\end{align*}
\]
Equality of Natural Numbers as an Inductive Definition

\[
\begin{align*}
\text{equalzero:} & & \text{eq}(0,0) \\
\text{equal:} & & \text{eq}(\text{succ}(a), \text{succ}(b)) \\
\end{align*}
\]

Judgements: \( \text{eq}(n_1, n_2) \) “\( n_1 \) and \( n_2 \) are equal”

Variables: \( a \) \( b \)

Symbols: \( 0 \) \( \text{succ}(\ ) \)

Is 3 = 3? A reverse derivation

\[
\begin{align*}
\text{eq}(&0,0) &\text{equalzero}
\text{eq}(\text{succ}(0),\text{succ}(0)) &\text{equal}
\text{eq}(\text{succ}(\text{succ}(0)),\text{succ}(\text{succ}(0))) &\text{equal}
\text{eq}(\text{succ}(\text{succ}(\text{succ}(0))),\text{succ}(\text{succ}(\text{succ}(0)))) &\text{equal}
\end{align*}
\]
Equality of Natural Numbers as an Inductive Definition

\[
\begin{align*}
\text{equalzero} & : \text{eq}(0, 0) \\
\text{equal} & : \text{eq}(\text{succ}(a), \text{succ}(b)) \rightarrow \text{eq}(a, b)
\end{align*}
\]

Judgements: \(\text{eq}(n_1, n_2)\) “\(n_1\) and \(n_2\) are equal”

Variables: \(a\), \(b\)

Symbols: \(0\), \(\text{succ}(\quad)\)

Is 3 = 3? A reverse derivation

\[
\begin{align*}
\text{eq}(0, 0) & \quad \text{equalzero} \\
\text{eq}(\text{succ}(0), \text{succ}(0)) & \quad \text{equal} \\
\text{eq}(\text{succ}(\text{succ}(0)), \text{succ}(\text{succ}(0))) & \quad \text{equal} \\
\text{eq}(\text{succ}(\text{succ}(\text{succ}(0))), \text{succ}(\text{succ}(\text{succ}(0)))) & \quad \text{equal}
\end{align*}
\]

Is 1 = 2?

\(\text{eq}(\text{succ}(0), \text{succ}(\text{succ}(0)))\)
Equality of Natural Numbers as an Inductive Definition

\[
\begin{align*}
\text{equalzero} & \quad \text{eq}(0, 0) \\
\text{equal} & \quad \text{eq}(\text{succ}(a), \text{succ}(b))
\end{align*}
\]

Judgements: \( \text{eq}(n_1, n_2) \) “\( n_1 \) and \( n_2 \) are equal”
Variables: \( a \quad b \)
Symbols: \( 0 \quad \text{succ}(\ ) \)

Is 3 = 3? A reverse derivation

\[
\begin{align*}
\text{eq}(0, 0) & \quad \text{equalzero} \\
\text{eq}(\text{succ}(0), \text{succ}(0)) & \quad \text{equal} \\
\text{eq}(\text{succ}(\text{succ}(0)), \text{succ}(\text{succ}(0))) & \quad \text{equal} \\
\text{eq}(\text{succ}(\text{succ}(\text{succ}(0))), \text{succ}(\text{succ}(\text{succ}(0)))) & \quad \text{equal}
\end{align*}
\]

Is 1 = 2?

\[
\begin{align*}
\text{eq}(0, \text{succ}(0)) & \quad \text{equal} \\
\text{eq}(\text{succ}(0), \text{succ}(\text{succ}(0))) & \quad \text{equal}
\end{align*}
\]
Equality of Natural Numbers as an Inductive Definition

\[
\begin{array}{c}
\text{eq}(\emptyset, \emptyset) \quad \text{equalzero} \\
\text{eq}(\text{succ}(a), \text{succ}(b)) \quad \text{equal}
\end{array}
\]

Judgements: \( \text{eq}(n_1, n_2) \) “\( n_1 \) and \( n_2 \) are equal”
Variables: \( a \quad b \)
Symbols: \( \emptyset \quad \text{succ}(\ ) \)

Is 3 = 3? A reverse derivation

\[
\begin{array}{c}
\text{eq}(\emptyset, \emptyset) \\
\text{eq}(\text{succ}(\emptyset), \text{succ}(\emptyset)) \\
\text{eq}(\text{succ}(\text{succ}(\emptyset)), \text{succ}(\text{succ}(\emptyset))) \\
\text{eq}(\text{succ}(\text{succ}(\text{succ}(\emptyset))), \text{succ}(\text{succ}(\text{succ}(\emptyset))))
\end{array}
\]

Is 1 = 2?

\[
\begin{array}{c}
\text{eq}(\emptyset, \text{succ}(\emptyset)) \\
\text{eq}(\text{succ}(\emptyset), \text{succ}(\text{succ}(\emptyset))) \quad \text{equal}
\end{array}
\]

We are stuck: neither rule applies, so 1 ≠ 2
Equality of Natural Numbers as an Inductive Definition

\[
\begin{align*}
&\text{equalzero} & \theta = \theta \\
&\text{equal} & a = b \\
&\text{equal} & \text{succ}(a) = \text{succ}(b)
\end{align*}
\]

Judgements: \( n_1 = n_2 \) “\( n_1 \) and \( n_2 \) are equal” ← a relation/a set of pairs

Variables: \( a \quad b \)

Symbols: \( \theta \quad \text{succ}( ) \)

Is \( 3 = 3 \)?

\[
\begin{align*}
&\text{equalzero} & \theta = \theta \\
&\text{equal} & \text{succ}(\theta) = \text{succ}(\theta) \\
&\text{equal} & \text{succ}(\text{succ}(\theta)) = \text{succ}(\text{succ}(\theta)) \\
&\text{equal} & \text{succ}(\text{succ}(\text{succ}(\theta))) = \text{succ}(\text{succ}(\text{succ}(\theta)))
\end{align*}
\]

Is \( 1 = 2 \)?

\[
\begin{align*}
&\text{equal} & \theta = \text{succ}(\theta) \\
&\text{equal} & \text{succ}(\theta) = \text{succ}(\text{succ}(\theta))
\end{align*}
\]

We are stuck: neither rule applies, so \( 1 \neq 2 \)
Equality of Natural Numbers as an Inductive Definition

\[
\begin{align*}
\text{equalzero:} & \quad \begin{array}{c}
0 = 0 \\
\text{equal:} & \quad \begin{array}{c}
succ(a) = succ(b)
\end{array}
\end{array}
\end{align*}
\]

\textbf{data} \ Nat = \text{Zero} \mid \text{Succ} \ Nat

\textbf{natEqual} :: \ Nat \rightarrow \ Nat \rightarrow \text{Bool}

natEqual \text{Zero} \text{Zero} = \text{True} \quad \text{-- equalzero rule}
natEqual (\text{Succ} a) (\text{Succ} b) = \text{natEqual} a b \quad \text{-- equal rule}
natEqual _ _ = \text{False}

Again: single function because only one rule may ever match
Equality of Natural Numbers as an Inductive Definition

\[
\begin{align*}
\text{equalzero} & : \quad 0 = 0 \\
\text{equal} & : \quad a = b \quad \Rightarrow \quad \text{succ}(a) = \text{succ}(b)
\end{align*}
\]

\[\text{data Nat} = \text{Zero} \mid \text{Succ Nat}\]
\[\text{deriving Eq}\]

This Haskell’s default implementation of \(==\) for algebraic data types
Addition as an Inductive Definition

\[
\begin{align*}
  b & \in \mathbb{N} \quad \frac{\text{addzero}}{\text{sum}(0, b, b)} \\
  \frac{\text{add}}{\text{sum}(\text{succ}(a), b, \text{succ}(c))}
\end{align*}
\]

Judgments:
\[ n \in \mathbb{N} \quad \text{sum}(n_1, n_2, n_3) \leftarrow \text{a relation/a set of triples} \]

Variables:
\[ a \quad b \quad c \]

Symbols:
\[ 0 \quad \text{succ}(\quad) \]
Addition as an Inductive Definition

\[
\begin{align*}
  b \in \mathbb{N} & \quad \frac{}{\text{sum}(\theta, b, b)} \quad \text{addzero} \\
  \frac{}{\text{sum}(\text{succ}(a), b, \text{succ}(c))} & \quad \text{add}
\end{align*}
\]

Judgments: \( n \in \mathbb{N} \quad \text{sum}(n_1, n_2, n_3) \)
Variables: \( a \quad b \quad c \)
Symbols: \( \theta \quad \text{succ}(\quad) \)

Is \( 2 + 1 = 3? \)

\[
\text{sum}(\text{succ}(\text{succ}(\theta)), \text{succ}(\theta), \text{succ}(\text{succ}(\text{succ}(\theta))))
\]
Addition as an Inductive Definition

\[ b \in \mathbb{N} \quad \frac{\text{addzero}}{\text{sum}(\emptyset, b, b)} \]
\[ \frac{\text{add}}{\text{sum}(\text{succ}(a), b, \text{succ}(c))} \]

Judgments: \( n \in \mathbb{N} \quad \text{sum}(n_1, n_2, n_3) \)

Variables: \( a \quad b \quad c \)

Symbols: \( 0 \quad \text{succ}(\quad) \)

Is \( 2 + 1 = 3? \)

\[ \frac{\text{add}}{\text{sum}(\text{succ}(\emptyset), \text{succ}(\emptyset), \text{succ(succ}(\emptyset)))} \]
Addition as an Inductive Definition

\[ b \in \mathbb{N} \]
\[ \sum(0, b, b) \overset{\text{addzero}}{\Rightarrow} \]
\[ \sum(a, b, c) \overset{\text{add}}{\Rightarrow} \sum(\text{succ}(a), b, \text{succ}(c)) \]

Judgments: \( n \in \mathbb{N} \quad \sum(n_1, n_2, n_3) \)

Variables: \( a \quad b \quad c \)

Symbols: \( 0 \quad \text{succ}(\quad) \)

Is \( 2 + 1 = 3? \)

\[ \sum(0, \text{succ}(0), \text{succ}(0)) \overset{\text{add}}{\Rightarrow} \]
\[ \sum(\text{succ}(0), \text{succ}(0), \text{succ}(\text{succ}(0))) \overset{\text{add}}{\Rightarrow} \]
\[ \sum(\text{succ}(\text{succ}(0)), \text{succ}(0), \text{succ}(\text{succ}(\text{succ}(0)))) \]
Addition as an Inductive Definition

\[
\begin{align*}
& b \in \mathbb{N} \quad \frac{\text{sum}(0, b, b)}{\text{addzero}} \\
& \frac{\text{sum}(a, b, c)}{\text{add}} \quad \frac{\text{sum}(\text{succ}(a), b, \text{succ}(c))}{\text{add}}
\end{align*}
\]

Judgments: \( n \in \mathbb{N} \) \( \text{sum}(n_1, n_2, n_3) \)
Variables: \( a \quad b \quad c \)
Symbols: \( 0 \quad \text{succ}(\quad) \)

Is \( 2 + 1 = 3? \)

\[
\begin{align*}
& \text{succ}(0) \in \mathbb{N} \\
& \frac{\text{sum}(0, \text{succ}(0), \text{succ}(0))}{\text{addzero}} \\
& \frac{\text{sum}(\text{succ}(0), \text{succ}(0), \text{succ}(\text{succ}(0)))}{\text{add}} \\
& \frac{\text{sum}(\text{succ}(\text{succ}(0)), \text{succ}(0), \text{succ}(\text{succ}(\text{succ}(0))))}{\text{add}}
\end{align*}
\]
Addition as an Inductive Definition

\[
\begin{align*}
  b \in \mathbb{N} \quad & \frac{}{\text{addzero} \quad \text{sum}(\emptyset, b, b)} \\
  \quad \quad \text{add} \quad & \frac{\text{sum}(a, b, c)}{\text{sum}(\text{succ}(a), b, \text{succ}(c))}
\end{align*}
\]

Judgments: \( n \in \mathbb{N} \quad \text{sum}(n_1, n_2, n_3) \)

Variables: \( a \quad b \quad c \)

Symbols: \( 0 \quad \text{succ}(\quad) \)

Is \( 2 + 1 = 3? \)

\[
\begin{align*}
  0 \in \mathbb{N} \quad & \frac{}{\text{successor} \quad \text{succ}(0) \in \mathbb{N}} \\
  \quad \quad \text{addzero} \quad & \frac{\text{sum}(0, \text{succ}(0), \text{succ}(0))}{\text{add}} \\
  \quad \quad \text{add} \quad & \frac{\text{sum}(\text{succ}(0), \text{succ}(0), \text{succ}(\text{succ}(0)))}{\text{add}} \\
  \quad \quad \text{add} \quad & \frac{\text{sum}(\text{succ}(\text{succ}(0)), \text{succ}(0), \text{succ}(\text{succ}(\text{succ}(0))))}{\text{add}}
\end{align*}
\]
Addition as an Inductive Definition

\[
\begin{align*}
    b \in \mathbb{N} & \quad \frac{\text{sum}(\theta, b, b)}{\text{addzero}} \\
    \text{sum}(a, b, c) & \quad \frac{\text{sum}(\text{succ}(a), b, \text{succ}(c))}{\text{add}}
\end{align*}
\]

Judgments: \( n \in \mathbb{N} \quad \text{sum}(n_1, n_2, n_3) \)

Variables: \( a \quad b \quad c \)

Symbols: \( \theta \quad \text{succ}(\quad) \)

Is \( 2 + 1 = 3? \)

\[
\begin{align*}
    \theta \in \mathbb{N} & \quad \frac{\text{zero}}{} \\
    \text{succ}(\theta) \in \mathbb{N} & \quad \frac{\text{successor}}{} \\
    \text{sum}(\theta, \text{succ}(\theta), \text{succ}(\theta)) & \quad \frac{\text{addzero}}{} \\
    \text{sum}(\text{succ}(\theta), \text{succ}(\theta), \text{succ}(\text{succ}(\theta))) & \quad \frac{\text{add}}{} \\
    \text{sum}(\text{succ}(\text{succ}(\theta)), \text{succ}(\theta), \text{succ}(\text{succ}(\text{succ}(\theta)))) & \quad \frac{\text{add}}{}
\end{align*}
\]
Addition as an Inductive Definition

\[
\begin{align*}
\text{addzero} & : b \in \mathbb{N} \\
\theta + b &= b \\
\text{add} & : a + b = c \\
\text{succ}(a) + b &= \text{succ}(c)
\end{align*}
\]

Judgments: \( n \in \mathbb{N} \quad n_1 + n_2 = n_3 \) ← a relation/a set of triples

Variables: \( a \quad b \quad c \)

Symbols: \( \theta \quad \text{succ}() \)

Is \( 2 + 1 = 3? \)

\[
\begin{align*}
\text{zero} & : \theta \in \mathbb{N} \\
\text{successor} & : \text{succ}(\theta) \in \mathbb{N} \\
\text{addzero} & : \theta + \text{succ}(\theta) = \text{succ}(\theta) \\
\text{add} & : \text{succ}(\theta) + \text{succ}(\theta) = \text{succ}(\text{succ}(\theta)) \\
\text{add} & : \text{succ}(\text{succ}(\theta)) + \text{succ}(\theta) = \text{succ}(\text{succ}(\text{succ}(\theta)))
\end{align*}
\]
Addition as an Inductive Definition

\[
\begin{align*}
\text{addzero} & : \quad \forall b \in \mathbb{N}. \quad \emptyset + b = b \\
\text{add} & : \quad \forall a, b, c \in \mathbb{N}. \quad a + b = c \Rightarrow \text{succ}(a) + b = \text{succ}(c)
\end{align*}
\]

```haskell
data Nat = Zero | Succ Nat deriving Eq

sumsTo :: Nat -> Nat -> Nat -> Bool
-- Is \(a + b = c\)?
sumsTo Zero b b' | b == b' = True  -- addzero rule
sumsTo (Succ a) b (Succ c) = sumsTo a b c  -- add rule
sumsTo _ _ _ = False  -- E.g., (Succ a) _ Zero
```

No need to check whether \(b \in \mathbb{N}\): the types enforce this
Haskell patterns can’t check for equality like \(\text{sumsTo Zero b b}\), so I added guard \(b == 'b\)
Rather awkward to ask “is this it?”
**Addition as an Inductive Definition**

\[
\begin{align*}
b \in \mathbb{N} \\
\theta + b &= b \quad \text{addzero} \quad & a + b &= c \\
\text{succ}(a) + b &= \text{succ}(c) \quad \text{add}
\end{align*}
\]

**data Nat = Zero | Succ Nat**

```haskell
deriving (Eq, Show)
```

**addNat :: Nat -> Nat -> Nat**  -- Given \(a\) and \(b\), what \(c\) satisfies \(a + b = c\)?

- `addNat Zero b = b`  -- addzero rule
- `addNat (Succ a) b = Succ (addNat a b)`  -- add rule

The dataflow makes this easy and it’s obviously a total function
Addition as an Inductive Definition

\[
\begin{align*}
\[ b \in \mathbb{N} \] & \quad \text{addzero} \\
0 + b &= b \\
\hline
\text{add} & \\
\frac{a + b = c}{\text{succ}(a) + b = \text{succ}(c)}
\end{align*}
\]

\textbf{data} Nat = Zero | Succ Nat

\textbf{deriving} (Eq, Show)

\textbf{subNat} :: Nat -> Nat -> \text{Maybe} Nat

-- Given \( c \) and \( a \), what \( b \) satisfies \( a + b = c \)?

\text{subNat} c \text{ Zero } = \text{Just} c \quad -- \text{addzero rule}

\text{subNat} (\text{Succ} \ c) (\text{Succ} \ a) = \text{subNat} \ c \ a \quad -- \text{add rule}

\text{subNat Zero} (\text{Succ} \ _) = \text{Nothing} \quad -- \text{failure}

Still straightforward dataflow, but the function is no longer total
A Definition of Binary Trees

Judgments:  $t$ tree  "$t$ is a tree"

Variables:  $t_1$  $t_2$  $t_1$ and $t_2$ may be equal

Symbols:  leaf  branch( , )
A Definition of Binary Trees

Judgments: $t$ tree “$t$ is a tree”
Variables: $t_1$ $t_2$ $t_1$ and $t_2$ may be equal
Symbols: leaf branch( , )

Derivations are generally tree-structured

$\text{branch(\text{branch(leaf,leaf),leaf)}\text{ tree}$
A Definition of Binary Trees

\[ \text{leaf} \quad \text{tree} \quad \text{branch} \]

\[ t_1 \text{ tree} \quad t_2 \text{ tree} \quad \text{branch}(t_1, t_2) \text{ tree} \]

Judgments: \( t \) tree "\( t \) is a tree"

Variables: \( t_1 \quad t_2 \quad t_1 \) and \( t_2 \) may be equal

Symbols: \( \text{leaf} \quad \text{branch}(\ , \ ) \)

Derivations are generally tree-structured

\[ \text{branch(leaf,leaf) tree} \quad \text{leaf tree} \quad \text{branch} \]

\[ \text{branch(\text{branch(leaf,leaf),leaf\)}} \text{ tree} \]
A Definition of Binary Trees

\[
\begin{align*}
\text{leaf} & \quad \text{leaf} & \quad \text{tree} & \quad t_1 & \quad \text{tree} & \quad t_2 & \quad \text{tree} & \quad \text{branch}
\end{align*}
\]

Judgments: \( t \) tree “\( t \) is a tree”
Variables: \( t_1 \quad t_2 \quad t_1 \) and \( t_2 \) may be equal
Symbols: \( \text{leaf} \quad \text{branch}(\ , \ ) \)

Derivations are generally tree-structured

\[
\begin{align*}
\text{leaf} & \quad \text{tree} & \quad \text{leaf} & \quad \text{tree} & \quad \text{branch} & \quad \text{leaf} & \quad \text{leaf} & \quad \text{tree} & \quad \text{branch}\[\text{branch(leaf,leaf),leaf)} & \quad \text{tree}
\end{align*}
\]
A Definition of Binary Trees

\[
\text{leaf} \quad \text{tree} \quad \text{branch}
\]

\[
t_1 \text{ tree} \quad t_2 \text{ tree} \quad \text{branch}(t_1, t_2) \text{ tree}
\]

**Judgments:** \( t \) tree “\( t \) is a tree”

**Variables:** \( t_1 \quad t_2 \quad t_1 \) and \( t_2 \) may be equal

**Symbols:** \text{leaf} \quad \text{branch}(, , )

Derivations are generally tree-structured

\[
\text{leaf} \quad \text{tree} \quad \text{branch(leaf, leaf) tree} \\
\text{leaf} \quad \text{tree} \quad \text{branch(leaf, leaf) tree} \\
\text{branch(leaf, leaf), leaf) tree}
\]
A Definition of Binary Trees

Judgments: \( t \) tree "t is a tree"
Variables: \( t_1 \) \( t_2 \) \( t_1 \) and \( t_2 \) may be equal
Symbols: leaf \( \text{branch}(\ , \) \)

\[
\begin{align*}
\text{data} \quad \text{Tree} &= \text{Leaf} \mid \text{Branch Tree Tree} \\
\text{isTree} :: \text{Tree} &\rightarrow \text{Bool} \\
\text{isTree} \ \text{Leaf} &= \text{True} \\
\text{isTree} \ (\text{Branch} \ l \ r) &= \text{isTree} \ l \ &\& \text{isTree} \ r \quad \text{-- Must test both branches}
\end{align*}
\]

Trivially true because of Haskell’s types, but note two-way recursion