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
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 $\rightarrow J_1 \ J_2 \ \ldots \ J_k$

Conclusion: A Judgment $\rightarrow \ \ J$

“If all the premises hold, the conclusion follows”

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

Axiom $\rightarrow 0 \in \mathbb{N}$ zero

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

Variables: $a$ ← Sequences of symbols

Symbols: $0$ succ( )

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>succ(0)</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>succ(succ(0))</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>succ(succ(succ(0)))</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>succ(succ(succ(succ(0))))</td>
<td>4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Inference Rule

Premises: Judgments \(\rightarrow\) \(J_1, J_2, \ldots, J_k\) \(\rightarrow\) Conclusion: A Judgment \(\rightarrow\) \(J\)

“"If all the premises hold, the conclusion follows”"

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

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

Judgments: \(a \in \mathbb{N}\)  \(\text{succ}(0)\)  \(\text{succ}(\text{succ}(0))\)  \(\text{succ}(\text{succ}(\text{succ}(0)))\)

Variables: \(\text{a} \leftarrow \text{Sequences of symbols}\)

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

Premises: Judgments $\rightarrow J_1 \ J_2 \ \cdots \ J_k$

Conclusion: A Judgment $\rightarrow J$

Rule-Name

“If all the premises hold, the conclusion follows”

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

$0 \in \mathbb{N}$ zero

$a \in \mathbb{N}$ successor

$succ(0) \in \mathbb{N}$

$succ(succ(0)) \in \mathbb{N}$

$succ(true) \in \mathbb{N}$

Technically a scheme

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

$0 \in \mathbb{N}$

$succ(0) \in \mathbb{N}$

$succ(succ(0)) \in \mathbb{N}$

$succ(true) \in \mathbb{N}$

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

Consistent replacement only:

$foo \in \mathbb{N}$

$succ(bar) \in \mathbb{N}$

is not a rule
Inference Rule

Premises: Judgments → \( J_1 \), \( J_2 \), ..., \( J_k \) → \( J \) → Conclusion: A Judgment → Rule-Name

“If all the premises hold, the conclusion follows”

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

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

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

\[
0 \in \mathbb{N}
\]
Inference Rule

Premises: Judgments \[ J_1 J_2 \ldots J_k \] 
Conclusion: A Judgment \[ J \]

"If all the premises hold, the conclusion follows"

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

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

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

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

0 ∈ N \text{zero}

a ∈ N \text{successor}

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

0 ∈ N \text{zero}

succ(0) ∈ N \text{successor}

succ(succ(0)) ∈ N \text{successor} \leftarrow \text{choose } a = \text{succ}(0)
Inference Rule

Premises: Judgments → \[ J_1 \quad J_2 \quad \cdots \quad J_k \]  

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

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

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

zeroIsN :: String -> Bool

successorIsN :: String -> Bool

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

\[ 0 \in \mathbb{N} \quad \text{successor} \quad \text{succ}(a) \in \mathbb{N} \]

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

\[
\text{successorIsN} &: \text{String} \rightarrow \text{Bool}
\]
The Natural Numbers

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

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
The Natural Numbers

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

```haskell
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')
```
The Natural Numbers

### Zero

\[ 0 \in \mathbb{N} \]

### Successor

\[ \text{succ}(a) \in \mathbb{N} \]

---

```haskell
import Data.List (stripPrefix)

isNat :: String -> Bool

isNat "0" = True  -- zero rule
isNat s = case match "succ(" ")" s of
  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

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

\textbf{data} Nat = Zero | Succ Nat \quad -- \text{Algebraic data type: either “Zero” or “Succ } \, n\text{”}

\begin{align*}
\text{zeroIsN} & \quad :: \text{Nat} \to \text{Bool} \\
\text{zeroIsN Zero} & = \text{True} \\
\text{zeroIsN } \_ & = \text{False}
\end{align*}

\begin{align*}
\text{successorIsN} & \quad :: \text{Nat} \to \text{Bool} \\
\text{successorIsN (Succ a)} & = \text{zeroIsN a} \lor \text{successorIsN a} \quad -- \text{Try both} \\
\text{successorIsN } \_ & = \text{False}
\end{align*}
The Natural Numbers

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

\textbf{data} Nat = Zero | Succ Nat

\textbf{isNat} :: Nat \to \textbf{Bool}

\text{isNat} \ Zero = \textbf{True} \quad -- \text{zero rule}

\text{isNat} \ (\text{Succ} \ a) = \text{isNat} \ a \quad -- \text{successor rule}

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

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

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

Variables: \( a \quad b \)

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

Is 3 = 3? A reverse derivation

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

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

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

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

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

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

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

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

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

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

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

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

Variables: \( a \quad b \)

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

Is 3 = 3? A reverse derivation

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

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

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

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

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?

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

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

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

Variables: \( a \quad b \)

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

Is 3 = 3? A reverse derivation

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

Is 1 = 2?

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

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

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

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

Variables: \( a \quad b \)

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

Is \( 3 = 3? \)

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

Is \( 1 = 2? \)

\[ \begin{align*}
0 &= \text{succ}(0) & \text{equal} \\
\text{succ}(0) &= \text{succ}(\text{succ}(0))
\end{align*} \]

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

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

**data** Nat = Zero | Succ Nat

natEqual :: Nat -> Nat -> Bool

natEqual Zero Zero = True -- equalzero rule

natEqual (Succ a) (Succ b) = natEqual a b -- equal rule

natEqual _ _ = False

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

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

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

\text{deriving Eq}

This Haskell’s default implementation of \text{==} for algebraic data types
Addition as an Inductive Definition

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

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

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

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

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

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

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

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

Is \( 2 + 1 = 3? \)

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

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

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

Variables: \( a \ b \ c \)

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

Is \( 2 + 1 = 3? \)
Addition as an Inductive Definition

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

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 \( 2 + 1 = 3? \)

\[
\begin{align*}
\text{add} & \quad \frac{}{} \\
\text{add} & \quad \frac{}{}
\end{align*}
\]

\[
\begin{align*}
\text{add} & \quad \frac{}{}
\end{align*}
\]

\[
\begin{align*}
\text{add} & \quad \frac{}{}
\end{align*}
\]

\[
\begin{align*}
\text{add} & \quad \frac{}{}
\end{align*}
\]

\[
\begin{align*}
\text{add} & \quad \frac{}{}
\end{align*}
\]
Addition as an Inductive Definition

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

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

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

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

Is 2 + 1 = 3?

\[ \text{succ}(0) \in \mathbb{N} \]

\[ \frac{\text{sum}(0, \text{succ}(0), \text{succ}(0))}{\text{addzero}} \quad \frac{\text{sum}(\text{succ}(0), \text{succ}(0), \text{succ}(\text{succ}(0)))}{\text{add}} \]

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

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

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 \( 2 + 1 = 3? \)

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

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

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

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

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

Is \( 2 + 1 = 3? \)

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

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

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

Is \( 2 + 1 = 3 \)?

\[
\begin{align*}
\text{zero:} & \quad 0 \in \mathbb{N} \\
\text{successor:} & \quad \text{succ}(0) \in \mathbb{N} \\
\text{addzero:} & \quad 0 + \text{succ}(0) = \text{succ}(0) \\
\text{add:} & \quad \text{succ}(0) + \text{succ}(0) = \text{succ}(\text{succ}(0)) \\
\text{add:} & \quad \text{succ}(\text{succ}(0)) + \text{succ}(0) = \text{succ}(\text{succ}(\text{succ}(0)))
\end{align*}
\]
Addition as an Inductive Definition

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

\[\begin{align*}
  \text{data} & \quad \text{Nat} = \text{Zero} | \text{Succ} \text{ Nat} \\
  \text{deriving} & \quad \text{Eq}
\end{align*}\]

\[
\text{sumsTo} :: \text{Nat} \rightarrow \text{Nat} \rightarrow \text{Nat} \rightarrow \text{Bool} \quad -- \text{Is } a + b = c?
\]

\[
\begin{align*}
  \text{sumsTo} \text{ Zero} \quad b \quad b' \mid b == b' & = \text{True} \quad -- \text{addzero rule} \\
  \text{sumsTo} \text{ (Succ a)} \quad b \quad \text{ (Succ c)} & = \text{sumsTo} \text{ a} \quad b \quad c \quad -- \text{add rule} \\
  \text{sumsTo} \quad _ \quad _ \quad _ & = \text{False} \quad -- \text{E.g., (Succ a) _ Zero}
\end{align*}
\]

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

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

\[
\begin{align*}
\quad \text{data Nat} &= \text{Zero} \mid \text{Succ Nat} \\
&\quad \text{deriving (Eq, Show)} \\
\quad \text{addNat} :: \text{Nat} \rightarrow \text{Nat} \rightarrow \text{Nat} \\
&\quad \quad \text{-- Given a and b, what c satisfies } a + b = c? \\
\quad \text{addNat} \ \text{Zero} \quad b &= b \\
\quad \text{addNat} \ \text{Succ a} \quad b &= \text{Succ (addNat a b)} \\
&\quad \quad \text{-- add zero rule} \\
&\quad \quad \text{-- add rule} \\
\end{align*}
\]

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} \\
  \text{\(0 + b = b\)} & \\
  a + b = c & \quad \text{add} \\
  \text{\(\text{succ}(a) + b = \text{succ}(c)\)} &
\end{align*}
\]

\textbf{data} Nat = Zero | Succ Nat

\textbf{deriving} (Eq, Show)

\textbf{subNat} :: Nat \to Nat \to \text{Maybe} Nat  
\quad \text{-- Given \(c\) and \(a\), what \(b\) satisfies \(a + b = c\)？}

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

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

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

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

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

\[
\begin{array}{c}
\text{leaf} \\
\text{tree}
\end{array}
\quad
\begin{array}{c}
\text{branch} \\
(t_1, t_2) \\
\text{tree}
\end{array}
\]

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

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

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

Derivations are generally tree-structured

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

Derivations are generally tree-structured
A Definition of Binary Trees

\[
\begin{align*}
\text{leaf} & \quad \text{tree} \\
\text{branch(} & \quad \text{t}_1 \quad \text{t}_2 \quad \text{tree} \\
\text{branch(} & \quad \text{t}_1, \quad \text{t}_2) \quad \text{tree}
\end{align*}
\]

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

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

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

Derivations are generally tree-structured

\[
\begin{align*}
\text{leaf} & \quad \text{tree} \\
\text{branch(} & \quad \text{leaf,leaf) \quad \text{tree} \\
\text{branch(} & \quad \text{branch(leaf,leaf),leaf) \quad \text{tree}
\end{align*}
\]
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 \( \text{branch}(, , ) \)

Derivations are generally tree-structured

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

**data**  Tree = Leaf | Branch Tree Tree

**isTree :: Tree \rightarrow Bool**

\( \text{isTree} \text{ Leaf} = \text{True} \)

\( \text{isTree} \text{ (Branch } l r) = \text{isTree} \text{ } l \&\& \text{isTree} \text{ } r \quad \text{-- Must test both branches} \)

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