Chick: A Dependently-Typed List Language

Abd-El-Aziz Zayed Louis Hildebrand

McGill University

April 10, 2025

Motivation

Many list-related functions have constraints related to the list length. [1] Can we check them statically?

Module List

```
module List: sig .. end
```

List operations.

```
val tl : 'a list -> 'a list
```

Return the given list without its first element.

Raises `Failure` if the list is empty.

```
val nth : 'a list -> int -> 'a
```

Return the `n`-th element of the given list. The first element (head of the list) is at position 0.

Raises

- `Failure` if the list is too short.

```
val map2 : ('a -> 'b -> 'c) -> 'a list -> 'b list -> 'c list
```

`map2 f [a1; ...; an] [b1; ...; bn]` is `[f a1 b1; ...; f an bn]`.

Raises `Invalid_argument` if the two lists are determined to have different lengths.

[1] https://ocaml.org/manual/5.1/api/List.html

Outline

Syntax of Types

Type	A	$::=$	$\texttt{Nat}$	Natural numbers
		$\mid$	$\texttt{Vec}\ \ell$	Fixed-size list of nats
		$\mid$	$\Pi(x\colon A_1).A_2$	Dependent functions
Length	ℓ	$::=$	$n \mid x \mid \ell_1 + \ldots + \ell_k$	
Numeral	n	$\in$	$\mathbb{N}$	

Note

$A_1 \to A_2 \triangleq \Pi(\_\colon A_1).A_2$

Syntax of Terms

$$
\begin{array}{llll}
\text{Syn} & s & ::= & x \mid s\ t \\
 & & \mid & \texttt{head}\ \ell\ s \mid \texttt{tail}\ \ell\ s \\
\text{Chk} & t & ::= & s \mid \lambda x.t \mid \texttt{fix}\ x.t \\
 & & \mid & n \mid t_1 + \ldots + t_k \\
 & & \mid & (\texttt{match}\ s\ \texttt{with} \mid 0 \to t_1 \mid x + 1 \to t_2) \\
 & & \mid & \texttt{nil} \mid \texttt{cons}\ \ell\ t_1\ t_2 \\
 & & \mid & (\texttt{match}\ s\ \texttt{with} \mid \texttt{nil} \to t_1 \mid \texttt{cons}\ x_1\ x_2\ x_3 \to t_2)
\end{array}
$$

Example

$$\texttt{cons} \; : \; \Pi(n\colon \texttt{Nat}).\texttt{Nat} \to \texttt{Vec}\; n \to \texttt{Vec}\;(n+1)$$

i.e., $\texttt{cons} \; : \; \Pi(n\colon \texttt{Nat}).\Pi(\_\colon \texttt{Nat}).\Pi(\_\colon \texttt{Vec}\; n).\texttt{Vec}\;(n+1)$

e.g., $[10] \triangleq \texttt{cons}\;(n=0)\;10\;\texttt{nil}$

Example

$$\texttt{cons} \ : \ \Pi(n\colon \texttt{Nat}).\texttt{Nat} \to \texttt{Vec}\ n \to \texttt{Vec}\ (n+1)$$

i.e., $\texttt{cons} \ : \ \Pi(n\colon \texttt{Nat}).\Pi(\_\colon \texttt{Nat}).\Pi(\_\colon \texttt{Vec}\ n).\texttt{Vec}\ (n+1)$

e.g., $[10] \triangleq \texttt{cons}\ (n = 0)\ 10\ \texttt{nil}$

Example

$$\begin{aligned}
&\texttt{cons} : \Pi(n\colon \texttt{Nat}).\texttt{Nat} \to \texttt{Vec}\ n \to \texttt{Vec}\ (n+1) \\
\text{i.e.,}\quad &\texttt{cons} : \Pi(n\colon \texttt{Nat}).\Pi(_\colon \texttt{Nat}).\Pi(_\colon \texttt{Vec}\ n).\texttt{Vec}\ (n+1) \\
\text{e.g.,}\quad &[10] \triangleq \texttt{cons}\ (n=0)\ 10\ \texttt{nil}
\end{aligned}$$

Example

$$\begin{aligned}
& \texttt{cons} \;:\; \Pi(n\colon \texttt{Nat}).\texttt{Nat} \to \texttt{Vec}\ n \to \texttt{Vec}\ (n+1) \\
\text{i.e.,}\quad & \texttt{cons} \;:\; \Pi(n\colon \texttt{Nat}).\Pi(_\colon \texttt{Nat}).\Pi(_\colon \texttt{Vec}\ n).\texttt{Vec}\ (n+1) \\
\text{e.g.,}\quad & [10] \triangleq \texttt{cons}\ (n=0)\ 10\ \texttt{nil}
\end{aligned}$$

One More Example

e.g., $\texttt{drop}\ (k = 2)\ (n = 3)\ [9, 8, 7, 6, 5] = [7, 6, 5]$

// Discard first k elements

$$
\begin{aligned}
&\texttt{drop}\ :\ \Pi(k\colon \texttt{Nat}).\Pi(n\colon \texttt{Nat}).\texttt{Vec}\ (k+n) \to \texttt{Vec}\ n = \\
&\quad \texttt{fix}\ \mathrm{drop}.\lambda k.\lambda n.\lambda v. \\
&\qquad \texttt{match}\ k\ \texttt{with} \\
&\qquad \mid 0 \to \texttt{nil} \\
&\qquad \mid k' + 1 \to \texttt{drop}\ k\ n\ (\texttt{tail}\ (k' + n)\ v)
\end{aligned}
$$

One More Example

e.g., $\texttt{drop}\ (k = 2)\ (n = 3)\ [9, 8, 7, 6, 5] = [7, 6, 5]$

Can you spot the mistakes?

$$
\begin{aligned}
&// \text{ Discard first } k \text{ elements}\\
&\texttt{drop} \;:\; \Pi(k\colon \texttt{Nat}).\Pi(n\colon \texttt{Nat}).\texttt{Vec}\ (k+n) \to \texttt{Vec}\ n =\\
&\quad \texttt{fix}\ \mathsf{drop}.\lambda k.\lambda n.\lambda v.\\
&\qquad \texttt{match}\ k\ \texttt{with}\\
&\qquad \mid 0 \to \texttt{nil}\\
&\qquad \mid k' + 1 \to \texttt{drop}\ k\ n\ (\texttt{tail}\ (k' + n)\ v)
\end{aligned}
$$

One More Example

e.g., $\texttt{drop}\ (k = 2)\ (n = 3)\ [9, 8, 7, 6, 5] = [7, 6, 5]$

Can you spot the mistakes?

// Discard first k elements

$$
\begin{aligned}
&\texttt{drop} \;:\; \Pi(k\colon \texttt{Nat}).\Pi(n\colon \texttt{Nat}).\texttt{Vec}\ (k+n) \to \texttt{Vec}\ n = \\
&\quad \texttt{fix}\ \text{drop}.\lambda k.\lambda n.\lambda v. \\
&\qquad \texttt{match}\ k\ \texttt{with} \\
&\qquad \mid 0 \to \texttt{nil} \qquad \texttt{nil}\ \text{does not have type}\ \texttt{Vec}\ n \\
&\qquad \mid k' + 1 \to \texttt{drop}\ k\ n\ (\texttt{tail}\ (k' + n)\ v)
\end{aligned}
$$

One More Example

e.g., $\texttt{drop}\ (k = 2)\ (n = 3)\ [9, 8, 7, 6, 5] = [7, 6, 5]$

Can you spot the mistakes?

$// \text{ Discard first } k \text{ elements}$

$$\texttt{drop} \ :\ \Pi(k\colon \texttt{Nat}).\Pi(n\colon \texttt{Nat}).\texttt{Vec}\ (k+n) \to \texttt{Vec}\ n =$$

$$\texttt{fix}\ \mathrm{drop}.\lambda k.\lambda n.\lambda v.$$

$$\texttt{match}\ k\ \texttt{with}$$

$$\mid 0 \to v \qquad v : \texttt{Vec}\ (0+n)$$

$$\mid k' + 1 \to \texttt{drop}\ k\ n\ (\texttt{tail}\ (k'+n)\ v)$$

One More Example

e.g., $\texttt{drop}\ (k = 2)\ (n = 3)\ [9, 8, 7, 6, 5] = [7, 6, 5]$

Can you spot the mistakes?

$$
\begin{aligned}
&// \text{ Discard first } k \text{ elements} \\
&\texttt{drop} \ : \ \Pi(k : \texttt{Nat}).\Pi(n : \texttt{Nat}).\texttt{Vec}\ (k + n) \to \texttt{Vec}\ n = \\
&\quad \texttt{fix}\ \texttt{drop}.\lambda k.\lambda n.\lambda v. \\
&\quad\quad \texttt{match}\ k\ \texttt{with} \\
&\quad\quad \mid 0 \to v \qquad v : \texttt{Vec}\ (0 + n) \\
&\quad\quad \mid k' + 1 \to \texttt{drop}\ k\ n\ (\texttt{tail}\ (k' + n)\ v)
\end{aligned}
$$

$\texttt{tail}\ (k' + n)\ v$ does not have type $\texttt{Vec}\ (k + n)$

One More Example

e.g., $\texttt{drop}\ (k = 2)\ (n = 3)\ [9, 8, 7, 6, 5] = [7, 6, 5]$

All good now :)

// Discard first k elements

$$
\begin{aligned}
&\texttt{drop} \;:\; \Pi(k\colon \texttt{Nat}).\Pi(n\colon \texttt{Nat}).\texttt{Vec}\ (k+n) \to \texttt{Vec}\ n = \\
&\quad \texttt{fix}\ \texttt{drop}.\lambda k.\lambda n.\lambda v. \\
&\qquad \texttt{match}\ k\ \texttt{with} \\
&\qquad \mid 0 \to v \qquad v : \texttt{Vec}\ (0+n) \\
&\qquad \mid k'+1 \to \texttt{drop}\ k'\ n\ (\texttt{tail}\ (k'+n)\ v)
\end{aligned}
$$

$v : \texttt{Vec}\ (k'+1+n), \texttt{tail}\ \ldots : \texttt{Vec}\ (k'+n), \texttt{drop}\ \ldots : \texttt{Vec}\ n$

Judgements

$\boxed{\Gamma \vdash s \Longrightarrow A}$ (In context Γ, s synthesizes type A)

$\boxed{\Gamma \vdash t \Longleftarrow A}$ (In context Γ, check that t has type A)

$\boxed{A_1 \equiv A_2}$ (Types are equivalent)

$\boxed{\ell_1 \equiv \ell_2}$ (Lengths are equivalent)

Rules: Checking

$$\boxed{\Gamma \vdash t \Longleftarrow A}$$

$$\frac{}{\Gamma \vdash \texttt{nil} \Longleftarrow \texttt{Vec}\ 0}$$

Rules: Checking

$$\boxed{\Gamma \vdash t \Longleftarrow A}$$

$$\frac{}{\Gamma \vdash \texttt{nil} \Longleftarrow \texttt{Vec}\ 0}$$

What about $\Gamma \vdash \texttt{nil} \Longleftarrow \texttt{Vec}\ (0 + 0)$?

Rules: Checking

$$\boxed{\Gamma \vdash t \Longleftarrow A}$$

$$\frac{\ell \equiv 0}{\Gamma \vdash \mathtt{nil} \Longleftarrow \mathtt{Vec}\ \ell}$$

Rules: Checking

$$\boxed{\Gamma \vdash t \Longleftarrow A}$$

$$\frac{\ell \equiv 0}{\Gamma \vdash \texttt{nil} \Longleftarrow \texttt{Vec}\ \ell}$$

$$\frac{\begin{array}{cc}\Gamma \vdash x \Longrightarrow \texttt{Nat} & \Gamma \vdash t_0 \Longleftarrow A \\ \Gamma' = (\Gamma, y : \texttt{Nat}) & \Gamma' \vdash t_1 \Longleftarrow A\end{array}}{\Gamma \vdash (\texttt{match}\ x\ \texttt{with} \mid 0 \to t_0 \mid y + 1 \to t_1) \Longleftarrow A}$$

Rules: Checking

$$\boxed{\Gamma \vdash t \Longleftarrow A}$$

$$\frac{\ell \equiv 0}{\Gamma \vdash \texttt{nil} \Longleftarrow \texttt{Vec}\ \ell}$$

$$\frac{\Gamma \vdash x \Longrightarrow \texttt{Nat} \quad \Gamma \vdash t_0 \Longleftarrow A \quad \Gamma' = (\Gamma, y : \texttt{Nat}) \quad \Gamma' \vdash t_1 \Longleftarrow A}{\Gamma \vdash (\texttt{match}\ x\ \texttt{with} \mid 0 \to t_0 \mid y + 1 \to t_1) \Longleftarrow A}$$

Rules: Checking

$$\boxed{\Gamma \vdash t \Longleftarrow A}$$

$$\frac{\ell \equiv 0}{\Gamma \vdash \texttt{nil} \Longleftarrow \texttt{Vec}\ \ell}$$

$$\frac{\Gamma \vdash x \Longrightarrow \texttt{Nat} \quad \Gamma \vdash t_0 \Longleftarrow A \qquad \Gamma' = (\Gamma, y : \texttt{Nat}) \quad \Gamma' \vdash t_1 \Longleftarrow A}{\Gamma \vdash (\texttt{match}\ x\ \texttt{with} \mid 0 \to t_0 \mid y + 1 \to t_1) \Longleftarrow A}$$

Rules: Checking

$$\boxed{\Gamma \vdash t \Longleftarrow A}$$

$$\frac{\ell \equiv 0}{\Gamma \vdash \texttt{nil} \Longleftarrow \texttt{Vec}\ \ell}$$

$$\frac{\begin{array}{c}\Gamma \vdash x \Longrightarrow \texttt{Nat} \quad \Gamma \vdash t_0 \Longleftarrow A \\ \Gamma' = (\Gamma, y : \texttt{Nat}) \quad \Gamma' \vdash t_1 \Longleftarrow A\end{array}}{\Gamma \vdash (\texttt{match}\ x\ \texttt{with} \mid 0 \to t_0 \mid y + 1 \to t_1) \Longleftarrow A}$$

Rules: Checking

$$\boxed{\Gamma \vdash t \Longleftarrow A}$$

$$\frac{\ell \equiv 0}{\Gamma \vdash \texttt{nil} \Longleftarrow \texttt{Vec}\ \ell}$$

$$\frac{\Gamma \vdash x \Longrightarrow \texttt{Nat} \quad [0/x]\Gamma \vdash [0/x]t_0 \Longleftarrow [0/x]A \qquad \Gamma' = (\Gamma, y : \texttt{Nat}) \quad \Gamma' \vdash t_1 \Longleftarrow A}{\Gamma \vdash (\texttt{match}\ x\ \texttt{with} \mid 0 \to t_0 \mid y + 1 \to t_1) \Longleftarrow A}$$

Rules: Checking

$$\boxed{\Gamma \vdash t \Longleftarrow A}$$

$$\frac{\ell \equiv 0}{\Gamma \vdash \texttt{nil} \Longleftarrow \texttt{Vec}\ \ell}$$

$$\frac{\begin{array}{c}\Gamma \vdash x \Longrightarrow \texttt{Nat} \quad [0/x]\Gamma \vdash [0/x]t_0 \Longleftarrow [0/x]A \\ \Gamma' = (\Gamma, y : \texttt{Nat}) \quad [y+1/x]\Gamma' \vdash [y+1/x]t_1 \Longleftarrow [y+1/x]A\end{array}}{\Gamma \vdash (\texttt{match}\ x\ \texttt{with} \mid 0 \to t_0 \mid y+1 \to t_1) \Longleftarrow A}$$

Rules: Checking

$$\frac{\Gamma \vdash x \Longrightarrow \texttt{Nat} \quad [0/x]\Gamma \vdash [0/x]t_0 \Longleftarrow [0/x]A \qquad \Gamma' = (\Gamma, y : \texttt{Nat}) \quad [y+1/x]\Gamma' \vdash [y+1/x]t_1 \Longleftarrow [y+1/x]A}{\Gamma \vdash (\texttt{match}\ x\ \texttt{with} \mid 0 \to t_0 \mid y+1 \to t_1) \Longleftarrow A}$$

$$
\begin{aligned}
&\texttt{drop} \ : \ \Pi(k : \texttt{Nat}).\Pi(n : \texttt{Nat}).\texttt{Vec}\ (k+n) \to \texttt{Vec}\ n = \\
&\quad \texttt{fix drop}.\lambda k.\lambda n.\lambda v. \\
&\qquad \texttt{match}\ k\ \texttt{with} \\
&\qquad \mid 0 \to v \qquad v : \texttt{Vec}\ (0+n) \\
&\qquad \mid k'+1 \to \texttt{drop}\ k'\ n\ (\texttt{tail}\ (k'+n)\ v)
\end{aligned}
$$

$v : \texttt{Vec}\ (k'+1+n), \texttt{tail} \ldots : \texttt{Vec}\ (k'+n)$

Rules: Equality

$$\boxed{A_1 \equiv A_2}$$

$$\begin{cases} \texttt{Nat} \equiv \texttt{Nat} & \\ \Pi(x\colon A_1).B_1 \equiv \Pi(x\colon A_2).B_2 & \text{if } A_1 \equiv A_2 \text{ and } B_1 \equiv B_2 \\ \texttt{Vec}\ \ell_1 \equiv \texttt{Vec}\ \ell_2 & \text{if } \ell_1 \equiv \ell_2 \end{cases}$$

$$\boxed{\ell_1 \equiv \ell_2}$$

- Normalize and compare syntactically.
  - $1 + n + (2 + m) \longrightarrow 3 + n + m$
  - $(n + 0) + 3 + m \longrightarrow 3 + n + m$

Rules: Equality

$$\boxed{A_1 \equiv A_2}$$

$$\begin{cases} \texttt{Nat} \equiv \texttt{Nat} & \\ \Pi(x\colon A_1).B_1 \equiv \Pi(x\colon A_2).B_2 & \text{if } A_1 \equiv A_2 \text{ and } B_1 \equiv B_2 \\ \texttt{Vec}\ \ell_1 \equiv \texttt{Vec}\ \ell_2 & \text{if } \ell_1 \equiv \ell_2 \end{cases}$$

$$\boxed{\ell_1 \equiv \ell_2}$$

- Normalize and compare syntactically.
  - $1 + n + (2 + m) \longrightarrow 3 + n + m$
  - $(n + 0) + 3 + m \longrightarrow 3 + n + m$

Outline

An Unfortunate Restriction

- Subject of `match` must be a variable x or synthesize `Vec` x
    - Needed for substitution
- This is why `head` and `tail` must be built-in!

$$
\begin{aligned}
&\texttt{tail} \;:\; \Pi(n : \texttt{Nat}).\texttt{Vec}\,(n+1) \to \texttt{Vec}\; n = \\
&\quad \lambda n.\lambda v. \\
&\qquad \texttt{match}\; {\color{red} v}\; \texttt{with} \qquad {\color{red} v : \texttt{Vec}\,(n+1)} \\
&\qquad \mid \texttt{cons}\; n'\; x\; v' \to v'
\end{aligned}
$$

A Better Approach

$$\texttt{tail} \;:\; \Pi(n\colon \texttt{Nat}).\texttt{Vec}\,(n+1) \to \texttt{Vec}\;n =$$
$$\quad \lambda n.\lambda v.$$
$$\qquad \texttt{match}\; v\; \texttt{with}$$
$$\qquad \mid \texttt{cons}\; n'\; \_\; v' \to v'$$

A Better Approach

$$\texttt{tail} \; : \; \Pi(n\colon \texttt{Nat}).\texttt{Vec}\;(n+1) \to \texttt{Vec}\;n =$$
$$\lambda n.\lambda v.$$
$$\texttt{match}\; v \;\texttt{with}$$
$$\mid \texttt{cons}\; n' \;\_\; v' \to v'$$

Is the `cons` branch correct? (i.e., $\texttt{Vec}\;n \equiv \texttt{Vec}\;n'$?)

$$\forall n, n' \in \mathbb{N}.(n'+1 = n+1) \implies n = n' \qquad \checkmark$$

A Better Approach

$$\texttt{tail} \ : \ \Pi(n\colon \texttt{Nat}).\texttt{Vec}\ (n+1) \to \texttt{Vec}\ n =$$
$$\lambda n.\lambda v.$$
$$\texttt{match}\ v\ \texttt{with}$$
$$\mid \texttt{cons}\ n'\ \_\ v' \to v'$$

Is the cons branch correct? (i.e., $\texttt{Vec}\ n \equiv \texttt{Vec}\ n'$?)

$$\forall n, n' \in \mathbb{N}.(n'+1 = n+1) \implies n = n' \qquad \checkmark$$

Is the nil branch unreachable? (i.e., $\texttt{Vec}\ (n+1) \not\equiv \texttt{Vec}\ 0$?)

$$\neg\left(\exists n \in \mathbb{N}.n+1 = 0\right) \qquad \checkmark$$

In General

- Add a new context $\Delta ::= \cdot \mid \Delta, \ell_1 = \ell_2$
  - When entering a `match`, extend Δ
  - $\ell \equiv \ell'$: make a $\forall$ proposition
  - Reachability : make a $\exists$ proposition

$$\boxed{\Gamma | \Delta \vdash s \Longrightarrow A} \quad \boxed{\Gamma | \Delta \vdash t \Longleftarrow A}$$

$$\boxed{\Delta \vdash A_1 \equiv A_2} \quad \boxed{\Delta \vdash \ell_1 \equiv \ell_2}$$

- To decide whether propositions are true, use Z3 solver [2]
- Propositions in Presburger arithmetic are always decidable! [3]

[2] https://github.com/Z3Prover/z3

[3] Presburger 1929

In General

- Add a new context $\Delta ::= \cdot \mid \Delta, \ell_1 = \ell_2$
  - When entering a `match`, extend Δ
  - $\ell \equiv \ell'$: make a $\forall$ proposition
  - Reachability : make a $\exists$ proposition

$$\boxed{\Gamma|\Delta \vdash s \Longrightarrow A} \quad \boxed{\Gamma|\Delta \vdash t \Longleftarrow A}$$

$$\boxed{\Delta \vdash A_1 \equiv A_2} \quad \boxed{\Delta \vdash \ell_1 \equiv \ell_2}$$

- To decide whether propositions are true, use Z3 solver [2]
- Propositions in Presburger arithmetic are always decidable! [3]

[2] https://github.com/Z3Prover/z3

[3] Presburger 1929

Outline

Booleans

- Updated syntax:

$$\begin{array}{llcl} \text{Type} & A & ::= & \ldots \mid \texttt{Bool} \\ \text{Chk} & t & ::= & \ldots \\ & & \mid & \texttt{true} \mid \texttt{false} \\ & & \mid & (\texttt{match}\ s\ \texttt{with} \mid \texttt{true} \to t_1 \mid \texttt{false} \to t_2) \end{array}$$

- New typing rules:

$$\frac{}{\Gamma|\Delta \vdash \texttt{true} \Longleftarrow \texttt{Bool}} \qquad \frac{}{\Gamma|\Delta \vdash \texttt{false} \Longleftarrow \texttt{Bool}}$$

$$\frac{\Gamma|\Delta \vdash s \Longrightarrow \texttt{Bool} \quad \Gamma|\Delta \vdash t_1 \Longleftarrow A \quad \Gamma|\Delta \vdash t_2 \Longleftarrow A}{\Gamma|\Delta \vdash (\texttt{match}\ s\ \texttt{with} \mid \texttt{true} \to t_1 \mid \texttt{false} \to t_2) \Longleftarrow A}$$

- New case for type equality : $\Delta \vdash \texttt{Bool} \equiv \texttt{Bool}$
- All pretty straightforward!

Booleans

- Updated syntax:

$$
\begin{array}{llll}
\text{Type} & A & ::= & \ldots \mid \texttt{Bool} \\
\text{Chk} & t & ::= & \ldots \\
 & & \mid & \texttt{true} \mid \texttt{false} \\
 & & \mid & (\texttt{match}\ s\ \texttt{with} \mid \texttt{true} \rightarrow t_1 \mid \texttt{false} \rightarrow t_2)
\end{array}
$$

- New typing rules:

$$
\frac{}{\Gamma|\Delta \vdash \texttt{true} \Longleftarrow \texttt{Bool}} \qquad \frac{}{\Gamma|\Delta \vdash \texttt{false} \Longleftarrow \texttt{Bool}}
$$

$$
\frac{\Gamma|\Delta \vdash s \Longrightarrow \texttt{Bool} \quad \Gamma|\Delta \vdash t_1 \Longleftarrow A \quad \Gamma|\Delta \vdash t_2 \Longleftarrow A}{\Gamma|\Delta \vdash (\texttt{match}\ s\ \texttt{with} \mid \texttt{true} \rightarrow t_1 \mid \texttt{false} \rightarrow t_2) \Longleftarrow A}
$$

- New case for type equality : $\Delta \vdash \texttt{Bool} \equiv \texttt{Bool}$
- All pretty straightforward!

Outline

Limitations

- Vectors can only contain natural numbers
  - Can add polymorphism (using Hindley-Milner)
- Impossible to implement `filter`—what should the output length be?
  - Can do this using *dependent pairs*
- Explicit parameters are not ergonomic
  - Could add implicit parameters (as in Agda)
- Only addition allowed in lengths
  - Cannot implement `split`, `join`, etc.

Conclusion

- You can check simple constraints on list sizes using dependent types!
  - Enforce constraints at compile time
  - Catch some (but obviously not all) common typos
- How to check equivalence of lengths? How to prevent infinite loops in type checker?
  - Presburger arithmetic + Z3

Polymorphism

We implement the Hindley-Milner algorithm.

Type scheme	$\hat{A}$	$::=$	...	$\mid \forall \alpha_1, \alpha_2, \ldots, \alpha_N . A$
Type	A	$::=$	...	$\mid \alpha$
Context	Γ	$::=$	$\cdot$	$\mid \Gamma, x : \hat{A}$
Type subst.	σ	$::=$	$\cdot$	$\mid \sigma, A/\alpha$

$$\frac{\Gamma(x) = \hat{A} \quad \Gamma \vdash \hat{A} \sqsubseteq A}{\Gamma \vdash x \Longrightarrow A} \qquad \frac{\Gamma, x : \forall \_ . A \vdash t \Longleftarrow B}{\Gamma \vdash \lambda x . t \Longleftarrow \Pi(x : A) . B}$$

- Generalization: $\alpha \to \alpha$ generalizes to $\forall \alpha . \alpha \to \alpha$.
- Instantiation: $\forall \alpha . \alpha \to \alpha$ can be instantiated to $\mathtt{Nat} \to \mathtt{Nat}$.
- Unification: $\mathrm{unify}(\alpha \to \alpha, \mathtt{Nat} \to \mathtt{Nat}) = \{\mathtt{Nat}/\alpha\}$

Polymorphism

We implement the Hindley-Milner algorithm.

$$
\begin{array}{llcll}
\text{Type scheme} & \hat{A} & ::= & \ldots & \mid \forall \alpha_1, \alpha_2, \ldots, \alpha_N . A \\
\text{Type} & A & ::= & \ldots & \mid \alpha \\
\text{Context} & \Gamma & ::= & \cdot & \mid \Gamma, x \colon \hat{A} \\
\text{Type subst.} & \sigma & ::= & \cdot & \mid \sigma, A/\alpha
\end{array}
$$

$$
\frac{\Gamma(x) = \hat{A} \quad \Gamma \vdash \hat{A} \sqsubseteq A}{\Gamma \vdash x \Longrightarrow A}
\qquad
\frac{\Gamma, x : \forall \_ . A \vdash t \Longleftarrow B}{\Gamma \vdash \lambda x . t \Longleftarrow \Pi(x \colon A).B}
$$

- Generalization: $\alpha \to \alpha$ generalizes to $\forall \alpha . \alpha \to \alpha$.
- Instantiation: $\forall \alpha . \alpha \to \alpha$ can be instantiated to $\mathtt{Nat} \to \mathtt{Nat}$.
- Unification: $\text{unify}(\alpha \to \alpha, \mathtt{Nat} \to \mathtt{Nat}) = \{\mathtt{Nat}/\alpha\}$