An introduction to recursion and induction
A recursive datatype: toy lists

datatype 'a list = Nil | Cons 'a ''a list
A recursive datatype: toy lists

datatype 'a list = Nil | Cons 'a "’a list"

Nil: empty list
Cons x xs: head x :: ’a, tail xs :: ’a list
A recursive datatype: toy lists

datatype 'a list = Nil | Cons 'a ""a list"

Nil: empty list
Cons x xs: head x :: 'a, tail xs :: 'a list
A toy list: Cons False (Cons True Nil)
A recursive datatype: toy lists

datatype 'a list = Nil | Cons 'a ''a list

Nil: empty list

Cons x xs: head x :: 'a, tail xs :: 'a list

A toy list: Cons False (Cons True Nil)

Predefined lists: [False, True]
Concrete syntax

In .thy files:
Types and formulae need to be inclosed in "..."
Concrete syntax

In `.thy` files:
Types and formulae need to be inclosed in "..."

Except for single identifiers, e.g. ‘a
Concrete syntax

In .thy files:
Types and formulae need to be inclosed in "..."

Except for single identifiers, e.g. ’a

"..." normally not shown on slides
Structural induction on lists

$P \, xs$ holds for all lists $xs$ if
Structural induction on lists

$P \; xs$ holds for all lists $xs$ if

- $P \; Nil$
Structural induction on lists

\( P \, xs \) holds for all lists \( xs \) if

- \( P \, Nil \)
- and for arbitrary \( x \) and \( xs \), \( P \, xs \) implies \( P \, (\text{Cons} \, x \, xs) \)
A recursive function: append

Definition by *primitive recursion*:

\[
\text{primrec } \text{app} :: \text{'a list } \Rightarrow \text{'a list } \Rightarrow \text{'a list where}
\]
\[
\text{app } \text{Nil } \text{ys} = ? \mid
\text{app } (\text{Cons } x \text{ xs}) \text{ ys} = ??
\]
A recursive function: append

Definition by \textit{primitive recursion}:

\begin{verbatim}
primrec app :: 'a list ⇒ 'a list ⇒ 'a list where
app Nil ys = ? |
app (Cons x xs) ys = ??
\end{verbatim}

1 rule per constructor

Recursive calls must drop the constructor \iff Termination
Demo: append and reverse
General schema:

```plaintext
lemma name : "..."
apply ( ... )
apply ( ... )
:
done
```

If the lemma is suitable as a simplification rule:

```plaintext
lemma name [simp] : "..."
```
Proof methods

• Structural induction
  • Format: \((\text{induct } x)\)
    \(x\) must be a free variable in the first subgoal.
    The type of \(x\) must be a datatype.
  • Effect: generates 1 new subgoal per constructor

• Simplification and a bit of logic
  • Format: \(\text{auto}\)
  • Effect: tries to solve as many subgoals as possible using simplification and basic logical reasoning.
Top down proofs

“completes” any proof.

Suitable for top down developments:
Assume lemmas first, prove them later.
Disproving

quickcheck

tries to find counterexample by random testing