Inductive-inductive definitions

Fredrik Nordvall Forsberg

Department of Computer Science
Swansea University
csfnf@swansea.ac.uk

BCTCS April 8, 2010

Joint work with Anton Setzer

Induction principles in Martin-Löf type theory

This talk is about a new induction principle in Martin-Löf type theory, which we call *induction-induction*, or *inductive-inductive definitions*.

- What is type theory?
- What is an induction principle?
- What is induction-induction?

What is type theory?

Martin-Löf type theory

- Martin-Löf type theory is a framework for constructive mathematics.
- Alternatively, a dependently typed functional programming language (cf. Agda).
- If you are scared, think of it like set theory:
  - $x : A$ means x is an element of type A.
    - $17 : \mathbb{N}$
    - length : String $\to \mathbb{N}$
  - We can form Cartesian products $(A \times B)$, disjoint unions $(A + B)$, function spaces $(A \to B)$ etc

Martin-Löf type theory (cont.)

- In addition, we also have dependent types, i.e. types that depend on values.
  - $\mathrm{Vec}(n)$, the type of lists of length n (vectors)
  - $\mathrm{Mat}(k, \ell)$, the type of $k \times \ell$ matrices
  - $\mathrm{Even}(n)$, the type of proofs that n is an even number (empty if n is odd)
- To make full use of this, we have dependent function spaces $(x : A) \to B(x)$ and dependent products $(x : A) \times B(x)$.
  - $f : (x : A) \to B(x)$ is a function f such that $f(a) : B(a)$ for every $a : A$.
  - $p : (x : A) \times B(x)$ is a pair $\langle a, b \rangle$ such that $a : A$ and $b : B(a)$.
- Examples:
  - $\mathrm{tail} : (n : \mathbb{N}) \to \mathrm{Vec}(n + 1) \to \mathrm{Vec}(n)$
  - $\mathrm{matMult} : (k, \ell, m : \mathbb{N}) \to \mathrm{Mat}(k, \ell) \to \mathrm{Mat}(\ell, m) \to \mathrm{Mat}(k, m)$
  - $\langle 2, \mathrm{twoIsEven} \rangle : (n : \mathbb{N}) \times \mathrm{Even}(n)$

Propositions as types

- Propositions can be interpreted as types in a (for a constructivist) natural way
  - To interpret $(\exists x \in A)P(x)$, use $(x : A) \times P(x)$.
  - To interpret $(\forall x \in A)P(x)$, use $(x : A) \to P(x)$.
- Under this interpretation, $x : A$ means "x is a proof of A".
  - $17 : \mathbb{N}$ – 17 is a natural number (has type $\mathbb{N}$).
  - twoIsEven : Even(2) – twoIsEven is a proof that two is an even number.

What is an induction principle?

Induction is everywhere!

Let us first consider everyone's favourite inductive set: the natural numbers $\mathbb{N}$

$$\frac{}{0 : \mathbb{N}} \qquad \frac{n : \mathbb{N}}{\mathsf{succ}(n) : \mathbb{N}}$$

In other words, we have constructors

$$0 : \mathbb{N} \ , \quad \mathsf{succ} : \mathbb{N} \to \mathbb{N}.$$

The induction principle: if the constructor is *strictly positive*, i.e. the set to be defined does not appear to the left of an arrow in the domain ($c : (D \to D) \to D$ is not allowed), it defines an inductive set.

We are allowed to define functions from inductive types by structural recursion.

Indexed inductive definitions

We can generalise this to *indexed inductive definitions*, which lets us define a whole family $U : I \to \text{Set}$ of inductive sets in one go.
A famous example are vectors $\text{Vec} : \mathbb{N} \to \text{Set}$, i.e. lists indexed by their length.

$$\frac{}{\epsilon : \text{Vec}(0)} \qquad \frac{x : A \quad xs : \text{Vec}(n)}{x :: xs : \text{Vec}(n+1)}$$

Note how we relate vectors with different indices, so that we cannot define this as separate inductive sets.
With this principle, we can construct mutually defined inductive sets.

$$\frac{}{\text{zeroIsEven} : \text{Even}(0)} \qquad \frac{p : \text{Even}(n)}{\text{sEven}(n, p) : \text{Odd}(n+1)} \qquad \frac{p : \text{Odd}(n)}{\text{sOdd}(n, p) : \text{Even}(n+1)}$$

Indexed inductive definitions

We can generalise this to *indexed inductive definitions*, which lets us define a whole family $U : I \to \text{Set}$ of inductive sets in one go.
A famous example are vectors $\text{Vec} : \mathbb{N} \to \text{Set}$, i.e. lists indexed by their length.

$$\frac{}{\epsilon : \text{Vec}(0)} \qquad \frac{x : A \quad xs : \text{Vec}(n)}{x :: xs : \text{Vec}(n+1)}$$

Note how we relate vectors with different indices, so that we cannot define this as separate inductive sets.
With this principle, we can construct mutually defined inductive sets.

$$\frac{}{\text{zeroIsEven} : \text{Even}(0)} \quad \frac{p : \text{Even}(n)}{\text{sEven}(n, p) : \text{Odd}(n+1)} \quad \frac{p : \text{Odd}(n)}{\text{sOdd}(n, p) : \text{Even}(n+1)}$$

(index set $\{\star_{\text{even}}, \star_{\text{odd}}\}$; replace Even with $U(\star_{\text{even}})$ and Odd with $U(\star_{\text{odd}})$)

Indexed inductive definitions

We can generalise this to *indexed inductive definitions*, which lets us define a whole family $U : I \to \text{Set}$ of inductive sets in one go.
A famous example are vectors $\text{Vec} : \mathbb{N} \to \text{Set}$, i.e. lists indexed by their length.

$$\frac{}{\epsilon : \text{Vec}(0)} \qquad \frac{x : A \quad xs : \text{Vec}(n)}{x :: xs : \text{Vec}(n+1)}$$

Note how we relate vectors with different indices, so that we cannot define this as separate inductive sets.
With this principle, we can construct mutually defined inductive sets.

$$\frac{}{\text{zeroIsEven} : U(\star_{\text{even}}, 0)} \qquad \frac{p : U(\star_{\text{even}}, n)}{\text{sEven}(n, p) : U(\star_{\text{odd}}, n+1)}$$

$$\frac{p : U(\star_{\text{even}}, n)}{\text{sOdd}(n, p) : U(\star_{\text{even}}, n+1)}$$

(index set $\{\star_{\text{even}}, \star_{\text{odd}}\}$; replace Even with $U(\star_{\text{even}})$ and Odd with $U(\star_{\text{odd}})$)

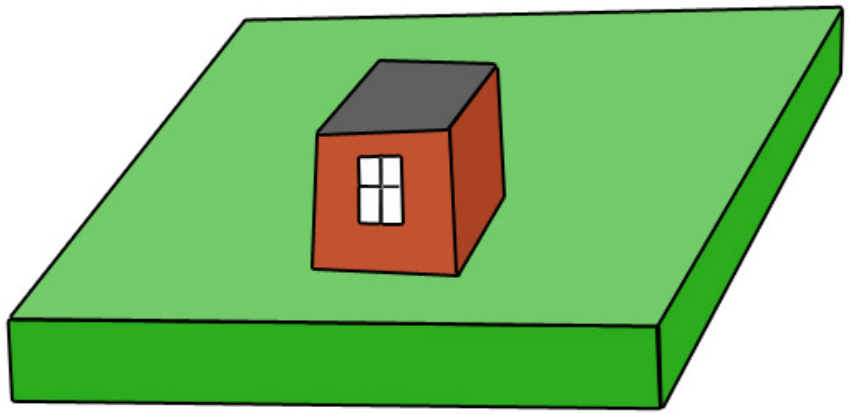
Induction-recursion

- Induction-recursion allows us to inductively define a set U, together with a recursive function $T : U \to D$ for some (large) type D (often $D = \mathrm{Set}$).
- The constructor for U might make use of T applied to elements from U, even in negative positions.
  - E.g. $\sigma : (x : U) \times (T(x) \to U) \to U$.
- As an example, consider the following inductive-recursive definition of highrises and their height:

$$\mathrm{Highrise} : \mathrm{Set} \qquad \mathrm{height} : \mathrm{Highrise} \to \mathbb{N}$$

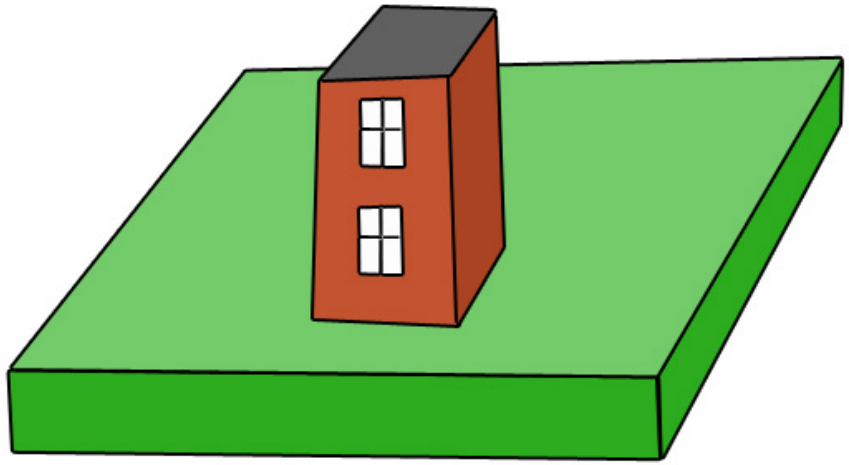
Inductive-recursive highrises

$$\frac{}{\text{oneFloor} : \text{Highrise}} \quad \text{height}(\text{oneFloor}) = 1$$



Inductive-recursive highrises

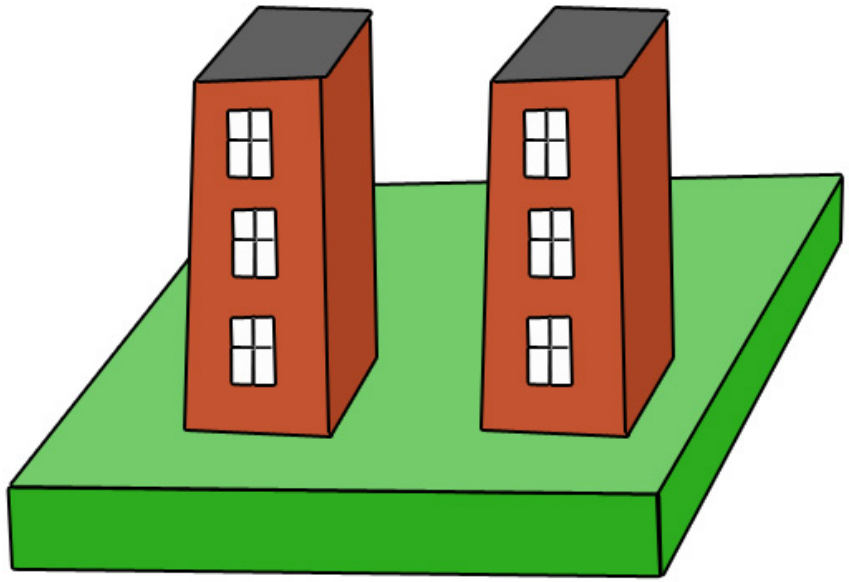
$$\frac{h : \text{Highrise}}{\text{addFloor}(h) : \text{Highrise}} \quad \text{height}(\text{addFloor}(h)) = \text{height}(h) + 1$$



Inductive-recursive highrises

$$\frac{h_1 : \text{Highrise} \quad h_2 : \text{Highrise} \quad p : \text{height}(h_1) == \text{height}(h_2)}{\text{archway}(h_1, h_2, p) : \text{Highrise}}$$

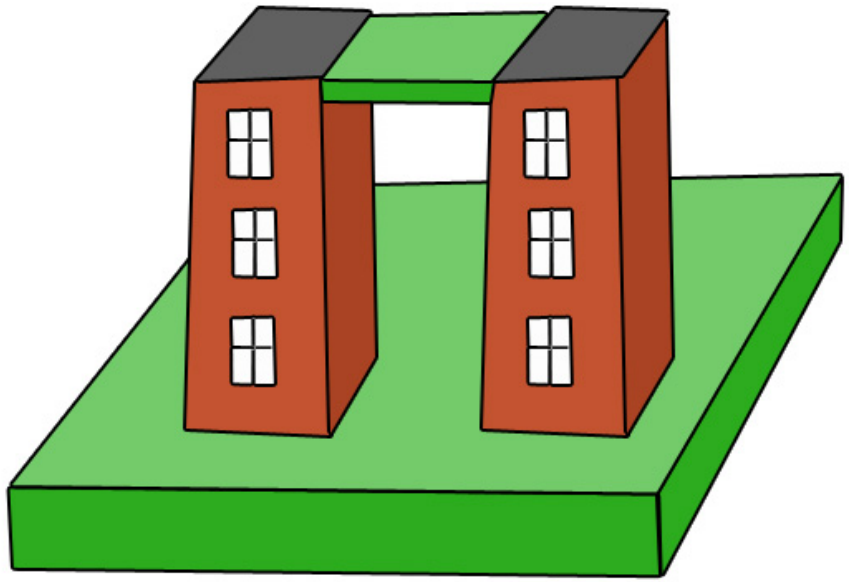
$$\text{height}(\text{archway}(h_1, h_2, p)) = \text{height}(h_1)$$



Inductive-recursive highrises

$$\frac{h_1 : \text{Highrise} \quad h_2 : \text{Highrise} \quad p : \text{height}(h_1) == \text{height}(h_2)}{\text{archway}(h_1, h_2, p) : \text{Highrise}}$$

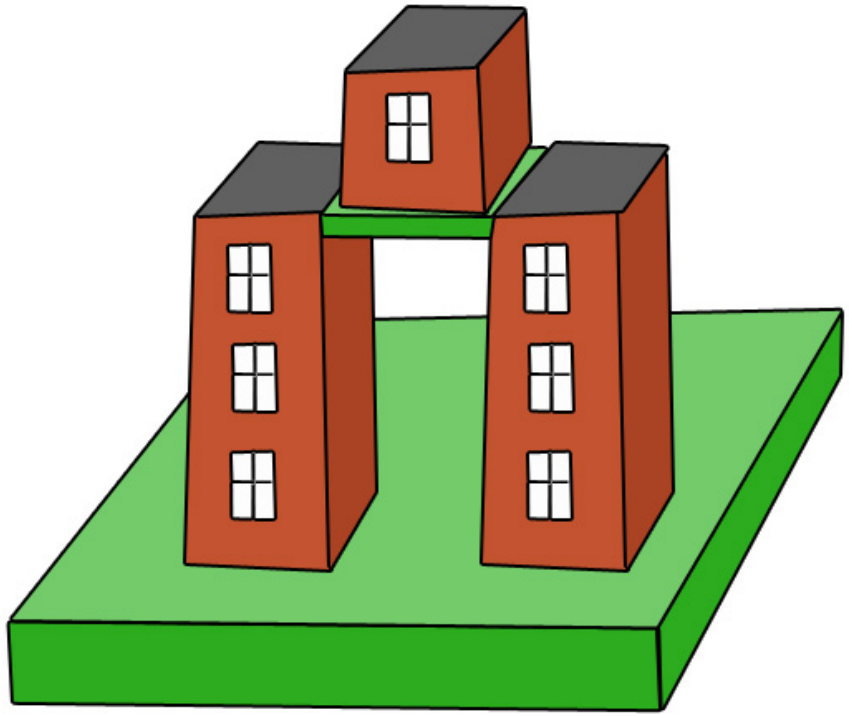
$$\text{height}(\text{archway}(h_1, h_2, p)) = \text{height}(h_1)$$



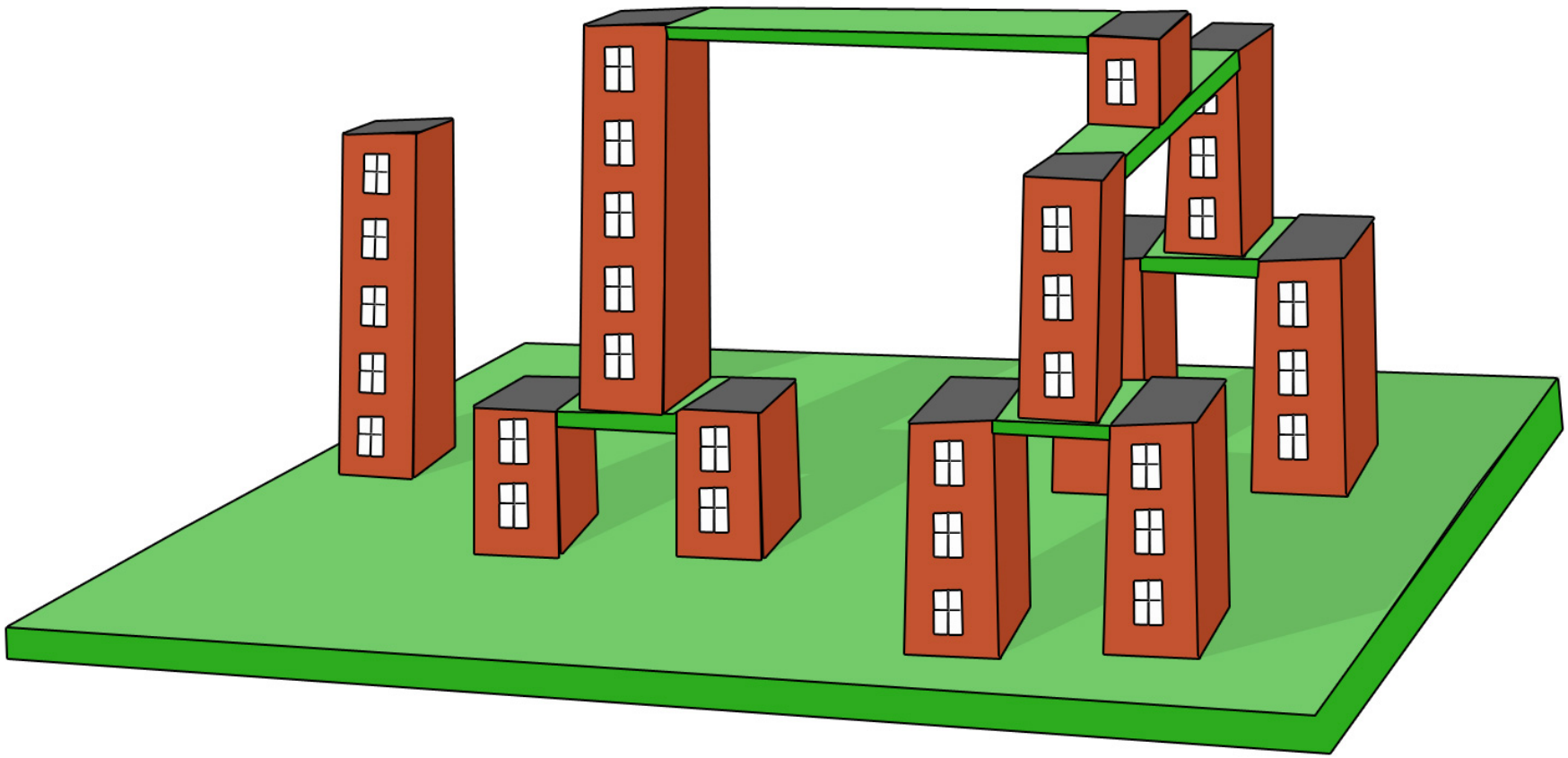
Inductive-recursive highrises

$$\frac{h_1 : \text{Highrise} \quad h_2 : \text{Highrise} \quad p : \text{height}(h_1) == \text{height}(h_2)}{\text{archway}(h_1, h_2, p) : \text{Highrise}}$$

$$\text{height}(\text{archway}(h_1, h_2, p)) = \text{height}(h_1)$$



Inductive-recursive highrises[1]



[1] Illustrations by Viktor Arnell

Induction principles

- Ordinary inductive definitions allow us to define one set inductively.
- Indexed inductive definitions allow us to define a family of sets $U : I \to \text{Set}$ inductively for any previously defined index set I.
- Induction-recursion allow us to define a set U inductively, and a function $T : U \to D$ recursively, for a previously defined (large) type D.
  - In particular, if $D = \text{Set}$, then we define $U : \text{Set}$ inductively and $T : U \to \text{Set}$ recursively.

Induction principles

- Ordinary inductive definitions allow us to define one set inductively.
- Indexed inductive definitions allow us to define a family of sets $U : I \to \text{Set}$ inductively for any previously defined index set I.
- Induction-recursion allow us to define a set U inductively, and a function $T : U \to D$ recursively, for a previously defined (large) type D.
  - In particular, if $D = \text{Set}$, then we define $U : \text{Set}$ inductively and $T : U \to \text{Set}$ recursively.

What if also $T : U \to \text{Set}$ is defined inductively?

What is induction-induction?

Induction-induction

- Induction-induction allows us to define

$$A : \mathrm{Set} \ , \qquad B : A \to \mathrm{Set}$$

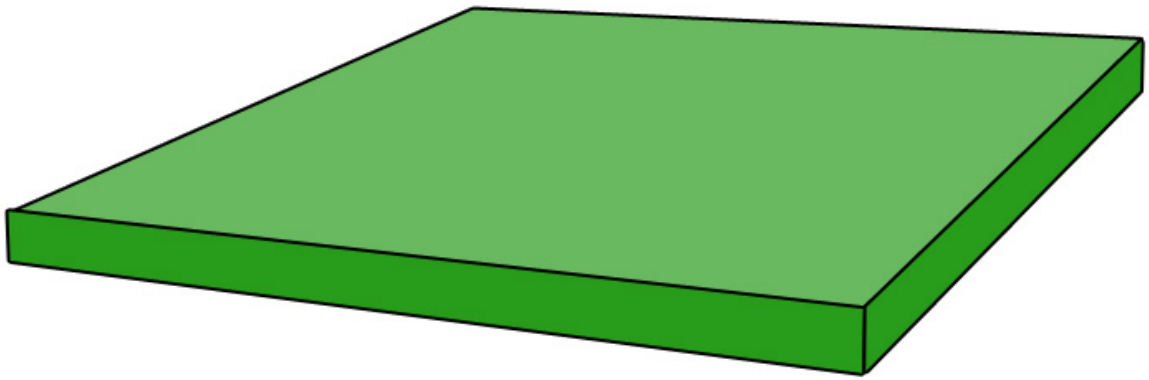
  where both A and $B(a)$ are defined inductively.
- The constructors for A can refer to B and vice versa.
- This is not an indexed inductive definition, since A is defined together with B, and not fixed beforehand.
- This is not an inductive-recursive definition, since B is inductively defined, not recursively.
- Example:

$$\mathrm{Platform} : \mathrm{Set} \ , \qquad \mathrm{Building} : \mathrm{Platform} \to \mathrm{Set}$$

  $p : \mathrm{Platform}$ means p is a platform. $b : \mathrm{Building}(p)$ means b is a building built on the platform p.

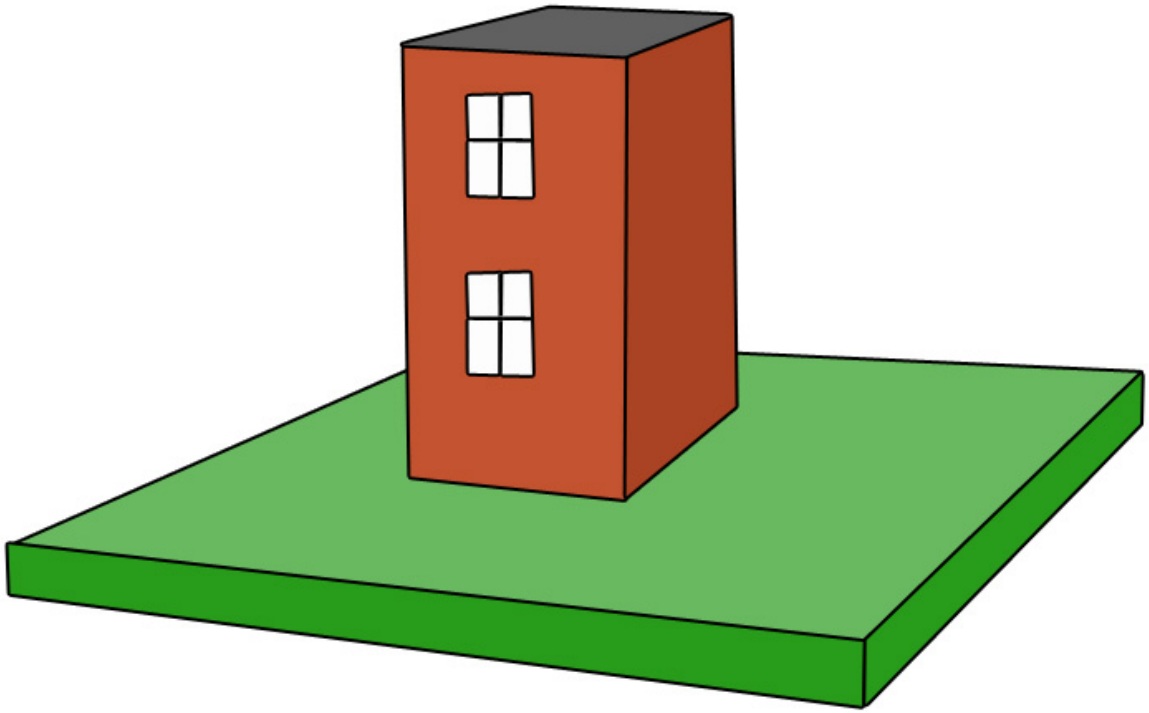
Example: buildings and platforms

$$\frac{}{\text{ground} : \text{Platform}}$$



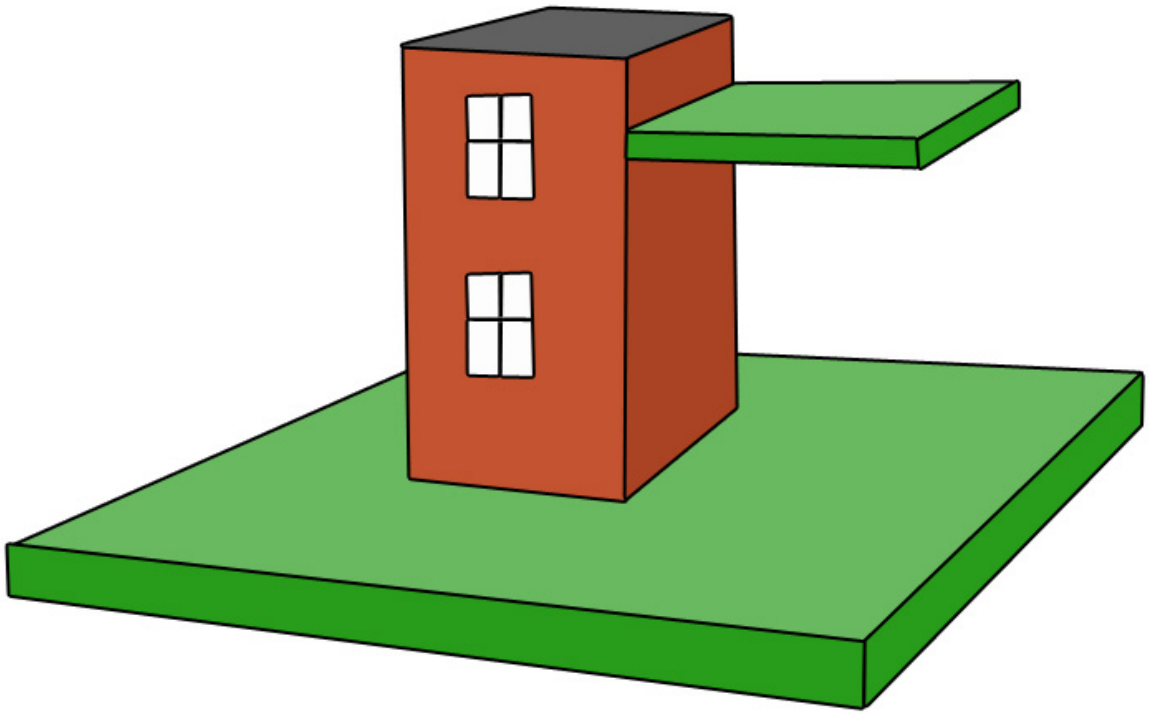
Example: buildings and platforms

$$\frac{p : \mathsf{Platform}}{\mathsf{onTop}(p) : \mathsf{Building}(p)}$$



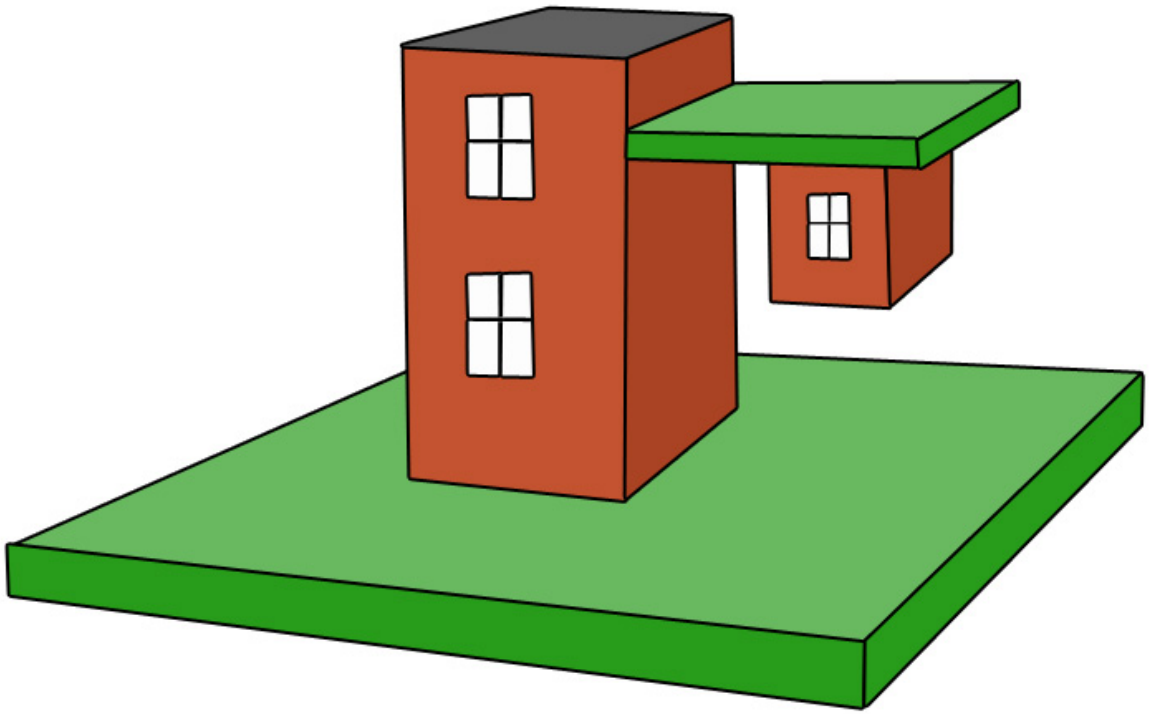
Example: buildings and platforms

$$\frac{p : \text{Platform} \quad b : \text{Building}(p)}{\text{extension}(p, b) : \text{Platform}}$$



Example: buildings and platforms

$$\frac{p : \mathsf{Platform} \quad b : \mathsf{Building}(p)}{\mathsf{hangingUnder}(p, b) : \mathsf{Building}(\mathsf{extension}(p, b))}$$



Note

Our examples for both induction-recursion and induction-induction have been rather silly.
However, the principles also have important uses.

- Induction-recursion can be used to define universes, which allows reflection in the theory.
- Induction-induction have been used informally to model type theory inside type theory (Danielsson, Chapman), but without justification of the principle.

An axiomatisation

An axiomatisation

- Induction-recursion was given a finite axiomatisation by Dybjer and Setzer.
- The main idea is to extend type theory with:
  - a datatype SP consisting of codes for sets defined by induction-recursion
  - a function Arg which "decodes" the code from SP, giving the domain of the constructor for the set.
  - for every $\gamma : \mathrm{SP}$, a set U_γ closed under $\mathrm{Arg}(\gamma)$, i.e. there is a constructor

$$\mathrm{intro}_\gamma : \mathrm{Arg}(\gamma, U_\gamma) \to U_\gamma$$

(omitting the recursively defined $T_\gamma : U_\gamma \to D$)

Extending the idea to induction-induction

Extending the technique used there, we can also give an axiomatisation for induction-induction.

- We will need two datatypes SP_A and SP_B, one for A and one for B.
- In addition, we will need to keep track of the elements we can refer to, so we will need parameters A_{ref} and B_{ref}.
  - For example, in

$$\mathrm{extension} : (p : \mathrm{Platform}) \times \mathrm{Building}(p) \to \mathrm{Platform} \ ,$$

    $\mathrm{Building}(p)$ refers to $p : \mathrm{Platform}$. After the first argument, A_{ref} will be extended to $A_{\mathrm{ref}} + \{p\}$.
- Also need machinery to allow constructors for B to use constructors for A.

$$\mathrm{hangingUnder} : (p : \mathrm{Platform}) \times (b : \mathrm{Building}(p)) \to \mathrm{Building}(\mathrm{extension}(p, b))$$

Consistency of the theory

We show that the theory is consistent by creating a set-theoretic model.

Theorem (Soundness)

(i) *If* $\vdash \Gamma$ *context, then* $[\![\Gamma]\!] \downarrow$.

(ii) *If* $\Gamma \vdash A : E$, *then* $[\![\Gamma]\!] \downarrow$, *and for all* $\rho \in [\![\Gamma]\!]$, $[\![A]\!]_\rho \in [\![E]\!]_\rho$.

(iii) *If* $\Gamma \vdash A = B : E$, *then* $[\![\Gamma]\!] \downarrow$, *and for all* $\rho \in [\![\Gamma]\!]$, $[\![A]\!]_\rho = [\![B]\!]_\rho$, *and* $[\![A]\!]_\rho \in [\![E]\!]_\rho$.

(iv) $\not\vdash a : \emptyset$. □

Summary

Summary: what is induction-induction?

- We have introduced *induction-induction*, an induction principle in Martin-Löf type theory.
- Allows simultaneous definition of a set A and a A-indexed set B, i.e. $B(a)$ is a set for every $a : A$.
- Both A and B are defined inductively, and the constructor for A can refer to B and vice versa.

More details at `http://cs.swansea.ac.uk/~csfnf`.

Thanks!