UNSW SYDNEY

COMP4161
Advanced Topics in Software Verification

Gerwin Klein, June Andronick, Miki Tanaka, Johannes Åman Pohjola

T3/2022

Content

➜ Foundations & Principles

- Intro, Lambda [illegible] atural deduction [1,2]
- Higher Order [illegible] part 1) [2,3[a]]
- Term rewriting [3,4]

➜ Proof & Specification Techniques

- Inductively defined sets, rule induction [4,5]
- Datatype induction, primitive recursion [5,7]
- General recursive functions, termination proofs [7[b]]
- Proof automation, Isar (part 2) [8]
- Hoare logic, proofs about programs, invariants [8,9]
- C verification [9,10]
- Practice, questions, exam prep [10[c]]

[a] a1 due; [b] a2 due; [c] a3 due

Last Time

- Conditional term re

Last Time

- Conditional term re
- Case Splitting with er

Last Time

- ➜ Conditional term re
- ➜ Case Splitting with er
- ➜ Congruence rules

Last Time

- ➜ Conditional term re
- ➜ Case Splitting with er
- ➜ Congruence rules
- ➜ AC Rules

Last Time

- ➜ Conditional term re
- ➜ Case Splitting with er
- ➜ Congruence rules
- ➜ AC Rules
- ➜ Knuth-Bendix Completion (Waldmeister)

Last Time

- Conditional term re
- Case Splitting with er
- Congruence rules
- AC Rules
- Knuth-Bendix Completion (Waldmeister)
- Orthogonal Rewrite Systems

Specification Techniques

Sets

Sets in Isabelle

Type **'a set**: sets over type 'a

Sets in Isabelle

Type **'a set**: sets over type 'a

➜ $\{\}$, $\{e_1, \ldots, e_n\}$,

Sets in Isabelle

Type **'a set**: sets over type 'a

- $\{\}$, $\{e_1, \ldots, e_n\}$,
- $e \in A$, $A \subseteq B$

Sets in Isabelle

Type **'a set**: sets over type 'a

- $\{\}$, $\{e_1, \ldots, e_n\}$,
- $e \in A$, $A \subseteq B$
- $A \cup B$, $A \cap B$, A

Sets in Isabelle

Type **'a set**: sets over type 'a

- $\{\}$, $\{e_1, \ldots, e_n\}$,
- $e \in A$, $A \subseteq B$
- $A \cup B$, $A \cap B$, A
- $\bigcup x \in A.\ B\ x$, $\bigcap x \in A.\ B\ x$, $\bigcap A$, $\bigcup A$

Sets in Isabelle

Type **'a set**: sets over type 'a

- $\{\}$, $\{e_1, \ldots, e_n\}$,
- $e \in A$, $A \subseteq B$
- $A \cup B$, $A \cap B$, A
- $\bigcup x \in A.\ B\ x$, $\bigcap x \in A.\ B\ x$, $\bigcap A$, $\bigcup A$
- $\{i..j\}$

Sets in Isabelle

Type **'a set**: sets over type 'a

- $\{\}$, $\{e_1, \ldots, e_n\}$,
- $e \in A$, $A \subseteq B$
- $A \cup B$, $A \cap B$, A
- $\bigcup x \in A.\ B\ x$, $\bigcap x \in A.\ B\ x$, $\bigcap A$, $\bigcup A$
- $\{i..j\}$
- insert :: $\alpha \Rightarrow \alpha$ set $\Rightarrow \alpha$ set

Sets in Isabelle

Type **'a set**: sets over type 'a

- $\{\}$, $\{e_1, \ldots, e_n\}$,
- $e \in A$, $A \subseteq B$
- $A \cup B$, $A \cap B$, A
- $\bigcup x \in A.\ B\ x$, $\bigcap x \in A.\ B\ x$, $\bigcap A$, $\bigcup A$
- $\{i..j\}$
- insert :: $\alpha \Rightarrow \alpha$ set $\Rightarrow \alpha$ set
- $f`A \equiv \{y.\ \exists x \in A.\ y = f\ x\}$
- ...

Proofs about Sets

Natural deduction proc

- ➜ equalityI: $[\![A \subseteq B;$  $A = B$

Proofs about Sets

Natural deduction proc

- ➜ equalityI: $[\![A \subseteq B;$  $A = B$
- ➜ subsetI: $(\bigwedge x.\ x \in A \Longrightarrow x \in B) \Longrightarrow A \subseteq B$

Proofs about Sets

Natural deduction proc

- equalityI: $[\![A \subseteq B;$ $A = B$
- subsetI: $(\bigwedge x.\ x \in A$ $) \Longrightarrow A \subseteq B$
- . . . **find_theorems**

Bounded Quantifiers

- $\forall x \in A.\ P\ x$

Bounded Quantifiers

➜ $\forall x \in A.\ P\ x \equiv \forall x.$  $P\ x$

Bounded Quantifiers

- $\forall x \in A.\ P\ x \equiv \forall x.\ P\ x$
- $\exists x \in A.\ P\ x$

Bounded Quantifiers

- $\forall x \in A.\ P\ x \equiv \forall x.\ P\ x$
- $\exists x \in A.\ P\ x \equiv \exists x.\ x$

Bounded Quantifiers

- $\forall x \in A.\ P\ x \equiv \forall x.$  $P\ x$
- $\exists x \in A.\ P\ x \equiv \exists x.$ x
- ballI: $(\bigwedge x.\ x \in A \Longrightarrow$ $) \Longrightarrow \forall x \in A.\ P\ x$
- bspec: $[\![\forall x \in A.\ P\ x; x \in A]\!] \Longrightarrow P\ x$

Bounded Quantifiers

- $\forall x \in A.\ P\ x \equiv \forall x.$  $P\ x$
- $\exists x \in A.\ P\ x \equiv \exists x.$ x
- ballI: $(\bigwedge x.\ x \in A \Longrightarrow$ $) \Longrightarrow \forall x \in A.\ P\ x$
- bspec: $[\![\forall x \in A.\ P\ x; x \in A]\!] \Longrightarrow P\ x$
- bexI: $[\![P\ x; x \in A]\!] \Longrightarrow \exists x \in A.\ P\ x$
- bexE: $[\![\exists x \in A.\ P\ x; \bigwedge x.\ [\![x \in A; P\ x]\!] \Longrightarrow Q]\!] \Longrightarrow Q$

Demo

Sets

The Three Basic Ways of Introducing Theorems

➜ **Axioms**:

Example: **axiom** refl: "$t = t$" **where**

The Three Basic Ways of Introducing Theorems

➜ **Axioms**:

Example: **axior**  **where** refl: "$t = t$"

Do not use. Evil. your logic inconsistent.

The Three Basic Ways of Introducing Theorems

- **Axioms**:

  Example: **axiom** refl: "$t = t$" **where**

  Do not use. Evil. your logic inconsistent.

- **Definitions:**

  Example: **definition** inj **where** "inj $f \equiv \forall x\ y.\ f\ x = f\ y \longrightarrow x = y$"

The Three Basic Ways of Introducing Theorems

- **Axioms**:

  Example: **axiom** refl: "$t = t$" **where**

  Do not use. Evil. Makes your logic inconsistent.

- **Definitions:**

  Example: **definition** inj **where** "inj $f \equiv \forall x\ y.\ f\ x = f\ y \longrightarrow x = y$"

  Introduces a new lemma called inj_def

The Three Basic Ways of Introducing Theorems

➜ **Axioms**:

Example: **axiom** refl: "$t = t$" **where**

Do not use. Evil. your logic inconsistent.

➜ **Definitions:**

Example: **definition** inj **where** "inj $f \equiv \forall x\ y.\ f\ x = f\ y \longrightarrow x = y$"

Introduces a new lemma called inj_def

➜ **Proofs:**

Example: **lemma** "inj $(\lambda x.\ x + 1)$"

The Three Basic Ways of Introducing Theorems

➜ **Axioms**:

Example: **axior** **where** refl: "$t = t$"

Do not use. Evil. **your logic inconsistent.**

➜ **Definitions:**

Example: **definition** inj **where** "inj $f \equiv \forall x\ y.\ f\ x = f\ y \longrightarrow x = y$"

Introduces a new lemma called inj_def

➜ **Proofs:**

Example: **lemma** "inj $(\lambda x.\ x + 1)$"

The harder, but safe choice.

The Three Basic Ways of Introducing Types

- **typedecl**: by name

  Example: **ty** es

  Introduces new typ hout any further assumptions

The Three Basic Ways of Introducing Types

- **typedecl**: by name

  Example: **ty**es

  Introduces new typ hout any further assumptions

- **type_synonym**: by abbreviation

  Example: **type_synonym** α rel = "$\alpha \Rightarrow \alpha \Rightarrow bool$"

  Introduces abbreviation *rel* for existing type $\alpha \Rightarrow \alpha \Rightarrow bool$

  Type abbreviations are immediately expanded internally

The Three Basic Ways of Introducing Types

- **typedecl**: by name

  Example: **ty** es

  Introduces new typ hout any further assumptions

- **type_synonym**: by abbreviation

  Example: **type_synonym** α rel = "$\alpha \Rightarrow \alpha \Rightarrow$ *bool*"

  Introduces abbreviation *rel* for existing type $\alpha \Rightarrow \alpha \Rightarrow$ *bool*

  Type abbreviations are immediately expanded internally

- **typedef**: by definiton as a set

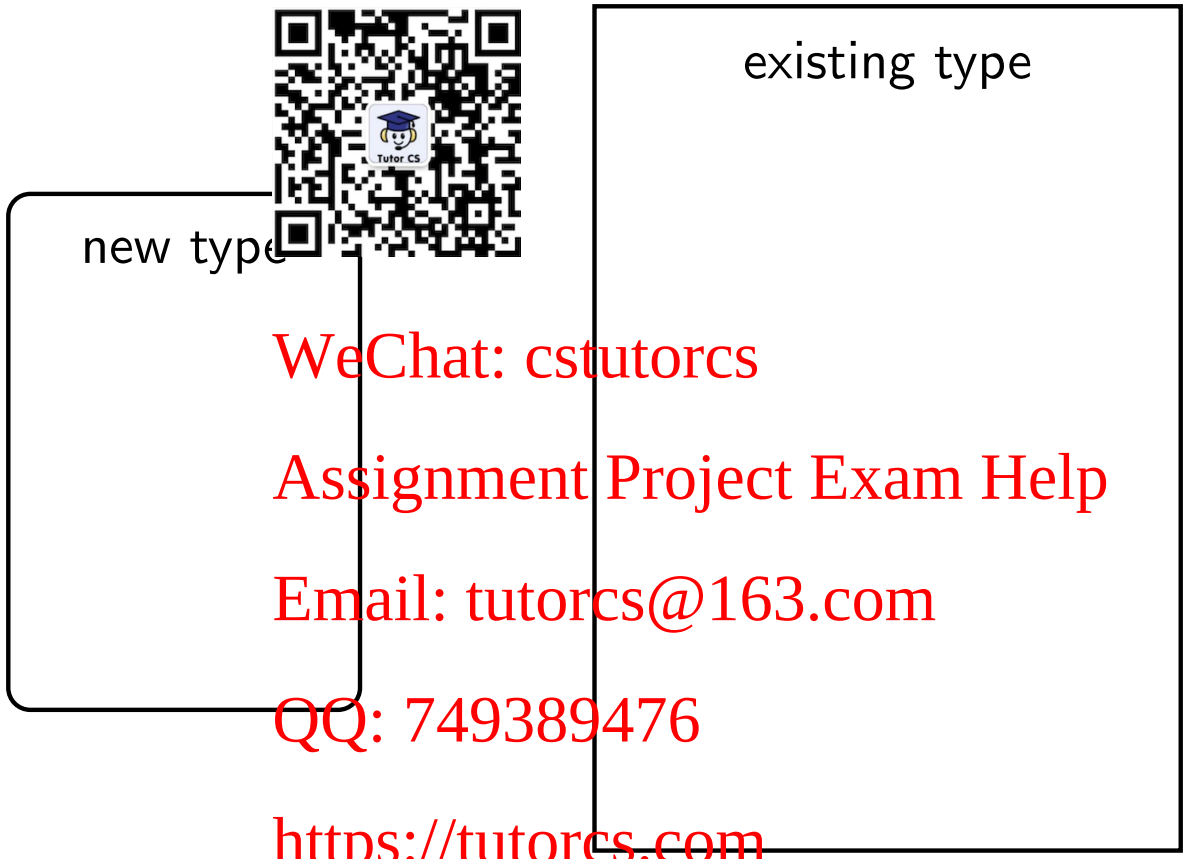
  Example: **typedef** new_type = "{some set}" <proof>

  Introduces a new type as a subset of an existing type.

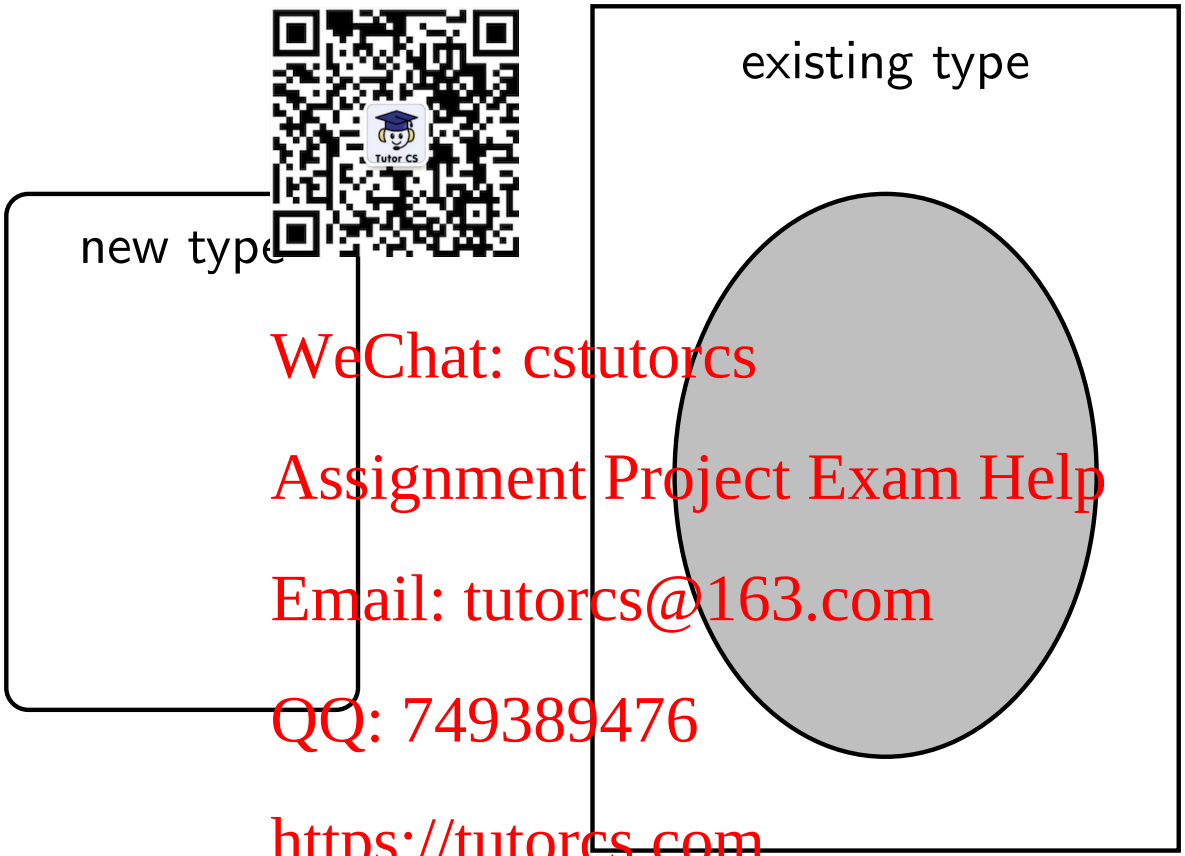
  The proof shows that the set on the rhs in non-empty.

How typedef works

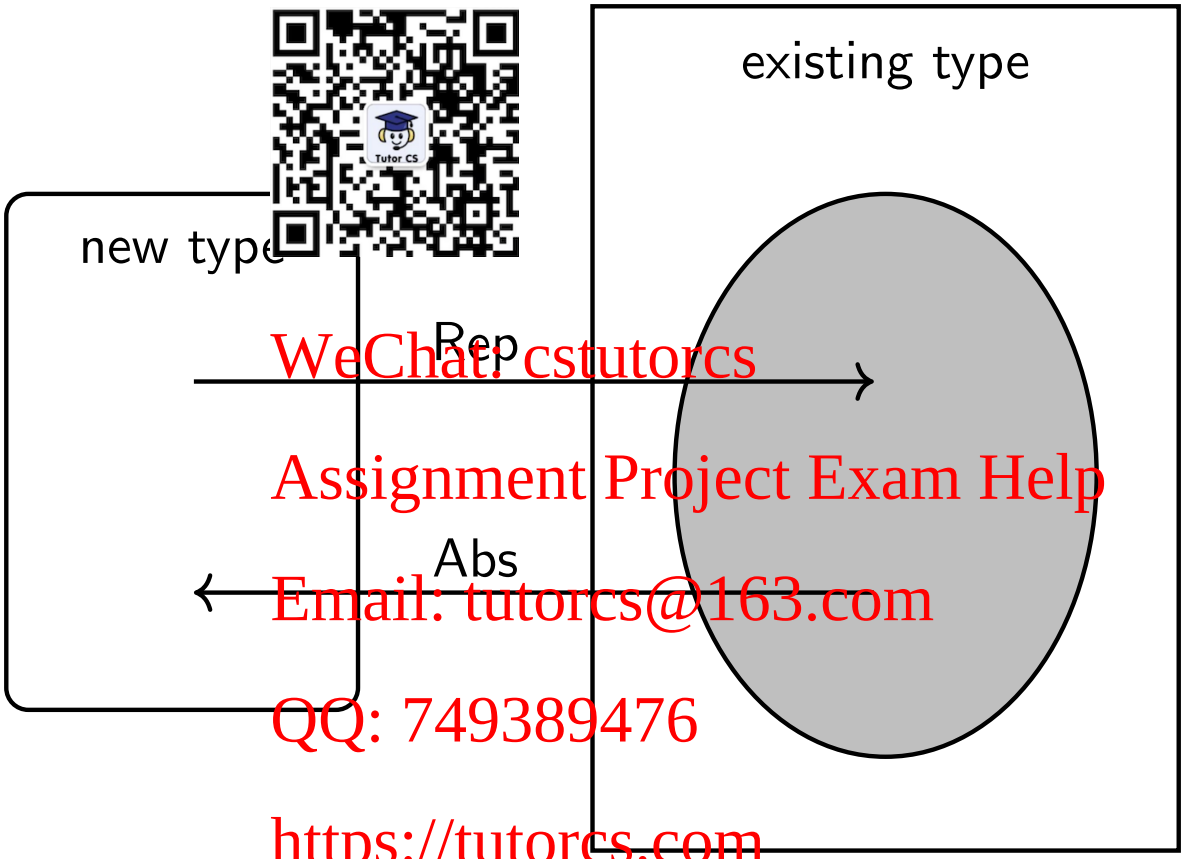
How typedef works



How typedef works



How typedef works



How typedef works

Example: Pairs

① Pick existing type: (α, β) Prod

Example: Pairs

(α, β) Prod

① Pick existing type: ool

② Identify subset:

Example: Pairs

(α, β) Prod

① Pick existing type: ool

② Identify subset:
(α, β) Prod $= \{f.\ \exists a\ b.\ f = (\lambda(x :: \alpha)\ (y :: \beta).\ x = a \wedge y = b\}$

③ We get from Isabelle:

Example: Pairs

(α, β) Prod

① Pick existing type: ool

② Identify subset:
(α, β) Prod $= \{f.\ \exists a\ b.\ f = (\lambda(x :: \alpha)\ (y :: \beta).\ x = a \wedge y = b\}$

③ We get from Isabelle:

- functions Abs_Prod, Rep_Prod
- both injective
- Abs_Prod (Rep_Prod x) = x

④ We now can:

Example: Pairs

(α, β) Prod

① Pick existing type: ool

② Identify subset:
(α, β) Prod $= \{f.\ \exists x\ y.\ f = \lambda(x :: \alpha)\ (y :: \beta).\ x = a \wedge y = b\}$

③ We get from Isabelle:

- functions Abs_Prod, Rep_Prod
- both injective
- Abs_Prod (Rep_Prod x) = x

④ We now can:

- define constants Pair, fst, snd in terms of Abs_Prod and Rep_Prod
- derive all characteristic theorems
- forget about Rep/Abs, use characteristic theorems instead

Demo

Introducing new Types

Inductive Definitions

Example

$$\frac{}{\langle \text{skip}, \sigma \rangle \longrightarrow \sigma} \qquad \frac{[\![e]\!]\sigma = v}{\langle x := e, \sigma \rangle \longrightarrow \sigma[x \mapsto v]}$$

$$\frac{\langle c_1, \sigma \rangle \longrightarrow \sigma' \quad \langle c_2, \sigma' \rangle \longrightarrow \sigma''}{\langle c_1 ; c_2, \sigma \rangle \longrightarrow \sigma''}$$

$$\frac{[\![b]\!]\sigma = \text{False}}{\langle \text{while } b \text{ do } c, \sigma \rangle \longrightarrow \sigma}$$

$$\frac{[\![b]\!]\sigma = \text{True} \quad \langle c, \sigma \rangle \longrightarrow \sigma' \quad \langle \text{while } b \text{ do } c, \sigma' \rangle \longrightarrow \sigma''}{\langle \text{while } b \text{ do } c, \sigma \rangle \longrightarrow \sigma''}$$

What does this mean?

What does this mean?

➜ $\langle c, \sigma \rangle \longrightarrow \sigma'$ fan  r a relation $(c, \sigma, \sigma') \in E$

What does this mean?

- $\langle c, \sigma \rangle \longrightarrow \sigma'$ fan  r a relation $(c, \sigma, \sigma') \in E$
- relations are sets: E state $\times$ state) set

What does this mean?

- $\langle c, \sigma \rangle \longrightarrow \sigma'$ fan[illegible]r a relation $(c, \sigma, \sigma') \in E$
- relations are sets: E[illegible] state $\times$ state) set
- the rules define a se[illegible]y

What does this mean?

- ➜ $\langle c, \sigma \rangle \longrightarrow \sigma'$ fan[illegible]r a relation $(c, \sigma, \sigma') \in E$
- ➜ relations are sets: E [illegible] state × state) set
- ➜ the rules define a se[illegible]y

Big: which set?

Simpler Example

$$\frac{n \in N}{n+1 \in N}$$

Simpler Example

$$\frac{n \in N}{n + 1 \in N}$$

- ➜ N is the set of natu s $\mathbb{N}$

Simpler Example

$$\frac{n \in N}{n+1 \in N}$$

- ➜ N is the set of natu[illegible]s $\mathbb{N}$
- ➜ But why not the se[illegible]mbers? $0 \in \mathbb{R}$, $n \in \mathbb{R} \implies n+1 \in \mathbb{R}$

Simpler Example

$$\frac{n \in N}{n+1 \in N}$$

- ➜ N is the set of natu… $\mathbb{N}$
- ➜ But why not the se… mbers? $0 \in \mathbb{R}$, $n \in \mathbb{R} \implies n+1 \in \mathbb{R}$
- ➜ $\mathbb{N}$ is the **smallest** set that is **consistent** with the rules.

Simpler Example

$$\frac{n \in N}{n+1 \in N}$$

- ➜ N is the set of natu[illegible]s $\mathbb{N}$
- ➜ But why not the se[illegible]mbers? $0 \in \mathbb{R}$, $n \in \mathbb{R} \implies n+1 \in \mathbb{R}$
- ➜ $\mathbb{N}$ is the **smallest** set that is **consistent** with the rules.

Why the smallest set?

Simpler Example

$$\frac{n \in N}{n+1 \in N}$$

- ➜ N is the set of natu[illegible]s $\mathbb{N}$
- ➜ But why not the se[illegible]mbers? $0 \in \mathbb{R}$, $n \in \mathbb{R} \implies n+1 \in \mathbb{R}$
- ➜ $\mathbb{N}$ is the **smallest** set that is **consistent** with the rules.

Why the smallest set?

- ➜ Objective: **no junk**. Only what must be in X shall be in X.

Simpler Example

$$\frac{n \in N}{n+1 \in N}$$

- ➜ N is the set of natu[illegible]s $\mathbb{N}$
- ➜ But why not the se[illegible]mbers? $0 \in \mathbb{R}$, $n \in \mathbb{R} \implies n+1 \in \mathbb{R}$
- ➜ $\mathbb{N}$ is the **smallest** set that is **consistent** with the rules.

Why the smallest set?

- ➜ Objective: **no junk**. Only what must be in X shall be in X.
- ➜ Gives rise to a nice proof principle (rule induction)

Simpler Example

$$\frac{n \in N}{n+1 \in N}$$

- ➜ N is the set of natu[illegible]s $\mathbb{N}$
- ➜ But why not the se[illegible]mbers? $0 \in \mathbb{R}$, $n \in \mathbb{R} \implies n+1 \in \mathbb{R}$
- ➜ $\mathbb{N}$ is the **smallest** set that is **consistent** with the rules.

Why the smallest set?

- ➜ Objective: **no junk**. Only what must be in X shall be in X.
- ➜ Gives rise to a nice proof principle (rule induction)
- ➜ Alternative (greatest set) occasionally also useful: coinduction

Rule Induction

$$\frac{n \in N}{n + 1 \in N}$$

duction principle

$$[\![P\ 0;\ \bigwedge n.\ P\ n \Longrightarrow P\ (n + 1)]\!] \Longrightarrow \forall x \in N.\ P\ x$$

Demo

Inductive Definitions

We have learned today ...

- Sets

We have learned today ...

- Sets
- Type Definitions

We have learned today ...

- ➜ Sets
- ➜ Type Definitions
- ➜ Inductive Definition