Foundations of Types in Isabelle/HOL

Simon Foster Jim Woodcock
University of York

18th August 2022

Overview

1. Relationship between sets and types
2. Defining new types using subsets
3. Description operators
4. Benefits and limitations of types

Outline

1. Relationship between sets and types
2. Defining new types using subsets
3. Description operators
4. Benefits and limitations of types

Types as Sets

- Every type `T` has corresponding non-empty set as its universe

  `UNIV :: T set`

- Any value `x :: T` is an element of the type's universe `x` $\in$ `UNIV`.
- All HOL types have at least one element, hence

  $\exists x.\, x \in$ `UNIV`   `UNIV_witness`

- Types are like maximal sets, the largest set of well-typed members.
- `(UNIV :: ocean set) = {Atlantic, Pacific, Indian, Arctic}`.

Types as Sets

- Every type `T` has corresponding non-empty set as its universe

  `UNIV :: T set`

- Any value `x :: T` is an element of the type's universe `x` $\in$ `UNIV`.
- All HOL types have at least one element, hence

  $\exists x.\, x \in$ `UNIV` `UNIV_witness`

- Types are like maximal sets, the largest set of well-typed members.
- `(UNIV :: ocean set) = {Atlantic, Pacific, Indian, Arctic}`.

Types as Sets

- Every type `T` has corresponding non-empty set as its universe

  `UNIV :: T set`

- Any value `x :: T` is an element of the type's universe `x ∈ UNIV`.
- All HOL types have at least one element, hence

  $\exists x. x \in$ `UNIV` `UNIV_witness`

- Types are like maximal sets, the largest set of well-typed members.
- `(UNIV :: ocean set) = {Atlantic, Pacific, Indian, Arctic}`.

Types as Sets

- Every type `T` has corresponding non-empty set as its universe

  `UNIV :: T set`

- Any value `x :: T` is an element of the type's universe `x ∈ UNIV`.

- All HOL types have at least one element, hence

  $\exists x.\, x \in$ `UNIV` `UNIV_witness`

- Types are like maximal sets, the largest set of well-typed members.

- `(UNIV :: ocean set) = {Atlantic, Pacific, Indian, Arctic}`.

Types as Sets

- Every type `T` has corresponding non-empty set as its universe

  `UNIV :: T set`

- Any value `x :: T` is an element of the type's universe $x \in$ `UNIV`.
- All HOL types have at least one element, hence

  $\exists x.\, x \in$ `UNIV` `UNIV_witness`

- Types are like maximal sets, the largest set of well-typed members.
- `(UNIV :: ocean set) = {Atlantic, Pacific, Indian, Arctic}`.

Types as Sets

- Every type T has corresponding non-empty set as its universe

  UNIV :: T set

- Any value x :: T is an element of the type's universe x $\in$ UNIV.
- All HOL types have at least one element, hence

  $\exists x.\, x \in$ UNIV   UNIV_witness

- Types are like maximal sets, the largest set of well-typed members.
- (UNIV :: ocean set) = {Atlantic, Pacific, Indian, Arctic}.

Types as Sets

- Every type `T` has corresponding non-empty set as its universe

  `UNIV :: T set`

- Any value `x :: T` is an element of the type's universe `x` $\in$ `UNIV`.

- All HOL types have at least one element, hence

  $\exists\, x.\, x \in$ `UNIV` `UNIV_witness`

- Types are like maximal sets, the largest set of well-typed members.

- `(UNIV :: ocean set) = {Atlantic, Pacific, Indian, Arctic}`.

Types as Sets

- Every type `T` has corresponding non-empty set as its universe

  `UNIV :: T set`

- Any value `x :: T` is an element of the type's universe `x ∈ UNIV`.
- All HOL types have at least one element, hence

  $\exists\, x.\, x \in$ `UNIV`   `UNIV_witness`

- Types are like maximal sets, the largest set of well-typed members.
- `(UNIV :: ocean set) = {Atlantic, Pacific, Indian, Arctic}`.

Example: Equality Proofs

$$x \notin fv(\Gamma)\ \frac{x \in A, \Gamma \vdash x \in B}{\Gamma \vdash A \subseteq B}\ \texttt{subsetI} \qquad \frac{\Gamma \vdash A \subseteq B \qquad \Gamma \vdash B \subseteq A}{\Gamma \vdash A = B}\ \texttt{equalityI}$$

```
lemma ocean_UNIV: "UNIV = {Atlantic, Pacific, Indian, Arctic}"
proof (rule equalityI; rule subsetI)
  fix x :: ocean
  assume "x ∈ UNIV"
  show "x ∈ {Atlantic, Pacific, Indian, Arctic}"
    by (cases x; simp) (* Automate Case Analysis *)
next
  fix x :: ocean
  assume "x ∈ {Atlantic, Pacific, Indian, Arctic}"
  show "x ∈ UNIV"
    by (fact UNIV_I)
qed
```

Example: Equality Proofs

$$\frac{x \in A, \Gamma \vdash x \in B}{\Gamma \vdash A \subseteq B}\ \texttt{subsetI} \quad (x \notin fv(\Gamma)) \qquad \frac{\Gamma \vdash A \subseteq B \qquad \Gamma \vdash B \subseteq A}{\Gamma \vdash A = B}\ \texttt{equalityI}$$

```
lemma ocean_UNIV: "UNIV = {Atlantic, Pacific, Indian, Arctic}"
proof (rule equalityI; rule subsetI)
  fix x :: ocean
  assume "x ∈ UNIV"
  show "x ∈ {Atlantic, Pacific, Indian, Arctic}"
    by (cases x; simp) (* Automate Case Analysis *)
next
  fix x :: ocean
  assume "x ∈ {Atlantic, Pacific, Indian, Arctic}"
  show "x ∈ UNIV"
    by (fact UNIV_I)
qed
```

Example: Equality Proofs

$$x \notin \mathit{fv}(\Gamma)\ \frac{x \in A, \Gamma \vdash x \in B}{\Gamma \vdash A \subseteq B}\ \texttt{subsetI} \qquad \frac{\Gamma \vdash A \subseteq B \qquad \Gamma \vdash B \subseteq A}{\Gamma \vdash A = B}\ \texttt{equalityI}$$

```
lemma ocean_UNIV: "UNIV = {Atlantic, Pacific, Indian, Arctic}"
proof (rule equalityI; rule subsetI)
  fix x :: ocean
  assume "x ∈ UNIV"
  show "x ∈ {Atlantic, Pacific, Indian, Arctic}"
    by (cases x; simp) (* Automate Case Analysis *)
next
  fix x :: ocean
  assume "x ∈ {Atlantic, Pacific, Indian, Arctic}"
  show "x ∈ UNIV"
    by (fact UNIV_I)
qed
```

Example: Equality Proofs

$$x \notin fv(\Gamma)\ \frac{x \in A, \Gamma \vdash x \in B}{\Gamma \vdash A \subseteq B}\ \texttt{subsetI} \qquad \frac{\Gamma \vdash A \subseteq B \qquad \Gamma \vdash B \subseteq A}{\Gamma \vdash A = B}\ \texttt{equalityI}$$

```
lemma ocean_UNIV: "UNIV = {Atlantic, Pacific, Indian, Arctic}"
proof (rule equalityI; rule subsetI)
  fix x :: ocean
  assume "x ∈ UNIV"
  show "x ∈ {Atlantic, Pacific, Indian, Arctic}"
    by (cases x; simp) (* Automate Case Analysis *)
next
  fix x :: ocean
  assume "x ∈ {Atlantic, Pacific, Indian, Arctic}"
  show "x ∈ UNIV"
    by (fact UNIV_I)
qed
```

Example: Equality Proofs

$$\frac{x \in A, \Gamma \vdash x \in B}{\Gamma \vdash A \subseteq B}\; x \notin \mathit{fv}(\Gamma)\ \texttt{subsetI} \qquad \frac{\Gamma \vdash A \subseteq B \qquad \Gamma \vdash B \subseteq A}{\Gamma \vdash A = B}\ \texttt{equalityI}$$

```
lemma ocean_UNIV: "UNIV = {Atlantic, Pacific, Indian, Arctic}"
proof (rule equalityI; rule subsetI)
  fix x :: ocean
  assume "x ∈ UNIV"
  show "x ∈ {Atlantic, Pacific, Indian, Arctic}"
    by (cases x; simp) (* Automate Case Analysis *)
next
  fix x :: ocean
  assume "x ∈ {Atlantic, Pacific, Indian, Arctic}"
  show "x ∈ UNIV"
    by (fact UNIV_I)
qed
```

Example: Equality Proofs

$$\frac{x \in A, \Gamma \vdash x \in B}{\Gamma \vdash A \subseteq B}\ \texttt{subsetI}\ \ (x \notin \mathit{fv}(\Gamma)) \qquad \frac{\Gamma \vdash A \subseteq B \qquad \Gamma \vdash B \subseteq A}{\Gamma \vdash A = B}\ \texttt{equalityI}$$

```
lemma ocean_UNIV: "UNIV = {Atlantic, Pacific, Indian, Arctic}"
proof (rule equalityI; rule subsetI)
  fix x :: ocean
  assume "x ∈ UNIV"
  show "x ∈ {Atlantic, Pacific, Indian, Arctic}"
    by (cases x; simp) (* Automate Case Analysis *)
next
  fix x :: ocean
  assume "x ∈ {Atlantic, Pacific, Indian, Arctic}"
  show "x ∈ UNIV"
    by (fact UNIV_I)
qed
```

Example: Equality Proofs

$$x \notin fv(\Gamma)\ \frac{x \in A, \Gamma \vdash x \in B}{\Gamma \vdash A \subseteq B}\ \texttt{subsetI} \qquad \frac{\Gamma \vdash A \subseteq B \qquad \Gamma \vdash B \subseteq A}{\Gamma \vdash A = B}\ \texttt{equalityI}$$

```
lemma ocean_UNIV: "UNIV = {Atlantic, Pacific, Indian, Arctic}"
proof (rule equalityI; rule subsetI)
  fix x :: ocean
  assume "x ∈ UNIV"
  show "x ∈ {Atlantic, Pacific, Indian, Arctic}"
    by (cases x; simp) (* Automate Case Analysis *)
next
  fix x :: ocean
  assume "x ∈ {Atlantic, Pacific, Indian, Arctic}"
  show "x ∈ UNIV"
    by (fact UNIV_I)
qed
```

Example: Equality Proofs

$$\frac{x \in A, \Gamma \vdash x \in B}{\Gamma \vdash A \subseteq B}\ \texttt{subsetI}\ \ (x \notin \mathit{fv}(\Gamma)) \qquad \frac{\Gamma \vdash A \subseteq B \qquad \Gamma \vdash B \subseteq A}{\Gamma \vdash A = B}\ \texttt{equalityI}$$

```
lemma ocean_UNIV: "UNIV = {Atlantic, Pacific, Indian, Arctic}"
proof (rule equalityI; rule subsetI)
  fix x :: ocean
  assume "x ∈ UNIV"
  show "x ∈ {Atlantic, Pacific, Indian, Arctic}"
    by (cases x; simp) (* Automate Case Analysis *)
next
  fix x :: ocean
  assume "x ∈ {Atlantic, Pacific, Indian, Arctic}"
  show "x ∈ UNIV"
    by (fact UNIV_I)
qed
```

Example: Equality Proofs

$$x \notin fv(\Gamma)\ \frac{x \in A, \Gamma \vdash x \in B}{\Gamma \vdash A \subseteq B}\ \texttt{subsetI} \qquad \frac{\Gamma \vdash A \subseteq B \qquad \Gamma \vdash B \subseteq A}{\Gamma \vdash A = B}\ \texttt{equalityI}$$

```
lemma ocean_UNIV: "UNIV = {Atlantic, Pacific, Indian, Arctic}"
proof (rule equalityI; rule subsetI)
  fix x :: ocean
  assume "x ∈ UNIV"
  show "x ∈ {Atlantic, Pacific, Indian, Arctic}"
    by (cases x; simp) (* Automate Case Analysis *)
next
  fix x :: ocean
  assume "x ∈ {Atlantic, Pacific, Indian, Arctic}"
  show "x ∈ UNIV"
    by (fact UNIV_I)
qed
```

Example: Equality Proofs

$$\frac{x \in A, \Gamma \vdash x \in B}{\Gamma \vdash A \subseteq B}\ \texttt{subsetI}\ \ {\scriptstyle x \notin fv(\Gamma)} \qquad \frac{\Gamma \vdash A \subseteq B \qquad \Gamma \vdash B \subseteq A}{\Gamma \vdash A = B}\ \texttt{equalityI}$$

```
lemma ocean_UNIV: "UNIV = {Atlantic, Pacific, Indian, Arctic}"
proof (rule equalityI; rule subsetI)
  fix x :: ocean
  assume "x ∈ UNIV"
  show "x ∈ {Atlantic, Pacific, Indian, Arctic}"
    by (cases x; simp) (* Automate Case Analysis *)
next
  fix x :: ocean
  assume "x ∈ {Atlantic, Pacific, Indian, Arctic}"
  show "x ∈ UNIV"
    by (fact UNIV_I)
qed
```

Example: Equality Proofs

$$\frac{x \in A, \Gamma \vdash x \in B}{\Gamma \vdash A \subseteq B}\ \texttt{subsetI} \quad (x \notin \mathit{fv}(\Gamma)) \qquad \frac{\Gamma \vdash A \subseteq B \qquad \Gamma \vdash B \subseteq A}{\Gamma \vdash A = B}\ \texttt{equalityI}$$

```
lemma ocean_UNIV: "UNIV = {Atlantic, Pacific, Indian, Arctic}"
proof (rule equalityI; rule subsetI)
  fix x :: ocean
  assume "x ∈ UNIV"
  show "x ∈ {Atlantic, Pacific, Indian, Arctic}"
    by (cases x; simp) (* Automate Case Analysis *)
next
  fix x :: ocean
  assume "x ∈ {Atlantic, Pacific, Indian, Arctic}"
  show "x ∈ UNIV"
    by (fact UNIV_I)
qed
```

Example: Equality Proofs

$$x \notin fv(\Gamma)\ \frac{x \in A, \Gamma \vdash x \in B}{\Gamma \vdash A \subseteq B}\ \texttt{subsetI} \qquad \frac{\Gamma \vdash A \subseteq B \qquad \Gamma \vdash B \subseteq A}{\Gamma \vdash A = B}\ \texttt{equalityI}$$

```
lemma ocean_UNIV: "UNIV = {Atlantic, Pacific, Indian, Arctic}"
proof (rule equalityI; rule subsetI)
  fix x :: ocean
  assume "x ∈ UNIV"
  show "x ∈ {Atlantic, Pacific, Indian, Arctic}"
    by (cases x; simp) (* Automate Case Analysis *)
next
  fix x :: ocean
  assume "x ∈ {Atlantic, Pacific, Indian, Arctic}"
  show "x ∈ UNIV"
    by (fact UNIV_I)
qed
```

Example: Equality Proofs

$$x \notin fv(\Gamma)\ \frac{x \in A, \Gamma \vdash x \in B}{\Gamma \vdash A \subseteq B}\ \texttt{subsetI} \qquad \frac{\Gamma \vdash A \subseteq B \qquad \Gamma \vdash B \subseteq A}{\Gamma \vdash A = B}\ \texttt{equalityI}$$

```
lemma ocean_UNIV: "UNIV = {Atlantic, Pacific, Indian, Arctic}"
proof (rule equalityI; rule subsetI)
  fix x :: ocean
  assume "x ∈ UNIV"
  show "x ∈ {Atlantic, Pacific, Indian, Arctic}"
    by (cases x; simp) (* Automate Case Analysis *)
next
  fix x :: ocean
  assume "x ∈ {Atlantic, Pacific, Indian, Arctic}"
  show "x ∈ UNIV"
    by (fact UNIV_I)
qed
```

Example: Equality Proofs

$$\frac{x \in A, \Gamma \vdash x \in B}{\Gamma \vdash A \subseteq B}\ \texttt{subsetI} \quad (x \notin \mathit{fv}(\Gamma)) \qquad \frac{\Gamma \vdash A \subseteq B \qquad \Gamma \vdash B \subseteq A}{\Gamma \vdash A = B}\ \texttt{equalityI}$$

```
lemma ocean_UNIV: "UNIV = {Atlantic, Pacific, Indian, Arctic}"
proof (rule equalityI; rule subsetI)
  fix x :: ocean
  assume "x ∈ UNIV"
  show "x ∈ {Atlantic, Pacific, Indian, Arctic}"
    by (cases x; simp) (* Automate Case Analysis *)
next
  fix x :: ocean
  assume "x ∈ {Atlantic, Pacific, Indian, Arctic}"
  show "x ∈ UNIV"
    by (fact UNIV_I)
qed
```

Example: Equality Proofs

$$x \notin fv(\Gamma)\ \frac{x \in A, \Gamma \vdash x \in B}{\Gamma \vdash A \subseteq B}\ \texttt{subsetI} \qquad \frac{\Gamma \vdash A \subseteq B \qquad \Gamma \vdash B \subseteq A}{\Gamma \vdash A = B}\ \texttt{equalityI}$$

```
lemma ocean_UNIV: "UNIV = {Atlantic, Pacific, Indian, Arctic}"
proof (rule equalityI; rule subsetI)
  fix x :: ocean
  assume "x ∈ UNIV"
  show "x ∈ {Atlantic, Pacific, Indian, Arctic}"
    by (cases x; simp) (* Automate Case Analysis *)
next
  fix x :: ocean
  assume "x ∈ {Atlantic, Pacific, Indian, Arctic}"
  show "x ∈ UNIV"
    by (fact UNIV_I)
qed
```

Outline

1. Relationship between sets and types
2. Defining new types using subsets
3. Description operators
4. Benefits and limitations of types

Types Definitions

- **type_synonym**, **datatype**, and **record** create types.
- But there is a low-level mechanism.
- New types can be created from a non-empty subset of an existing type.
- Use the command **typedef**:

  typedef NT = "A :: T set"**by** ...

- Requires a proof that the provided set A is non-empty.
- **typedef** is used internally by both **datatype** and **record**.
- Generates conversion functions:

  $\mathrm{Abs\_NT} :: \mathrm{T} \Rightarrow \mathrm{NT}$

  $\mathrm{Rep\_NT} :: \mathrm{NT} \Rightarrow \mathrm{T}$.

Types Definitions

- **type_synonym**, **datatype**, and **record** create types.
- But there is a low-level mechanism.
- New types can be created from a non-empty subset of an existing type.
- Use the command **typedef**:

  typedef NT = "A :: T set" **by** ...

- Requires a proof that the provided set A is non-empty.
- **typedef** is used internally by both **datatype** and **record**.
- Generates conversion functions:

  Abs_NT :: T ⇒ NT

  Rep_NT :: NT ⇒ T.

Types Definitions

- **type_synonym**, **datatype**, and **record** create types.
- But there is a low-level mechanism.
- New types can be created from a non-empty subset of an existing type.
- Use the command **typedef**:

  typedef NT = "A :: T set" **by** ...

- Requires a proof that the provided set A is non-empty.
- **typedef** is used internally by both **datatype** and **record**.
- Generates conversion functions:

  Abs_NT :: T $\Rightarrow$ NT

  Rep_NT :: NT $\Rightarrow$ T.

Types Definitions

- **type_synonym**, **datatype**, and **record** create types.
- But there is a low-level mechanism.
- New types can be created from a non-empty subset of an existing type.
- Use the command **typedef**:

  typedef NT = "A :: T set" **by** ...

- Requires a proof that the provided set A is non-empty.
- **typedef** is used internally by both **datatype** and **record**.
- Generates conversion functions:

  $\text{Abs\_NT} :: T \Rightarrow NT$

  $\text{Rep\_NT} :: NT \Rightarrow T.$

Types Definitions

- **type_synonym**, **datatype**, and **record** create types.
- But there is a low-level mechanism.
- New types can be created from a non-empty subset of an existing type.
- Use the command **typedef**:

```
typedef NT = "A :: T set" by ...
```

- Requires a proof that the provided set A is non-empty.
- **typedef** is used internally by both **datatype** and **record**.
- Generates conversion functions:

  $\mathrm{Abs\_NT} :: \mathrm{T} \Rightarrow \mathrm{NT}$

  $\mathrm{Rep\_NT} :: \mathrm{NT} \Rightarrow \mathrm{T}.$

Types Definitions

- **type_synonym**, **datatype**, and **record** create types.
- But there is a low-level mechanism.
- New types can be created from a non-empty subset of an existing type.
- Use the command **typedef**:

```
typedef NT = "A :: T set" by ...
```

- Requires a proof that the provided set A is non-empty.
- **typedef** is used internally by both **datatype** and **record**.
- Generates conversion functions:

$$\mathrm{Abs\_NT} :: \mathrm{T} \Rightarrow \mathrm{NT}$$

$$\mathrm{Rep\_NT} :: \mathrm{NT} \Rightarrow \mathrm{T}.$$

Types Definitions

- **type_synonym**, **datatype**, and **record** create types.
- But there is a low-level mechanism.
- New types can be created from a non-empty subset of an existing type.
- Use the command **typedef**:

```
typedef NT = "A :: T set" by ...
```

- Requires a proof that the provided set A is non-empty.
- **typedef** is used internally by both **datatype** and **record**.
- Generates conversion functions:

  $\mathrm{Abs\_NT} :: T \Rightarrow NT$

  $\mathrm{Rep\_NT} :: NT \Rightarrow T.$

Types Definitions

- **type_synonym**, **datatype**, and **record** create types.
- But there is a low-level mechanism.
- New types can be created from a non-empty subset of an existing type.
- Use the command **typedef**:

  ```
  typedef NT = "A :: T set" by ...
  ```

- Requires a proof that the provided set A is non-empty.
- **typedef** is used internally by both **datatype** and **record**.
- Generates conversion functions:

  $\mathrm{Abs\_NT} :: \mathrm{T} \Rightarrow \mathrm{NT}$

  $\mathrm{Rep\_NT} :: \mathrm{NT} \Rightarrow \mathrm{T}.$

Types Definitions

- **type_synonym**, **datatype**, and **record** create types.
- But there is a low-level mechanism.
- New types can be created from a non-empty subset of an existing type.
- Use the command **typedef**:

  typedef NT = "A :: T set"**by** ...

- Requires a proof that the provided set A is non-empty.
- **typedef** is used internally by both **datatype** and **record**.
- Generates conversion functions:

  Abs_NT :: T ⇒ NT

  Rep_NT :: NT ⇒ T.

Types Definitions

- **type_synonym**, **datatype**, and **record** create types.
- But there is a low-level mechanism.
- New types can be created from a non-empty subset of an existing type.
- Use the command **typedef**:

```
typedef NT = "A :: T set" by ...
```

- Requires a proof that the provided set A is non-empty.
- **typedef** is used internally by both **datatype** and **record**.
- Generates conversion functions:

```
Abs_NT :: T ⇒ NT
Rep_NT :: NT ⇒ T.
```

Types Definitions

- **type_synonym**, **datatype**, and **record** create types.
- But there is a low-level mechanism.
- New types can be created from a non-empty subset of an existing type.
- Use the command **typedef**:

  typedef NT = "A :: T set"**by** ...

- Requires a proof that the provided set A is non-empty.
- **typedef** is used internally by both **datatype** and **record**.
- Generates conversion functions:

  Abs_NT :: T ⇒ NT

  Rep_NT :: NT ⇒ T.

Types Definitions Visualised

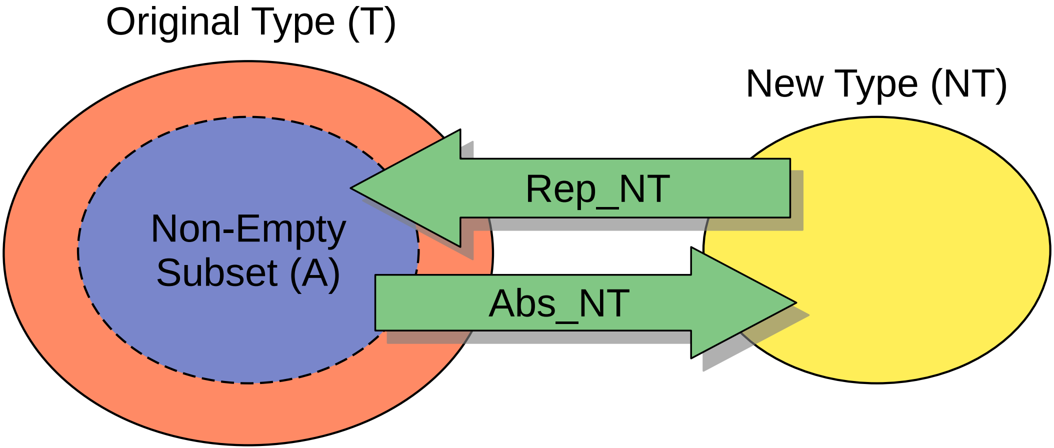
```
typedef NT = "A :: T set" by ...
```

Types Definitions Visualised

```
typedef NT = "A :: T set" by ...
```

Types Definitions Visualised

```
typedef NT = "A :: T set" by ...
```



Conversion Functions

- Conversion functions $\texttt{Abs\_NT} :: \texttt{T} \Rightarrow \texttt{NT}$ and $\texttt{Rep\_NT} :: \texttt{NT} \Rightarrow \texttt{T}$.
- They satisfy:

$$\texttt{Abs\_NT}\,(\texttt{Rep\_NT}\ x) = x$$

$$x \in A \Longrightarrow \texttt{Rep\_NT}\,(\texttt{Abs\_NT}\ x) = x$$

$$\texttt{Rep\_NT}\ x \in A$$

Conversion Functions

```
typedef NT = "A :: T set" by ...
```

- Conversion functions `Abs_NT :: T ⇒ NT` and `Rep_NT :: NT ⇒ T`.
- They satisfy:

$$\mathrm{Abs\_NT}\,(\mathrm{Rep\_NT}\;x) = x$$

$$x \in A \Longrightarrow \mathrm{Rep\_NT}\,(\mathrm{Abs\_NT}\;x) = x$$

$$\mathrm{Rep\_NT}\;x \in A$$

Conversion Functions

```
typedef NT = "A :: T set" by ...
```

- Conversion functions `Abs_NT :: T ⇒ NT` and `Rep_NT :: NT ⇒ T`.
- They satisfy:

Conversion Functions

```
typedef NT = "A :: T set" by ...
```

- Conversion functions $\texttt{Abs\_NT} :: \texttt{T} \Rightarrow \texttt{NT}$ and $\texttt{Rep\_NT} :: \texttt{NT} \Rightarrow \texttt{T}$.
- They satisfy:

$$\texttt{Abs\_NT}\,(\texttt{Rep\_NT}\,x) = x$$

$$x \in A \Longrightarrow \texttt{Rep\_NT}\,(\texttt{Abs\_NT}\,x) = x$$

$$\texttt{Rep\_NT}\,x \in A$$

Conversion Functions

```
typedef NT = "A :: T set" by ...
```

- Conversion functions `Abs_NT :: T ⇒ NT` and `Rep_NT :: NT ⇒ T`.
- They satisfy:

$$\mathtt{Abs\_NT}\,(\mathtt{Rep\_NT}\,x) = x$$

$$x \in A \Longrightarrow \mathtt{Rep\_NT}\,(\mathtt{Abs\_NT}\,x) = x$$

$$\mathtt{Rep\_NT}\,x \in A$$

Conversion Functions

```
typedef NT = "A :: T set" by ...
```

- Conversion functions `Abs_NT :: T ⇒ NT` and `Rep_NT :: NT ⇒ T`.
- They satisfy:

$$\mathtt{Abs\_NT}\,(\mathtt{Rep\_NT}\,x) = x$$

$$x \in A \Longrightarrow \mathtt{Rep\_NT}\,(\mathtt{Abs\_NT}\,x) = x$$

$$\mathtt{Rep\_NT}\,x \in A$$

Conversion Functions

```
typedef NT = "A :: T set" by ...
```

- Conversion functions `Abs_NT :: T ⇒ NT` and `Rep_NT :: NT ⇒ T`.
- They satisfy:

$$\mathtt{Abs\_NT}\,(\mathtt{Rep\_NT}\,x) = x$$

$$x \in A \Longrightarrow \mathtt{Rep\_NT}\,(\mathtt{Abs\_NT}\,x) = x$$

$$\mathtt{Rep\_NT}\,x \in A$$

Example: Definining Non-Zero Numbers (1)

- Prove that {x :: nat. x > 0} is non-empty by supplying witness 1.
- **typedef** creates from_nat1 :: nat1 ⇒ nat.
- That converts a non-zero nat to a nat.
- And to_nat1 :: nat ⇒ nat1 the does the converse, such that

x > 0 ⟹ from_nat1 (to_nat1 x) = x   (to_nat1_inverse)

Example: Definining Non-Zero Numbers (1)

Example

```
typedef nat1 = "{x :: nat. x > 0}"
  morphisms from_nat1 to_nat1
  by (rule_tac x="1" in exI, simp)
```

- Prove that `{x :: nat. x > 0}` is non-empty by supplying witness 1.
- **typedef** creates `from_nat1 :: nat1 ⇒ nat`.
- That converts a non-zero `nat` to a `nat`.
- And `to_nat1 :: nat ⇒ nat1` the does the converse, such that

$$x > 0 \Longrightarrow \mathit{from\_nat1}\ (\mathit{to\_nat1}\ x) = x \qquad (\mathit{to\_nat1\_inverse})$$

Example: Definining Non-Zero Numbers (1)

Example

```
typedef nat1 = "{x :: nat. x > 0}"
  morphisms from_nat1 to_nat1
  by (rule_tac x="1" in exI, simp)
```

- Prove that `{x :: nat. x > 0}` is non-empty by supplying witness 1.
- **typedef** creates `from_nat1 :: nat1 ⇒ nat`.
- That converts a non-zero `nat` to a `nat`.
- And `to_nat1 :: nat ⇒ nat1` the does the converse, such that

$$x > 0 \Longrightarrow \mathtt{from\_nat1}\ (\mathtt{to\_nat1}\ x) = x \qquad (\mathtt{to\_nat1\_inverse})$$

Example: Definining Non-Zero Numbers (1)

Example

```
typedef nat1 = "{x :: nat. x > 0}"
  morphisms from_nat1 to_nat1
  by (rule_tac x="1" in exI, simp)
```

- Prove that {x :: nat. x > 0} is non-empty by supplying witness 1.
- **typedef** creates from_nat1 :: nat1 ⇒ nat.
- That converts a non-zero nat to a nat.
- And to_nat1 :: nat ⇒ nat1 the does the converse, such that

$$x > 0 \Longrightarrow \text{from\_nat1 (to\_nat1 x)} = x \qquad \text{(to\_nat1\_inverse)}$$

Example: Definining Non-Zero Numbers (1)

Example

```
typedef nat1 = "{x :: nat. x > 0}"
  morphisms from_nat1 to_nat1
  by (rule_tac x="1" in exI, simp)
```

- Prove that `{x :: nat. x > 0}` is non-empty by supplying witness `1`.
- `typedef` creates `from_nat1 :: nat1 ⇒ nat`.
- That converts a non-zero `nat` to a `nat`.
- And `to_nat1 :: nat ⇒ nat1` the does the converse, such that

  `x > 0 ⟹ from_nat1 (to_nat1 x) = x     (to_nat1_inverse)`

Example: Definining Non-Zero Numbers (1)

Example

```
typedef nat1 = "{x :: nat. x > 0}"
  morphisms from_nat1 to_nat1
  by (rule_tac x="1" in exI, simp)
```

- Prove that `{x :: nat. x > 0}` is non-empty by supplying witness `1`.
- `typedef` creates `from_nat1 :: nat1 ⇒ nat`.
- That converts a non-zero `nat` to a `nat`.
- And `to_nat1 :: nat ⇒ nat1` the does the converse, such that

  `x > 0 ⟹ from_nat1 (to_nat1 x) = x     (to_nat1_inverse)`

Example: Definining Non-Zero Numbers (1)

Example

```
typedef nat1 = "{x :: nat. x > 0}"
  morphisms from_nat1 to_nat1
  by (rule_tac x="1" in exI, simp)
```

- Prove that `{x :: nat. x > 0}` is non-empty by supplying witness `1`.
- **typedef** creates `from_nat1 :: nat1 ⇒ nat`.
- That converts a non-zero `nat` to a `nat`.
- And `to_nat1 :: nat ⇒ nat1` the does the converse, such that

$$x > 0 \Longrightarrow \texttt{from\_nat1 (to\_nat1 x)} = x \qquad \texttt{(to\_nat1\_inverse)}$$

Example: Definining Non-Zero Numbers (1)

Example

```
typedef nat1 = "{x :: nat. x > 0}"
  morphisms from_nat1 to_nat1
  by (rule_tac x="1" in exI, simp)
```

- Prove that `{x :: nat. x > 0}` is non-empty by supplying witness `1`.
- **typedef** creates `from_nat1 :: nat1 ⇒ nat`.
- That converts a non-zero `nat` to a `nat`.
- And `to_nat1 :: nat ⇒ nat1` the does the converse, such that

  $$x > 0 \Longrightarrow from\_nat1\ (to\_nat1\ x) = x \quad (to\_nat1\_inverse)$$

Example: Definining Non-Zero Numbers (1)

Example

```
typedef nat1 = "{x :: nat. x > 0}"
  morphisms from_nat1 to_nat1
  by (rule_tac x="1" in exI, simp)
```

- Prove that `{x :: nat. x > 0}` is non-empty by supplying witness 1.
- **typedef** creates `from_nat1 :: nat1 ⇒ nat`.
- That converts a non-zero `nat` to a `nat`.
- And `to_nat1 :: nat ⇒ nat1` the does the converse, such that

  `x > 0 ⟹ from_nat1 (to_nat1 x) = x     (to_nat1_inverse)`

Example: Definining Non-Zero Numbers (1)

Example

```
typedef nat1 = "{x :: nat. x > 0}"
  morphisms from_nat1 to_nat1
  by (rule_tac x="1" in exI, simp)
```

- Prove that `{x :: nat. x > 0}` is non-empty by supplying witness `1`.
- **typedef** creates `from_nat1 :: nat1 ⇒ nat`.
- That converts a non-zero `nat` to a `nat`.
- And `to_nat1 :: nat ⇒ nat1` the does the converse, such that

  `x > 0 ⟹ from_nat1 (to_nat1 x) = x    (to_nat1_inverse)`

Example: Defining Non-Zero Numbers (2)

- We can define functions on `nat1` by lifting those on `nat`.
- This process is automated with the lifting package (out of scope).
- But hold on, what is value of `to_nat1(0)`?
- This is underspecified, and so HOL assigns an arbitrary value.

Example: Defining Non-Zero Numbers (2)

- We can define functions on `nat1` by lifting those on `nat`.
- This process is automated with the lifting package (out of scope).
- But hold on, what is value of `to_nat1(0)`?
- This is underspecified, and so HOL assigns an arbitrary value.

Example: Defining Non-Zero Numbers (2)

- We can define functions on `nat1` by lifting those on `nat`.

```
definition plus1 :: "nat1 ⇒ nat1 ⇒ nat1" where
  "plus1 x y = to_nat1 (from_nat1 x + from_nat1 y)"
```

- This process is automated with the lifting package (out of scope).
- But hold on, what is value of `to_nat1(0)`?
- This is underspecified, and so HOL assigns an arbitrary value.

Example: Defining Non-Zero Numbers (2)

- We can define functions on `nat1` by lifting those on `nat`.

```
definition plus1 :: "nat1 ⇒ nat1 ⇒ nat1" where
  "plus1 x y = to_nat1 (from_nat1 x + from_nat1 y)"

lemma plus1:
  assumes "x > 0" "y > 0"
  shows "plus1 (to_nat1 x) (to_nat1 y) = to_nat1 (x+y)"
    by (simp add: assms plus1_def to_nat1_inverse)
```

- This process is automated with the lifting package (out of scope).
- But hold on, what is value of `to_nat1(0)`?
- This is underspecified, and so HOL assigns an arbitrary value.

Example: Defining Non-Zero Numbers (2)

- We can define functions on `nat1` by lifting those on `nat`.

```
definition plus1 :: "nat1 ⇒ nat1 ⇒ nat1" where
  "plus1 x y = to_nat1 (from_nat1 x + from_nat1 y)"

lemma plus1:
  assumes "x > 0" "y > 0"
  shows "plus1 (to_nat1 x) (to_nat1 y) = to_nat1 (x+y)"
    by (simp add: assms plus1_def to_nat1_inverse)
```

- This process is automated with the lifting package (out of scope).
- But hold on, what is value of `to_nat1(0)`?
- This is underspecified, and so HOL assigns an arbitrary value.

Example: Defining Non-Zero Numbers (2)

- We can define functions on `nat1` by lifting those on `nat`.

```
definition plus1 :: "nat1 ⇒ nat1 ⇒ nat1" where
  "plus1 x y = to_nat1 (from_nat1 x + from_nat1 y)"

lemma plus1:
  assumes "x > 0" "y > 0"
  shows "plus1 (to_nat1 x) (to_nat1 y) = to_nat1 (x+y)"
    by (simp add: assms plus1_def to_nat1_inverse)
```

- This process is automated with the lifting package (out of scope).
- But hold on, what is value of `to_nat1(0)`?
- This is underspecified, and so HOL assigns an arbitrary value.

Example: Defining Non-Zero Numbers (2)

- We can define functions on `nat1` by lifting those on `nat`.

```
definition plus1 :: "nat1 ⇒ nat1 ⇒ nat1" where
  "plus1 x y = to_nat1 (from_nat1 x + from_nat1 y)"

lemma plus1:
  assumes "x > 0" "y > 0"
  shows "plus1 (to_nat1 x) (to_nat1 y) = to_nat1 (x+y)"
    by (simp add: assms plus1_def to_nat1_inverse)
```

- This process is automated with the lifting package (out of scope).
- But hold on, what is value of `to_nat1(0)`?
- This is underspecified, and so HOL assigns an arbitrary value.

Outline

1. Relationship between sets and types
2. Defining new types using subsets
3. Description operators
4. Benefits and limitations of types

Description Operators

- No accident that types are non-empty, but an integral part of the logic.
- Hilbert's choice: pick a value $x :: T$ such that $P\ x$ holds.
- Indefinite article in natural language: "a cat sitting on my roof".
- $k \triangleq (\epsilon x.\ x \in \{0, 1, 2, 3\})$: could be 0, 1, 2, or 3.
- But we can infer general properties like $k \geq 0$ and $k \leq 3$.
- Relies on axiom of choice: can always pick a single element from a set.
- If no such x exists, return an arbitrary value of type T (e.g. $\epsilon\ x.\ False$).
- Uncomputable in general, e.g. what is $\epsilon x :: real.\ x * x = 2$?.
- Lets us deal with partiality, e.g. $to\_nat1(0)$, $x/0$, and $\sqrt{-1}$ (for $\mathbb{R}$).

Description Operators

- No accident that types are non-empty, but an integral part of the logic.
- Hilbert's choice: pick a value $x :: T$ such that $P\ x$ holds.
- Indefinite article in natural language: "a cat sitting on my roof".
- $k \triangleq (\epsilon x.\ x \in \{0, 1, 2, 3\})$: could be 0, 1, 2, or 3.
- But we can infer general properties like $k \geq 0$ and $k \leq 3$.
- Relies on axiom of choice: can always pick a single element from a set.
- If no such x exists, return an arbitrary value of type T (e.g. $\epsilon\ x.\ False$).
- Uncomputable in general, e.g. what is $\epsilon x :: real.\ x * x = 2$?.
- Lets us deal with partiality, e.g. $to\_nat1(0)$, $x/0$, and $\sqrt{-1}$ (for $\mathbb{R}$).

Description Operators

- No accident that types are non-empty, but an integral part of the logic.

Indefinite Description Operator

$\epsilon\, x :: T .\, P\, x$ also written as $SOME\, x :: T .\, P\, x$

- Hilbert's choice: pick a value $x :: T$ such that $P\, x$ holds.
- Indefinite article in natural language: "a cat sitting on my roof".
- $k \triangleq (\epsilon x.\, x \in \{0, 1, 2, 3\})$: could be 0, 1, 2, or 3.
- But we can infer general properties like $k \geq 0$ and $k \leq 3$.
- Relies on axiom of choice: can always pick a single element from a set.
- If no such x exists, return an arbitrary value of type T (e.g. $\epsilon\, x.\, False$).
- Uncomputable in general, e.g. what is $\epsilon x :: real.\, x * x = 2$?.
- Lets us deal with partiality, e.g. $to\_nat1(0)$, $x/0$, and $\sqrt{-1}$ (for $\mathbb{R}$).

Description Operators

- No accident that types are non-empty, but an integral part of the logic.

Indefinite Description Operator

$\epsilon\, x :: T.\, P\, x$ also written as $SOME\, x :: T.\, P\, x$

- Hilbert's choice: pick a value $x :: T$ such that $P\, x$ holds.
- Indefinite article in natural language: "a cat sitting on my roof".
- $k \triangleq (\epsilon x.\, x \in \{0, 1, 2, 3\})$: could be 0, 1, 2, or 3.
- But we can infer general properties like $k \geq 0$ and $k \leq 3$.
- Relies on axiom of choice: can always pick a single element from a set.
- If no such x exists, return an arbitrary value of type T (e.g. $\epsilon\, x.\, False$).
- Uncomputable in general, e.g. what is $\epsilon x :: real.\, x * x = 2$?.
- Lets us deal with partiality, e.g. $to\_nat1(0)$, $x/0$, and $\sqrt{-1}$ (for $\mathbb{R}$).

Description Operators

- No accident that types are non-empty, but an integral part of the logic.

Indefinite Description Operator

$\epsilon\, x :: T.\, P\, x$ also written as $SOME\; x :: T.\, P\, x$

- Hilbert's choice: pick a value $x :: T$ such that $P\, x$ holds.
- Indefinite article in natural language: "a cat sitting on my roof".
- $k \triangleq (\epsilon x.\, x \in \{0, 1, 2, 3\})$: could be 0, 1, 2, or 3.
- But we can infer general properties like $k \geq 0$ and $k \leq 3$.
- Relies on axiom of choice: can always pick a single element from a set.
- If no such x exists, return an arbitrary value of type T (e.g. $\epsilon\, x.\, False$).
- Uncomputable in general, e.g. what is $\epsilon x :: real.\, x * x = 2$?.
- Lets us deal with partiality, e.g. $to\_nat1(0)$, $x/0$, and $\sqrt{-1}$ (for $\mathbb{R}$).

Description Operators

- No accident that types are non-empty, but an integral part of the logic.

Indefinite Description Operator

$\epsilon\, x :: T.\, P\, x$ also written as $SOME\, x :: T.\, P\, x$

- Hilbert's choice: pick a value $x :: T$ such that $P\, x$ holds.
- Indefinite article in natural language: "a cat sitting on my roof".
- $k \triangleq (\epsilon x.\, x \in \{0, 1, 2, 3\})$: could be 0, 1, 2, or 3.
- But we can infer general properties like $k \geq 0$ and $k \leq 3$.
- Relies on axiom of choice: can always pick a single element from a set.
- If no such x exists, return an arbitrary value of type T (e.g. $\epsilon\, x.\, \mathit{False}$).
- Uncomputable in general, e.g. what is $\epsilon x :: real.\, x * x = 2$?.
- Lets us deal with partiality, e.g. $to\_nat1(0)$, $x/0$, and $\sqrt{-1}$ (for $\mathbb{R}$).

Description Operators

- No accident that types are non-empty, but an integral part of the logic.

Indefinite Description Operator

$\epsilon\, x :: T.\, P\, x$ also written as $SOME\, x :: T.\, P\, x$

- Hilbert's choice: pick a value $x :: T$ such that $P\, x$ holds.
- Indefinite article in natural language: "a cat sitting on my roof".
- $k \triangleq (\epsilon x.\, x \in \{0, 1, 2, 3\})$: could be 0, 1, 2, or 3.
- But we can infer general properties like $k \geq 0$ and $k \leq 3$.
- Relies on axiom of choice: can always pick a single element from a set.
- If no such x exists, return an arbitrary value of type T (e.g. $\epsilon\, x.\, False$).
- Uncomputable in general, e.g. what is $\epsilon x :: real.\, x * x = 2$?.
- Lets us deal with partiality, e.g. $to\_nat1(0)$, $x/0$, and $\sqrt{-1}$ (for $\mathbb{R}$).

Description Operators

- No accident that types are non-empty, but an integral part of the logic.

Indefinite Description Operator

$\epsilon\, x :: T.\, P\, x$ also written as $SOME\, x :: T.\, P\, x$

- Hilbert's choice: pick a value $x :: T$ such that $P\, x$ holds.
- Indefinite article in natural language: "a cat sitting on my roof".
- $k \triangleq (\epsilon x.\, x \in \{0, 1, 2, 3\})$: could be 0, 1, 2, or 3.
- But we can infer general properties like $k \geq 0$ and $k \leq 3$.
- Relies on axiom of choice: can always pick a single element from a set.
- If no such x exists, return an arbitrary value of type T (e.g. $\epsilon\, x.\, False$).
- Uncomputable in general, e.g. what is $\epsilon x :: real.\, x * x = 2$?.
- Lets us deal with partiality, e.g. $to\_nat1(0)$, $x/0$, and $\sqrt{-1}$ (for $\mathbb{R}$).

Description Operators

- No accident that types are non-empty, but an integral part of the logic.

Indefinite Description Operator

$\epsilon\, x :: T.\, P\, x$ also written as $SOME\, x :: T.\, P\, x$

- Hilbert's choice: pick a value $x :: T$ such that $P\, x$ holds.
- Indefinite article in natural language: "a cat sitting on my roof".
- $k \triangleq (\epsilon x.\, x \in \{0, 1, 2, 3\})$: could be 0, 1, 2, or 3.
- But we can infer general properties like $k \geq 0$ and $k \leq 3$.
- Relies on axiom of choice: can always pick a single element from a set.
- If no such x exists, return an arbitrary value of type T (e.g. $\epsilon\, x.\, False$).
- Uncomputable in general, e.g. what is $\epsilon x :: real.\, x * x = 2$?.
- Lets us deal with partiality, e.g. $to\_nat1(0)$, $x/0$, and $\sqrt{-1}$ (for $\mathbb{R}$).

Description Operators

- No accident that types are non-empty, but an integral part of the logic.

Indefinite Description Operator

$\epsilon\, x :: T.\, P\, x$ also written as $SOME\, x :: T.\, P\, x$

- Hilbert's choice: pick a value $x :: T$ such that $P\, x$ holds.
- Indefinite article in natural language: "a cat sitting on my roof".
- $k \triangleq (\epsilon x.\, x \in \{0, 1, 2, 3\})$: could be 0, 1, 2, or 3.
- But we can infer general properties like $k \geq 0$ and $k \leq 3$.
- Relies on axiom of choice: can always pick a single element from a set.
- If no such x exists, return an arbitrary value of type T (e.g. $\epsilon\, x.\, False$).
- Uncomputable in general, e.g. what is $\epsilon x :: real.\, x * x = 2$?.
- Lets us deal with partiality, e.g. $to\_nat1(0)$, $x/0$, and $\sqrt{-1}$ (for $\mathbb{R}$).

Description Operators

- No accident that types are non-empty, but an integral part of the logic.

Indefinite Description Operator

$\epsilon\, x :: T.\, P\, x$ also written as $SOME\, x :: T.\, P\, x$

- Hilbert's choice: pick a value $x :: T$ such that $P\, x$ holds.
- Indefinite article in natural language: "a cat sitting on my roof".
- $k \triangleq (\epsilon x.\, x \in \{0, 1, 2, 3\})$: could be 0, 1, 2, or 3.
- But we can infer general properties like $k \geq 0$ and $k \leq 3$.
- Relies on axiom of choice: can always pick a single element from a set.
- If no such x exists, return an arbitrary value of type T (e.g. $\epsilon\, x.\, False$).
- Uncomputable in general, e.g. what is $\epsilon x :: real.\, x * x = 2$?.
- Lets us deal with partiality, e.g. $to\_nat1(0)$, $x/0$, and $\sqrt{-1}$ (for $\mathbb{R}$).

Description Operators

- No accident that types are non-empty, but an integral part of the logic.

Indefinite Description Operator

$\epsilon\, x :: T.\, P\, x$ also written as $SOME\, x :: T.\, P\, x$

- Hilbert's choice: pick a value $x :: T$ such that $P\, x$ holds.
- Indefinite article in natural language: "a cat sitting on my roof".
- $k \triangleq (\epsilon x.\, x \in \{0, 1, 2, 3\})$: could be 0, 1, 2, or 3.
- But we can infer general properties like $k \geq 0$ and $k \leq 3$.
- Relies on axiom of choice: can always pick a single element from a set.
- If no such x exists, return an arbitrary value of type T (e.g. $\epsilon\, x.\, False$).
- Uncomputable in general, e.g. what is $\epsilon x :: real.\, x * x = 2$?.
- Lets us deal with partiality, e.g. $to\_nat1(0)$, $x/0$, and $\sqrt{-1}$ (for $\mathbb{R}$).

Reasoning with Indefinite Descriptions

- Here's an introduction rule for indefinite description:
- We need to show some t exists satisfying P to reason about ϵ.
- Very rarely need to reason about description operators directly.

Reasoning with Indefinite Descriptions

- Here's an introduction rule for indefinite description:
- We need to show some t exists satisfying P to reason about ϵ.
- Very rarely need to reason about description operators directly.

Reasoning with Indefinite Descriptions

- Here's an introduction rule for indefinite description:

Introduction Rule

- We need to show some t exists satisfying P to reason about ϵ.
- Very rarely need to reason about description operators directly.

Reasoning with Indefinite Descriptions

- Here's an introduction rule for indefinite description:

Introduction Rule

$$y \notin \mathit{fv}(\Gamma, P, Q)\ \frac{\Gamma \vdash P(t) \qquad P(y), \Gamma \vdash Q(y)}{\Gamma \vdash Q(\epsilon\, x.\, P(x))}\ \texttt{someI2}$$

- We need to show some t exists satisfying P to reason about ϵ.
- Very rarely need to reason about description operators directly.

Reasoning with Indefinite Descriptions

- Here's an introduction rule for indefinite description:

Introduction Rule

$$y \notin \mathit{fv}(\Gamma, P, Q)\ \frac{\Gamma \vdash P(t) \qquad P(y), \Gamma \vdash Q(y)}{\Gamma \vdash Q(\epsilon\, x.\, P(x))}\ \texttt{someI2}$$

- We need to show some t exists satisfying P to reason about ϵ.
- Very rarely need to reason about description operators directly.

Reasoning with Indefinite Descriptions

- Here's an introduction rule for indefinite description:

Introduction Rule

$$y \notin \mathit{fv}(\Gamma, P, Q)\ \frac{\Gamma \vdash P(t) \qquad P(y), \Gamma \vdash Q(y)}{\Gamma \vdash Q(\epsilon\, x.\, P(x)))}\ \texttt{someI2}$$

- We need to show some t exists satisfying P to reason about ϵ.

Example

$$\frac{\Gamma \vdash \in \{0,1,2,3\} \qquad y \in \{0,1,2,3\}, \Gamma \vdash y \leq 3}{\Gamma \vdash (\epsilon\, x.\, x \in \{0,1,2,3\}) \leq 3}$$

- Very rarely need to reason about description operators directly.

Reasoning with Indefinite Descriptions

- Here's an introduction rule for indefinite description:

Introduction Rule

$$y \notin \mathit{fv}(\Gamma, P, Q) \;\; \frac{\Gamma \vdash P(t) \qquad P(y), \Gamma \vdash Q(y)}{\Gamma \vdash Q(\epsilon\, x.\, P(x)))} \; \texttt{someI2}$$

- We need to show some t exists satisfying P to reason about ϵ.

Example

$$\frac{\Gamma \vdash \in \{0,1,2,3\} \qquad y \in \{0,1,2,3\}, \Gamma \vdash y \leq 3}{\Gamma \vdash (\epsilon\, x.\, x \in \{0,1,2,3\}) \leq 3}$$

- Very rarely need to reason about description operators directly.

Reasoning with Indefinite Descriptions

- Here's an introduction rule for indefinite description:

Introduction Rule

$$y \notin \mathit{fv}(\Gamma, P, Q) \ \frac{\Gamma \vdash P(t) \qquad P(y), \Gamma \vdash Q(y)}{\Gamma \vdash Q(\epsilon\, x.\, P(x))} \ \texttt{someI2}$$

- We need to show some t exists satisfying P to reason about ϵ.

Example

$$\frac{\Gamma \vdash \in \{0, 1, 2, 3\} \qquad y \in \{0, 1, 2, 3\}, \Gamma \vdash y \leq 3}{\Gamma \vdash (\epsilon\, x.\, x \in \{0, 1, 2, 3\}) \leq 3}$$

- Very rarely need to reason about description operators directly.

Reasoning with Indefinite Descriptions

- Here's an introduction rule for indefinite description:

Introduction Rule

$$y \notin \mathit{fv}(\Gamma, P, Q)\ \frac{\Gamma \vdash P(t) \qquad P(y), \Gamma \vdash Q(y)}{\Gamma \vdash Q(\epsilon\, x.\, P(x))}\ \texttt{someI2}$$

- We need to show some t exists satisfying P to reason about ϵ.

Example

$$\frac{\Gamma \vdash\ ? \in \{0, 1, 2, 3\} \qquad y \in \{0, 1, 2, 3\}, \Gamma \vdash y \leq 3}{\Gamma \vdash (\epsilon\, x.\, x \in \{0, 1, 2, 3\}) \leq 3}$$

- Very rarely need to reason about description operators directly.

Reasoning with Indefinite Descriptions

- Here's an introduction rule for indefinite description:

Introduction Rule

$$y \notin \mathit{fv}(\Gamma, P, Q) \; \frac{\Gamma \vdash P(t) \qquad P(y), \Gamma \vdash Q(y)}{\Gamma \vdash Q(\epsilon\, x.\, P(x))} \; \texttt{someI2}$$

- We need to show some t exists satisfying P to reason about ϵ.

Example

$$\frac{\Gamma \vdash 0 \in \{0,1,2,3\} \qquad y \in \{0,1,2,3\}, \Gamma \vdash y \leq 3}{\Gamma \vdash (\epsilon\, x.\, x \in \{0,1,2,3\}) \leq 3}$$

- Very rarely need to reason about description operators directly.

Reasoning with Indefinite Descriptions

- Here's an introduction rule for indefinite description:

Introduction Rule

$$y \notin \mathit{fv}(\Gamma, P, Q)\ \frac{\Gamma \vdash P(t) \qquad P(y), \Gamma \vdash Q(y)}{\Gamma \vdash Q(\epsilon\, x.\, P(x))}\ \texttt{someI2}$$

- We need to show some t exists satisfying P to reason about ϵ.

Example

$$\frac{\Gamma \vdash 0 \in \{0,1,2,3\} \qquad y \in \{0,1,2,3\}, \Gamma \vdash y \leq 3}{\Gamma \vdash (\epsilon\, x.\, x \in \{0,1,2,3\}) \leq 3}$$

- Very rarely need to reason about description operators directly.

Reasoning with Indefinite Descriptions

- Here's an introduction rule for indefinite description:

Introduction Rule

$$y \notin \mathit{fv}(\Gamma, P, Q) \;\; \frac{\Gamma \vdash P(t) \qquad P(y), \Gamma \vdash Q(y)}{\Gamma \vdash Q(\epsilon\, x.\, P(x))} \; \texttt{someI2}$$

- We need to show some t exists satisfying P to reason about ϵ.

Example

$$\frac{\Gamma \vdash 0 \in \{0, 1, 2, 3\} \qquad y \in \{0, 1, 2, 3\}, \Gamma \vdash y \leq 3}{\Gamma \vdash (\epsilon\, x.\, x \in \{0, 1, 2, 3\}) \leq 3}$$

- Very rarely need to reason about description operators directly.

Example: Indefinite Descriptions

Example: Indefinite Descriptions

Indefinite Description Equality Theorem (of `someI`)

Example: Indefinite Descriptions

Indefinite Description Equality Theorem (of `someI`)

$$x \notin \mathit{fv}(P, \Gamma)\ \frac{\Gamma \vdash P(a) \qquad P(x), \Gamma \vdash x = a}{\Gamma \vdash (\epsilon\, x.\, P\, x) = a}\ \texttt{some\_equality}$$

Example: Indefinite Descriptions

Indefinite Description Equality Theorem (of `someI`)

$$\frac{\Gamma \vdash P(a) \qquad P(x), \Gamma \vdash x = a}{\Gamma \vdash (\epsilon\, x.\, P\, x) = a}\; x \notin \mathit{fv}(P, \Gamma) \quad \texttt{some\_equality}$$

Example

```
lemma nat_less1_0: "(SOME x :: nat. x < 1) = 0"
proof (rule some_equality)
  show "(0::nat) < 1" by simp
next
  fix x :: nat
  assume "x < 1"
  thus "x = 0" by simp
qed
```

Example: Indefinite Descriptions

Indefinite Description Equality Theorem (of `someI`)

$$\frac{\Gamma \vdash P(a) \qquad P(x), \Gamma \vdash x = a}{\Gamma \vdash (\epsilon\, x.\, P\, x) = a}\; x \notin \mathit{fv}(P, \Gamma) \quad \texttt{some\_equality}$$

Example

```
lemma nat_less1_0: "(SOME x :: nat. x < 1) = 0"
proof (rule some_equality)
  show "(0::nat) < 1" by simp
next
  fix x :: nat
  assume "x < 1"
  thus "x = 0" by simp
qed
```

Example: Indefinite Descriptions

Indefinite Description Equality Theorem (of `someI`)

$$x \notin \mathit{fv}(P,\Gamma)\ \frac{\Gamma \vdash P(a) \qquad P(x),\Gamma \vdash x = a}{\Gamma \vdash (\epsilon\, x.\, P\, x) = a}\ \texttt{some\_equality}$$

Example

```
lemma nat_less1_0: "(SOME x :: nat. x < 1) = 0"
proof (rule some_equality)
  show "(0::nat) < 1" by simp
next
  fix x :: nat
  assume "x < 1"
  thus "x = 0" by simp
qed
```

Example: Indefinite Descriptions

Indefinite Description Equality Theorem (of `someI`)

$$x \notin \mathit{fv}(P, \Gamma)\ \frac{\Gamma \vdash P(a) \qquad P(x), \Gamma \vdash x = a}{\Gamma \vdash (\epsilon\, x.\, P\, x) = a}\ \texttt{some\_equality}$$

Example

```
lemma nat_less1_0: "(SOME x :: nat. x < 1) = 0"
proof (rule some_equality)
  show "(0::nat) < 1" by simp
next
  fix x :: nat
  assume "x < 1"
  thus "x = 0" by simp
qed
```

Example: Indefinite Descriptions

Indefinite Description Equality Theorem (of `someI`)

$$\frac{\Gamma \vdash P(a) \qquad P(x), \Gamma \vdash x = a}{\Gamma \vdash (\epsilon\, x.\, P\, x) = a}\ x \notin \mathit{fv}(P, \Gamma) \quad \texttt{some\_equality}$$

Example

```
lemma nat_less1_0: "(SOME x :: nat. x < 1) = 0"
proof (rule some_equality)
  show "(0::nat) < 1" by simp
next
  fix x :: nat
  assume "x < 1"
  thus "x = 0" by simp
qed
```

Example: Indefinite Descriptions

Indefinite Description Equality Theorem (of `someI`)

$$\frac{\Gamma \vdash P(a) \qquad P(x), \Gamma \vdash x = a}{\Gamma \vdash (\epsilon\, x.\, P\, x) = a}\; x \notin \mathit{fv}(P, \Gamma) \quad \texttt{some\_equality}$$

Example

```
lemma nat_less1_0: "(SOME x :: nat. x < 1) = 0"
proof (rule some_equality)
  show "(0::nat) < 1" by simp
next
  fix x :: nat
  assume "x < 1"
  thus "x = 0" by simp
qed
```

Example: Indefinite Descriptions

Indefinite Description Equality Theorem (of `someI`)

$$\frac{\Gamma \vdash P(a) \qquad P(x), \Gamma \vdash x = a}{\Gamma \vdash (\epsilon\, x.\, P\, x) = a}\; x \notin fv(P, \Gamma) \quad \texttt{some\_equality}$$

Example

```
lemma nat_less1_0: "(SOME x :: nat. x < 1) = 0"
proof (rule some_equality)
  show "(0::nat) < 1" by simp
next
  fix x :: nat
  assume "x < 1"
  thus "x = 0" by simp
qed
```

Example: Indefinite Descriptions

Indefinite Description Equality Theorem (of `someI`)

$$x \notin \mathit{fv}(P,\Gamma)\ \frac{\Gamma \vdash P(a) \qquad P(x), \Gamma \vdash x = a}{\Gamma \vdash (\epsilon\, x.\, P\, x) = a}\ \texttt{some\_equality}$$

Example

```
lemma nat_less1_0: "(SOME x :: nat. x < 1) = 0"
proof (rule some_equality)
  show "(0::nat) < 1" by simp
next
  fix x :: nat
  assume "x < 1"
  thus "x = 0" by simp
qed
```

Example: Indefinite Descriptions

Indefinite Description Equality Theorem (of `someI`)

$$x \notin \mathit{fv}(P, \Gamma) \; \frac{\Gamma \vdash P(a) \qquad P(x), \Gamma \vdash x = a}{\Gamma \vdash (\epsilon\, x.\, P\, x) = a} \; \texttt{some\_equality}$$

Example

```
lemma nat_less1_0: "(SOME x :: nat. x < 1) = 0"
proof (rule some_equality)
  show "(0::nat) < 1" by simp
next
  fix x :: nat
  assume "x < 1"
  thus "x = 0" by simp
qed
```

Example: Indefinite Descriptions

Indefinite Description Equality Theorem (of `someI`)

$$\frac{\Gamma \vdash P(a) \qquad P(x), \Gamma \vdash x = a}{\Gamma \vdash (\epsilon\, x.\, P\, x) = a}\; x \notin \mathit{fv}(P, \Gamma) \quad \texttt{some\_equality}$$

Example

```
lemma nat_less1_0: "(SOME x :: nat. x < 1) = 0"
proof (rule some_equality)
  show "(0::nat) < 1" by simp
next
  fix x :: nat
  assume "x < 1"
  thus "x = 0" by simp
qed
```

Example: Indefinite Descriptions

Indefinite Description Equality Theorem (of `someI`)

$$x \notin \mathit{fv}(P, \Gamma)\ \frac{\Gamma \vdash P(a) \qquad P(x), \Gamma \vdash x = a}{\Gamma \vdash (\epsilon\, x.\, P\ x) = a}\ \texttt{some\_equality}$$

Example

```
lemma nat_less1_0: "(SOME x :: nat. x < 1) = 0"
proof (rule some_equality)
  show "(0::nat) < 1" by simp
next
  fix x :: nat
  assume "x < 1"
  thus "x = 0" by simp
qed
```

Definite Descriptions

- Gives the unique value x described by P.
- Like definite article in natural language: “the cat sitting on my roof”.
- Meaningful only when there is precisely one x satisfying P.

Definite Descriptions

Definite Description Operator

$\iota\, x :: T.\, P\, x$ also written as $\mathit{THE}\, x :: T.\, P\, x$

- Gives the unique value x described by P.
- Like definite article in natural language: “the cat sitting on my roof”.
- Meaningful only when there is precisely one x satisfying P.

Definite Descriptions

Definite Description Operator

$\iota\, x :: T.\, P\, x$ also written as $THE\, x :: T.\, P\, x$

- Gives the unique value x described by P.
- Like definite article in natural language: "the cat sitting on my roof".
- Meaningful only when there is precisely one x satisfying P.

Definite Descriptions

Definite Description Operator

$\iota\, x :: T.\, P\, x$ also written as $THE\; x :: T.\, P\, x$

- Gives the unique value x described by P.
- Like definite article in natural language: “the cat sitting on my roof”.
- Meaningful only when there is precisely one x satisfying P.

Definite Descriptions

Definite Description Operator

$\iota\, x :: T.\, P\, x$ also written as $THE\; x :: T.\, P\, x$

- Gives the unique value x described by P.
- Like definite article in natural language: “the cat sitting on my roof”.
- Meaningful only when there is precisely one x satisfying P.

Definite Descriptions

Definite Description Operator

$\iota\, x :: T\,.\, P\, x$ also written as $THE\; x :: T\,.\, P\, x$

- Gives the unique value x described by P.
- Like definite article in natural language: “the cat sitting on my roof”.
- Meaningful only when there is precisely one x satisfying P.

Definite Descriptions

Definite Description Operator

$$\iota\, x :: T.\, P\, x \text{ also written as } THE\, x :: T.\, P\, x$$

- Gives the unique value x described by P.
- Like definite article in natural language: “the cat sitting on my roof”.
- Meaningful only when there is precisely one x satisfying P.

Introduction Rule

Definite Descriptions

Definite Description Operator

$\iota\, x :: T.\, P\, x$ also written as $THE\ x :: T.\, P\, x$

- Gives the unique value x described by P.
- Like definite article in natural language: "the cat sitting on my roof".
- Meaningful only when there is precisely one x satisfying P.

Introduction Rule

$$y \notin \mathit{fv}(\Gamma, P, Q)\ \frac{\Gamma \vdash P(t) \qquad P(y), \Gamma \vdash y = t \qquad P(y), \Gamma \vdash Q(y)}{\Gamma \vdash Q(\iota\, x.\, P(x))}\ \texttt{theI2}$$

Outline

1. Relationship between sets and types
2. Defining new types using subsets
3. Description operators
4. Benefits and limitations of types

Types vs. Sets

- Types and sets seem quite similar. Why have both?
  - 'a set vs. $\mathbb{P}\,A$, 'a × 'b vs. $A \times B$ etc.
- Types resolve problems in naive set theory.
- Russell's paradox:
  - Let $R = \{x \mid x \notin x\}$ then $R \in R \Leftrightarrow R \notin R$ inconsistent!

Types vs. Sets

- Types and sets seem quite similar. Why have both?

  $\texttt{'a set}$ vs. $\mathbb{P}\,A$, $\texttt{'a} \times \texttt{'b}$ vs. $\texttt{A} \times \texttt{B}$ etc.

- Types resolve problems in naive set theory.

- Russell's paradox:

  Let $R = \{x \mid x \notin x\}$ then $R \in R \Leftrightarrow R \notin R$ inconsistent!.

Types vs. Sets

- Types and sets seem quite similar. Why have both?

  `'a set` vs. $\mathbb{P}\,A$, `'a × 'b` vs. `A × B` etc.

- Types resolve problems in naive set theory.
- Russell's paradox:

  Let $R = \{x \mid x \notin x\}$ then $R \in R \Leftrightarrow R \notin R$ inconsistent!.

Types vs. Sets

- Types and sets seem quite similar. Why have both?

  `'a set` vs. $\mathbb{P}\,A$, `'a` $\times$ `'b` vs. `A` $\times$ `B` etc.

- Types resolve problems in naive set theory.
- Russell's paradox:

  Let $R = \{x \mid x \notin x\}$ then $R \in R \Leftrightarrow R \notin R$ inconsistent!.

Types vs. Sets

- Types and sets seem quite similar. Why have both?

  `'a set` vs. $\mathbb{P}\,A$, `'a × 'b` vs. `A × B` etc.

- Types resolve problems in naive set theory.
- Russell's paradox:

  Let $R = \{x \mid x \notin x\}$ then $R \in R \Leftrightarrow R \notin R$ inconsistent!.

Types vs. Sets

- Types and sets seem quite similar. Why have both?

  `'a set` vs. $\mathbb{P}\, A$, `'a × 'b` vs. `A × B` etc.

- Types resolve problems in naive set theory.
- Russell's paradox:

  Let $R = \{x \mid x \notin x\}$ then $R \in R \Leftrightarrow R \notin R$ inconsistent!.

Russell's Paradox (Proof Attempt)

↯

$\Gamma \vdash R \in R$

`not_not`

$\Gamma \vdash \neg(R \notin R)$

`mem_Collect_eq`

$\Gamma \vdash R \notin \{x \mid x \notin x\}$

`R_def`

$\Gamma \vdash R \notin R$

β-reduction

$\Gamma \vdash (\lambda x.\, x \notin x)\, R$

`mem_Collect_eq`

$\Gamma \vdash R \in \{x \mid x \notin x\}$

`R_def`

$\Gamma \vdash R \in R$

Russell's Paradox (Proof Attempt)

$$\lightning$$

$$\Gamma \vdash R \in R$$

`not_not`

$$\Gamma \vdash \neg(R \notin R)$$

`mem_Collect_eq`

$$\Gamma \vdash R \notin \{x \mid x \notin x\}$$

`R_def`

$$\Gamma \vdash R \notin R$$

β-reduction

$$\Gamma \vdash (\lambda\, x.\, x \notin x)\; R$$

`mem_Collect_eq`

$$\Gamma \vdash R \in \{x \mid x \notin x\}$$

`R_def`

$$\Gamma \vdash R \in R$$

Russell's Paradox (Proof Attempt)

$$\lightning$$

$$\Gamma \vdash R \in R$$

`not_not`

$$\Gamma \vdash \neg(R \notin R)$$

`mem_Collect_eq`

$$\Gamma \vdash R \notin \{x \mid x \notin x\}$$

`R_def`

$$\Gamma \vdash R \notin R$$

β-reduction

$$\Gamma \vdash (\lambda\, x.\, x \notin x)\; R$$

`mem_Collect_eq`

$$\Gamma \vdash R \in \{x \mid x \notin x\}$$

$$\frac{}{\Gamma \vdash R \in R}\;\texttt{R\_def}$$

Russell's Paradox (Proof Attempt)

$$\lightning$$

$$\Gamma \vdash R \in R$$

not_not

$$\Gamma \vdash \neg(R \notin R)$$

mem_Collect_eq

$$\Gamma \vdash R \notin \{x \mid x \notin x\}$$

R_def

$$\Gamma \vdash R \notin R$$

β-reduction

$$\Gamma \vdash (\lambda\, x.\, x \notin x)\; R$$

mem_Collect_eq

$$\dfrac{\Gamma \vdash R \in \{x \mid x \notin x\}}{\Gamma \vdash R \in R}\;\texttt{R\_def}$$

Russell's Paradox (Proof Attempt)

$$\lightning$$

$$\Gamma \vdash R \in R$$

not_not

$$\Gamma \vdash \neg(R \notin R)$$

mem_Collect_eq

$$\Gamma \vdash R \notin \{x \mid x \notin x\}$$

R_def

$$\Gamma \vdash R \notin R$$

β-reduction

$$\Gamma \vdash (\lambda\, x.\, x \notin x)\; R$$

$$\dfrac{\dfrac{}{\Gamma \vdash R \in \{x \mid x \notin x\}}\ \texttt{mem\_Collect\_eq}}{\Gamma \vdash R \in R}\ \texttt{R\_def}$$

Russell's Paradox (Proof Attempt)

$$\lightning$$

$$\Gamma \vdash R \in R$$

not_not

$$\Gamma \vdash \neg(R \notin R)$$

mem_Collect_eq

$$\Gamma \vdash R \notin \{x \mid x \notin x\}$$

R_def

$$\Gamma \vdash R \notin R$$

β-reduction

$$\dfrac{\dfrac{\Gamma \vdash (\lambda\, x.\, x \notin x)\; R}{\Gamma \vdash R \in \{x \mid x \notin x\}}\;\texttt{mem\_Collect\_eq}}{\Gamma \vdash R \in R}\;\texttt{R\_def}$$

Russell's Paradox (Proof Attempt)

$$\lightning$$

$$\Gamma \vdash R \in R$$

`not_not`

$$\Gamma \vdash \neg(R \notin R)$$

`mem_Collect_eq`

$$\Gamma \vdash R \notin \{x \mid x \notin x\}$$

`R_def`

$$\Gamma \vdash R \notin R$$

$$\dfrac{}{\Gamma \vdash (\lambda\, x.\, x \notin x)\; R}\ \beta\text{-reduction}$$

$$\dfrac{\Gamma \vdash (\lambda\, x.\, x \notin x)\; R}{\Gamma \vdash R \in \{x \mid x \notin x\}}\ \texttt{mem\_Collect\_eq}$$

$$\dfrac{\Gamma \vdash R \in \{x \mid x \notin x\}}{\Gamma \vdash R \in R}\ \texttt{R\_def}$$

Russell's Paradox (Proof Attempt)

$$\lightning$$

$$\Gamma \vdash R \in R$$

`not_not`

$$\Gamma \vdash \neg(R \notin R)$$

`mem_Collect_eq`

$$\Gamma \vdash R \notin \{x \mid x \notin x\}$$

`R_def`

$$\dfrac{\dfrac{\dfrac{\Gamma \vdash R \notin R}{\Gamma \vdash (\lambda x.\, x \notin x)\; R}\ \beta\text{-reduction}}{\Gamma \vdash R \in \{x \mid x \notin x\}}\ \texttt{mem\_Collect\_eq}}{\Gamma \vdash R \in R}\ \texttt{R\_def}$$

Russell's Paradox (Proof Attempt)

$$\lightning$$

$$\Gamma \vdash R \in R$$

`not_not`

$$\Gamma \vdash \neg(R \notin R)$$

`mem_Collect_eq`

$$\Gamma \vdash R \notin \{x \mid x \notin x\}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma \vdash R \notin R}\;\texttt{R\_def}}{\Gamma \vdash (\lambda\, x.\, x \notin x)\; R}\;\beta\text{-reduction}}{\Gamma \vdash R \in \{x \mid x \notin x\}}\;\texttt{mem\_Collect\_eq}}{\Gamma \vdash R \in R}\;\texttt{R\_def}$$

Russell's Paradox (Proof Attempt)

$$\lightning$$

$$\Gamma \vdash R \in R$$

`not_not`

$$\Gamma \vdash \neg(R \notin R)$$

`mem_Collect_eq`

$$\cfrac{\cfrac{\cfrac{\cfrac{\Gamma \vdash R \notin \{x \mid x \notin x\}}{\Gamma \vdash R \notin R}\;\texttt{R\_def}}{\Gamma \vdash (\lambda\, x.\, x \notin x)\; R}\;\beta\text{-reduction}}{\Gamma \vdash R \in \{x \mid x \notin x\}}\;\texttt{mem\_Collect\_eq}}{\Gamma \vdash R \in R}\;\texttt{R\_def}$$

Russell's Paradox (Proof Attempt)

$$
\begin{array}{cl}
\lightning & \\
\Gamma \vdash R \in R & \\
\texttt{not\_not} & \\
\Gamma \vdash \neg(R \notin R) & \\
\hline
\Gamma \vdash R \notin \{x \mid x \notin x\} & \texttt{mem\_Collect\_eq} \\
\hline
\Gamma \vdash R \notin R & \texttt{R\_def} \\
\hline
\Gamma \vdash (\lambda x.\, x \notin x)\ R & \beta\text{-reduction} \\
\hline
\Gamma \vdash R \in \{x \mid x \notin x\} & \texttt{mem\_Collect\_eq} \\
\hline
\Gamma \vdash R \in R & \texttt{R\_def}
\end{array}
$$

Russell's Paradox (Proof Attempt)

$$\lightning$$

$$\Gamma \vdash R \in R$$

`not_not`

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Gamma \vdash \neg(R \notin R)}{\Gamma \vdash R \notin \{x \mid x \notin x\}}\ \texttt{mem\_Collect\_eq}}{\Gamma \vdash R \notin R}\ \texttt{R\_def}}{\Gamma \vdash (\lambda\, x.\, x \notin x)\ R}\ \beta\text{-reduction}}{\Gamma \vdash R \in \{x \mid x \notin x\}}\ \texttt{mem\_Collect\_eq}}{\Gamma \vdash R \in R}\ \texttt{R\_def}$$

Russell's Paradox (Proof Attempt)

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\lightning}{\Gamma \vdash R \in R}}{\Gamma \vdash \neg(R \notin R)}\,\texttt{not\_not}}{\Gamma \vdash R \notin \{x \mid x \notin x\}}\,\texttt{mem\_Collect\_eq}}{\Gamma \vdash R \notin R}\,\texttt{R\_def}}{\Gamma \vdash (\lambda\, x.\, x \notin x)\; R}\,\beta\text{-reduction}}{\Gamma \vdash R \in \{x \mid x \notin x\}}\,\texttt{mem\_Collect\_eq}}{\Gamma \vdash R \in R}\,\texttt{R\_def}
$$

Russell's Paradox (Proof Attempt)

$$\lightning$$

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Gamma \vdash R \in R}{\Gamma \vdash \neg(R \notin R)}\ \texttt{not\_not}}{\Gamma \vdash R \notin \{x \mid x \notin x\}}\ \texttt{mem\_Collect\_eq}}{\Gamma \vdash R \notin R}\ \texttt{R\_def}}{\Gamma \vdash (\lambda x.\, x \notin x)\ R}\ \beta\text{-reduction}}{\Gamma \vdash R \in \{x \mid x \notin x\}}\ \texttt{mem\_Collect\_eq}}{\Gamma \vdash R \in R}\ \texttt{R\_def}
$$

Russell's Paradox (Proof Attempt)

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\lightning}{\Gamma \vdash R \in R}}{\Gamma \vdash \neg(R \notin R)}\,\texttt{not\_not}}{\Gamma \vdash R \notin \{x \mid x \notin x\}}\,\texttt{mem\_Collect\_eq}}{\Gamma \vdash R \notin R}\,\texttt{R\_def}}{\Gamma \vdash (\lambda x.\, x \notin x)\; R}\,\beta\text{-reduction}}{\Gamma \vdash R \in \{x \mid x \notin x\}}\,\texttt{mem\_Collect\_eq}}{\Gamma \vdash R \in R}\,\texttt{R\_def}
$$

Russell’s Paradox (Proof Attempt)

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\lightning}{\Gamma \vdash R \in R}}{\Gamma \vdash \neg(R \notin R)}\;\texttt{not\_not}}{\Gamma \vdash R \notin \{x \mid x \notin x\}}\;\texttt{mem\_Collect\_eq}}{\Gamma \vdash R \notin R}\;\texttt{R\_def}}{\Gamma \vdash (\lambda x.\, x \notin x)\, R}\;\beta\text{-reduction}}{\Gamma \vdash R \in \{x \mid x \notin x\}}\;\texttt{mem\_Collect\_eq}}{\Gamma \vdash R \in R}\;\texttt{R\_def}
$$

Types vs. Sets

- Types and sets seem quite similar – why have both?
- Types resolve problems in naïve set theory.
- Russell's paradox: let $R = \{x \mid x \notin x\}$ then $R \in R \Leftrightarrow R \notin R$.
- Excluded by HOL since $x \in x$ and $x \notin x$ are both ill-typed.
- Sets are more flexible, e.g. we can have $A \cup B$ and $A \subseteq B$.
- But no equivalents for types.
- Type checking $x :: T$ is decidable, but $x \in A$ requires proof.
- In general, types improve automation by enforcing specific patterns.
- Conclusion: We need both. It takes experience to know which to use.

Types vs. Sets

- Types and sets seem quite similar – why have both?
- Types resolve problems in naïve set theory.
- Russell's paradox: let $R = \{x \mid x \notin x\}$ then $R \in R \Leftrightarrow R \notin R$.
- Excluded by HOL since $x \in x$ and $x \notin x$ are both ill-typed.
- Sets are more flexible, e.g. we can have $A \cup B$ and $A \subseteq B$
- But no equivalents for types.
- Type checking $x :: T$ is decidable, but $x \in A$ requires proof.
- In general, types improve automation by enforcing specific patterns.
- Conclusion: We need both. It takes experience to know which to use.

Types vs. Sets

- Types and sets seem quite similar – why have both?
- Types resolve problems in naïve set theory.
- Russell's paradox: let $R = \{x \mid x \notin x\}$ then $R \in R \Leftrightarrow R \notin R$.
- Excluded by HOL since $x \in x$ and $x \notin x$ are both ill-typed.
- Sets are more flexible, e.g. we can have $A \cup B$ and $A \subseteq B$
- But no equivalents for types.
- Type checking $x :: T$ is decidable, but $x \in A$ requires proof.
- In general, types improve automation by enforcing specific patterns.
- Conclusion: We need both. It takes experience to know which to use.

Types vs. Sets

- Types and sets seem quite similar – why have both?
- Types resolve problems in naïve set theory.
- Russell's paradox: let $R = \{x \mid x \notin x\}$ then $R \in R \Leftrightarrow R \notin R$.
- Excluded by HOL since $x \in x$ and $x \notin x$ are both ill-typed.
- Sets are more flexible, e.g. we can have $A \cup B$ and $A \subseteq B$
- But no equivalents for types.
- Type checking $x :: T$ is decidable, but $x \in A$ requires proof.
- In general, types improve automation by enforcing specific patterns.
- Conclusion: We need both. It takes experience to know which to use.

Types vs. Sets

- Types and sets seem quite similar – why have both?
- Types resolve problems in naïve set theory.
- Russell's paradox: let $R = \{x \mid x \notin x\}$ then $R \in R \Leftrightarrow R \notin R$.
- Excluded by HOL since $x \in x$ and $x \notin x$ are both ill-typed.
- Sets are more flexible, e.g. we can have $A \cup B$ and $A \subseteq B$
- But no equivalents for types.
- Type checking $x :: T$ is decidable, but $x \in A$ requires proof.
- In general, types improve automation by enforcing specific patterns.
- Conclusion: We need both. It takes experience to know which to use.

Types vs. Sets

- Types and sets seem quite similar – why have both?
- Types resolve problems in naïve set theory.
- Russell's paradox: let $R = \{x \mid x \notin x\}$ then $R \in R \Leftrightarrow R \notin R$.
- Excluded by HOL since $x \in x$ and $x \notin x$ are both ill-typed.
- Sets are more flexible, e.g. we can have $A \cup B$ and $A \subseteq B$
- But no equivalents for types.
- Type checking $x :: T$ is decidable, but $x \in A$ requires proof.
- In general, types improve automation by enforcing specific patterns.
- Conclusion: We need both. It takes experience to know which to use.

Types vs. Sets

- Types and sets seem quite similar – why have both?
- Types resolve problems in naïve set theory.
- Russell's paradox: let $R = \{x \mid x \notin x\}$ then $R \in R \Leftrightarrow R \notin R$.
- Excluded by HOL since $x \in x$ and $x \notin x$ are both ill-typed.
- Sets are more flexible, e.g. we can have $A \cup B$ and $A \subseteq B$
- But no equivalents for types.
- Type checking $x :: T$ is decidable, but $x \in A$ requires proof.
- In general, types improve automation by enforcing specific patterns.
- Conclusion: We need both. It takes experience to know which to use.

Types vs. Sets

- Types and sets seem quite similar – why have both?
- Types resolve problems in naïve set theory.
- Russell's paradox: let $R = \{x \mid x \notin x\}$ then $R \in R \Leftrightarrow R \notin R$.
- Excluded by HOL since $x \in x$ and $x \notin x$ are both ill-typed.
- Sets are more flexible, e.g. we can have $A \cup B$ and $A \subseteq B$
- But no equivalents for types.
- Type checking $x :: T$ is decidable, but $x \in A$ requires proof.
- In general, types improve automation by enforcing specific patterns.
- Conclusion: We need both. It takes experience to know which to use.

Types vs. Sets

- Types and sets seem quite similar – why have both?
- Types resolve problems in naïve set theory.
- Russell's paradox: let $R = \{x \mid x \notin x\}$ then $R \in R \Leftrightarrow R \notin R$.
- Excluded by HOL since $x \in x$ and $x \notin x$ are both ill-typed.
- Sets are more flexible, e.g. we can have $A \cup B$ and $A \subseteq B$
- But no equivalents for types.
- Type checking $x :: T$ is decidable, but $x \in A$ requires proof.
- In general, types improve automation by enforcing specific patterns.
- Conclusion: We need both. It takes experience to know which to use.

Types vs. Sets

- Types and sets seem quite similar – why have both?
- Types resolve problems in naïve set theory.
- Russell's paradox: let $R = \{x \mid x \notin x\}$ then $R \in R \Leftrightarrow R \notin R$.
- Excluded by HOL since $x \in x$ and $x \notin x$ are both ill-typed.
- Sets are more flexible, e.g. we can have $A \cup B$ and $A \subseteq B$
- But no equivalents for types.
- Type checking $x :: T$ is decidable, but $x \in A$ requires proof.
- In general, types improve automation by enforcing specific patterns.
- Conclusion: We need both. It takes experience to know which to use.

Conclusion

Next Lecture

- Automation and sledgehammer.

Conclusion

This Lecture

- Relationship between sets and types.
- Defining new types using subsets.
- Description operators.
- Benefits and limitations of types.

Next Lecture

- Automation and sledgehammer.

Conclusion

This Lecture

- Relationship between sets and types.
- Defining new types using subsets.
- Description operators.
- Benefits and limitations of types.

Next Lecture

- Automation and sledghammer.

Conclusion

This Lecture

- Relationship between sets and types.
- Defining new types using subsets.
- Description operators.
- Benefits and limitations of types.

Next Lecture

- Automation and sledgehammer.

Conclusion

This Lecture

- Relationship between sets and types.
- Defining new types using subsets.
- Description operators.
- Benefits and limitations of types.

Next Lecture

- Automation and sledgehammer

Conclusion

This Lecture

- Relationship between sets and types.
- Defining new types using subsets.
- Description operators.
- Benefits and limitations of types.

Next Lecture

- Automation and sledgehammer

Conclusion

This Lecture

- Relationship between sets and types.
- Defining new types using subsets.
- Description operators.
- Benefits and limitations of types.

Next Lecture

- Automation and sledghammer.

Conclusion

This Lecture

- Relationship between sets and types.
- Defining new types using subsets.
- Description operators.
- Benefits and limitations of types.

Next Lecture

- Automation and sledghammer.