Sets in Isabelle/HOL

Simon Foster   Jim Woodcock
University of York

18th August 2022

Overview

1. Set theory in Isabelle/HOL
2. Operators on sets
3. Finite sets
4. Uncomputable objects

Outline

1. Set theory in Isabelle/HOL
2. Operators on sets
3. Finite sets
4. Uncomputable objects

Collections of objects

- A set is any well-defined collection of objects.
- `'a set` – a set of elements drawn from the type `'a`.
- Small sets are defined by listing their elements (extension):

```
definition Oceans :: "ocean set" where
  "Oceans = {Atlantic, Arctic, Indian, Pacific}"
```

- In HOL, this is sugar for `insert :: 'a ⇒ 'a set ⇒ 'a set`.
- ```
  insert Atlantic
    (insert Arctic
      (insert Indian
        (insert Pacific {})))
  ```

$$\mathit{insert}\ x\ (\mathit{insert}\ x\ A) = \mathit{insert}\ x\ A \qquad \texttt{insert\_absorb2}$$

$$\mathit{insert}\ x\ (\mathit{insert}\ y\ A) = \mathit{insert}\ y\ (\mathit{insert}\ x\ A) \qquad \texttt{insert\_commute}$$

- Unlike a list, the occurrence and order of members is irrelevant.
  ```

Collections of objects

- A set is any well-defined collection of objects.
- `'a set` – a set of elements drawn from the type `'a`.
- Small sets are defined by listing their elements (extension):

```
definition Oceans :: "ocean set" where
  "Oceans = {Atlantic, Arctic, Indian, Pacific}"
```

- In HOL, this is sugar for `insert :: 'a ⇒ 'a set ⇒ 'a set`.
- ```
  insert Atlantic
    (insert Arctic
      (insert Indian
        (insert Pacific {})))
  ```

  insert x (insert x A) = insert x A

  insert x (insert y A) = insert y (insert x A)

- Unlike a list, the occurrence and order of members is irrelevant.
  ```

Collections of objects

- A set is any well-defined collection of objects.
- `'a set` – a set of elements drawn from the type `'a`.
- Small sets are defined by listing their elements (extension):

```
definition Oceans :: "ocean set" where
  "Oceans = {Atlantic, Arctic, Indian, Pacific}"
```

- In HOL, this is sugar for `insert :: 'a ⇒ 'a set ⇒ 'a set`.
- ```
  insert Atlantic
    (insert Arctic
      (insert Indian
        (insert Pacific {})))
  ```

$$insert\ x\ (insert\ x\ A) = insert\ x\ A \qquad \texttt{insert\_absorb}$$

$$insert\ x\ (insert\ y\ A) = insert\ y\ (insert\ x\ A) \qquad \texttt{insert\_commute}$$

- Unlike a list, the occurrence and order of members is irrelevant.
  ```

Collections of objects

- A set is any well-defined collection of objects.
- `'a set` – a set of elements drawn from the type `'a`.
- Small sets are defined by listing their elements (extension):

```
definition Oceans :: "ocean set" where
  "Oceans = {Atlantic, Arctic, Indian, Pacific}"
```

- In HOL, this is sugar for `insert :: 'a ⇒ 'a set ⇒ 'a set`.
- ```
  insert Atlantic
    (insert Arctic
      (insert Indian
        (insert Pacific {})))
  ```

$$insert\ x\ (insert\ x\ A) = insert\ x\ A \qquad \texttt{insert\_absorb}$$
$$insert\ x\ (insert\ y\ A) = insert\ y\ (insert\ x\ A) \qquad \texttt{insert\_commute}$$

- Unlike a list, the occurrence and order of members is irrelevant.
  ```

Collections of objects

- A set is any well-defined collection of objects.
- `'a set` – a set of elements drawn from the type `'a`.
- Small sets are defined by listing their elements (extension):

```
definition Oceans :: "ocean set" where
  "Oceans = {Atlantic, Arctic, Indian, Pacific}"
```

- In HOL, this is sugar for `insert :: 'a ⇒ 'a set ⇒ 'a set`.
-
```
insert Atlantic
  (insert Arctic
    (insert Indian
      (insert Pacific {})))
```

$insert\ x\ (insert\ x\ A) = insert\ x\ A$ insert_absorb

$insert\ x\ (insert\ y\ A) = insert\ y\ (insert\ x\ A)$ insert_commute

- Unlike a list, the occurrence and order of members is irrelevant.

Collections of objects

- A set is any well-defined collection of objects.
- `'a set` – a set of elements drawn from the type `'a`.
- Small sets are defined by listing their elements (extension):

```
definition Oceans :: "ocean set" where
  "Oceans = {Atlantic, Arctic, Indian, Pacific}"
```

- In HOL, this is sugar for `insert :: 'a ⇒ 'a set ⇒ 'a set`.
-
```
insert Atlantic
  (insert Arctic
    (insert Indian
      (insert Pacific {})))
```

$$\mathit{insert}\, x\, (\mathit{insert}\, x\, A) = \mathit{insert}\, x\, A \qquad \texttt{insert\_absorb2}$$

$$\mathit{insert}\, x\, (\mathit{insert}\, y\, A) = \mathit{insert}\, y\, (\mathit{insert}\, x\, A) \qquad \texttt{insert\_commute}$$

- Unlike a list, the occurrence and order of members is irrelevant.

Collections of objects

- A set is any well-defined collection of objects.
- `'a set` – a set of elements drawn from the type `'a`.
- Small sets are defined by listing their elements (extension):

```
definition Oceans :: "ocean set" where
  "Oceans = {Atlantic, Arctic, Indian, Pacific}"
```

- In HOL, this is sugar for `insert :: 'a ⇒ 'a set ⇒ 'a set`.
- ```
  insert Atlantic
    (insert Arctic
      (insert Indian
        (insert Pacific {})))
  ```

$$\textit{insert}\, x\, (\textit{insert}\, x\, A) = \textit{insert}\, x\, A \qquad \texttt{insert\_absorb2}$$

$$\textit{insert}\, x\, (\textit{insert}\, y\, A) = \textit{insert}\, y\, (\textit{insert}\, x\, A) \qquad \texttt{insert\_commute}$$

- Unlike a list, the occurrence and order of members is irrelevant.
  ```

Collections of objects

- A set is any well-defined collection of objects.
- `'a set` – a set of elements drawn from the type `'a`.
- Small sets are defined by listing their elements (extension):

```
definition Oceans :: "ocean set" where
  "Oceans = {Atlantic, Arctic, Indian, Pacific}"
```

- In HOL, this is sugar for `insert :: 'a ⇒ 'a set ⇒ 'a set`.
- ```
  insert Atlantic
    (insert Arctic
      (insert Indian
        (insert Pacific {})))
  ```

$$\textit{insert}\, x\, (\textit{insert}\, x\, A) = \textit{insert}\, x\, A \qquad \texttt{insert\_absorb2}$$

$$\textit{insert}\, x\, (\textit{insert}\, y\, A) = \textit{insert}\, y\, (\textit{insert}\, x\, A) \qquad \texttt{insert\_commute}$$

- Unlike a list, the occurrence and order of members is irrelevant.
  ```

Membership and extension

- Membership $x \in S$: x is an element of set S. Write $\neg(x \in S)$ as $x \notin S$.
- Extensionality
- Subset
- Empty set $\{\}$ (mathematically $\emptyset$):

Membership and extension

- Membership $x \in S$: x is an element of set S. Write $\neg(x \in S)$ as $x \notin S$.

  $t \in \{u_1, \ldots, u_n\} \Leftrightarrow t = u_1 \vee \ldots \vee t = u_n$

  $t \in \textit{insert}\ u\ S \Leftrightarrow t = u \vee t \in S$ `insert_iff`

- Extensionality

  $A = B \Leftrightarrow (\forall x. (x \in A) \Leftrightarrow (x \in B))$ `set_eq_iff`

- Subset

  $A \subseteq B \Leftrightarrow \forall x \in A. x \in B$ `subset_eq`

  $A = B \Leftrightarrow A \subseteq B \wedge B \subseteq A$ `set_eq_subset`

- Empty set $\{\}$ (mathematically $\emptyset$):

  $(x \in \{\}) = \textit{False}$ `empty_iff`

Membership and extension

- Membership $x \in S$: x is an element of set S. Write $\neg(x \in S)$ as $x \notin S$.

$$t \in \{\, u_1, \ldots, u_n \,\} \Leftrightarrow t = u_1 \vee \ldots \vee t = u_n$$

$$t \in \textit{insert}\ u\ S \Leftrightarrow t = u \vee t \in S \qquad \texttt{insert\_iff}$$

- Extensionality

$$A = B \Leftrightarrow \forall x.\, (x \in A) = (x \in B)$$

- Subset

$$A \subseteq B \Leftrightarrow \forall t \in A.\, t \in B$$

$$A = B \Leftrightarrow A \subseteq B \wedge B \subseteq A$$

- Empty set $\{\}$ (mathematically $\emptyset$):

$$(x \in \{\}) = \textit{False}$$

Membership and extension

- Membership $x \in S$: x is an element of set S. Write $\neg(x \in S)$ as $x \notin S$.

  $t \in \{\, u_1, \ldots, u_n \,\} \;\Leftrightarrow\; t = u_1 \vee \ldots \vee t = u_n$

  $t \in \textit{insert}\ u\ S \;\Leftrightarrow\; t = u \vee t \in S$ `insert_iff`

- Extensionality

  $A = B \;\Leftrightarrow\; (\forall x.\ (x \in A) = (x \in B))$ `set_eq_iff`

- Subset

  $A \subseteq B \;\Leftrightarrow\; \forall x \in A.\ x \in B$ `subset_eq`

  $A = B \;\Leftrightarrow\; A \subseteq B \wedge B \subseteq A$ `set_eq_subset`

- Empty set $\{\}$ (mathematically $\emptyset$):

  $(x \in \{\}) = \textit{False}$ `empty_iff`

Membership and extension

- **Membership** $x \in S$: x is an element of set S. Write $\neg(x \in S)$ as $x \notin S$.

  $$t \in \{\, u_1, \ldots, u_n \,\} \Leftrightarrow t = u_1 \vee \ldots \vee t = u_n$$

  $$t \in \textit{insert}\ u\ S \Leftrightarrow t = u \vee t \in S \qquad \texttt{insert\_iff}$$

- **Extensionality**

  $$A = B \Leftrightarrow (\forall x. (x \in A) \Leftrightarrow (x \in B)) \qquad \texttt{set\_eq\_iff}$$

- **Subset**

  $$A \subseteq B = \forall x \in A.\ x \in B \qquad \texttt{subset\_eq}$$

  $$A = B \Leftrightarrow A \subseteq B \wedge B \subseteq A \qquad \texttt{set\_eq\_subset}$$

- **Empty set** $\{\}$ (mathematically $\emptyset$):

  $$(x \in \{\}) = \textit{False} \qquad \texttt{empty\_iff}$$

Membership and extension

- Membership $x \in S$: x is an element of set S. Write $\neg(x \in S)$ as $x \notin S$.

$$t \in \{\, u_1, \ldots, u_n \,\} \Leftrightarrow t = u_1 \vee \ldots \vee t = u_n$$

$$t \in \textit{insert}\ u\ S \Leftrightarrow t = u \vee t \in S \qquad \texttt{insert\_iff}$$

- Extensionality

$$A = B \Leftrightarrow (\forall x.\, (x \in A) \Leftrightarrow (x \in B)) \qquad \texttt{set\_eq\_iff}$$

- Subset

$$A \subseteq B \Leftrightarrow \forall x \in A.\ x \in B \qquad \texttt{subset\_eq}$$

$$A = B \Leftrightarrow A \subseteq B \wedge B \subseteq A \qquad \texttt{set\_eq\_subset}$$

- Empty set $\{\}$ (mathematically $\emptyset$):

$$(x \in \{\}) = \textit{False} \qquad \texttt{empty\_iff}$$

Membership and extension

- Membership $x \in S$: x is an element of set S. Write $\neg(x \in S)$ as $x \notin S$.

  $$t \in \{\, u_1, \ldots, u_n \,\} \;\Leftrightarrow\; t = u_1 \vee \ldots \vee t = u_n$$

  $$t \in \textit{insert}\ u\ S \;\Leftrightarrow\; t = u \vee t \in S$$ `insert_iff`

- Extensionality

  $$A = B \;\Leftrightarrow\; (\forall\, x.\, (x \in A) \;\Leftrightarrow\; (x \in B))$$ `set_eq_iff`

- Subset

  $$A \subseteq B \;\Leftrightarrow\; \forall\, x \in A.\, x \in B$$ `subset_eq`

  $$A = B \;\Leftrightarrow\; A \subseteq B \wedge B \subseteq A$$ `set_eq_subset`

- Empty set $\{\}$ (mathematically $\emptyset$):

  $$(x \in \{\}) = \textit{False}$$ `empty_iff`

Membership and extension

- Membership $x \in S$: x is an element of set S. Write $\neg(x \in S)$ as $x \notin S$.

$$t \in \{\, u_1, \ldots, u_n \,\} \;\Leftrightarrow\; t = u_1 \vee \ldots \vee t = u_n$$

$$t \in \textit{insert}\ u\ S \;\Leftrightarrow\; t = u \vee t \in S \qquad \texttt{insert\_iff}$$

- Extensionality

$$A = B \;\Leftrightarrow\; (\forall\, x.\, (x \in A) \;\Leftrightarrow\; (x \in B)) \qquad \texttt{set\_eq\_iff}$$

- Subset

$$A \subseteq B \;\Leftrightarrow\; \forall\, x \in A.\, x \in B \qquad \texttt{subset\_eq}$$

$$A = B \;\Leftrightarrow\; A \subseteq B \wedge B \subseteq A \qquad \texttt{set\_eq\_subset}$$

- Empty set $\{\}$ (mathematically $\emptyset$):

$$(x \in \{\}) = \textit{False} \qquad \texttt{empty\_iff}$$

Membership and extension

- Membership $x \in S$: x is an element of set S. Write $\neg(x \in S)$ as $x \notin S$.

$$t \in \{\, u_1, \ldots, u_n \,\} \;\Leftrightarrow\; t = u_1 \vee \ldots \vee t = u_n$$

$$t \in \textit{insert}\ u\ S \;\Leftrightarrow\; t = u \vee t \in S$$ `insert_iff`

- Extensionality

$$A = B \;\Leftrightarrow\; (\forall\, x.\, (x \in A) \;\Leftrightarrow\; (x \in B))$$ `set_eq_iff`

- Subset

$$A \subseteq B \;\Leftrightarrow\; \forall\, x \in A.\, x \in B$$ `subset_eq`

$$A = B \;\Leftrightarrow\; A \subseteq B \wedge B \subseteq A$$ `set_eq_subset`

- Empty set $\{\}$ (mathematically $\emptyset$):

$$(x \in \{\}) = \textit{False}$$ `empty_iff`

Membership and extension

- Membership $x \in S$: x is an element of set S. Write $\neg(x \in S)$ as $x \notin S$.

$$t \in \{\, u_1, \ldots, u_n \,\} \;\Leftrightarrow\; t = u_1 \lor \ldots \lor t = u_n$$

$$t \in \textit{insert}\ u\ S \;\Leftrightarrow\; t = u \lor t \in S \qquad \texttt{insert\_iff}$$

- Extensionality

$$A = B \;\Leftrightarrow\; (\forall\, x.\, (x \in A) \;\Leftrightarrow\; (x \in B)) \qquad \texttt{set\_eq\_iff}$$

- Subset

$$A \subseteq B \;\Leftrightarrow\; \forall\, x \in A.\, x \in B \qquad \texttt{subset\_eq}$$

$$A = B \;\Leftrightarrow\; A \subseteq B \land B \subseteq A \qquad \texttt{set\_eq\_subset}$$

- Empty set $\{\}$ (mathematically $\emptyset$):

$$(c \in \{\}) = \textit{False} \qquad \texttt{empty\_iff}$$

Membership and extension

- Membership $x \in S$: x is an element of set S. Write $\neg(x \in S)$ as $x \notin S$.

  $$t \in \{\, u_1, \ldots, u_n \,\} \;\Leftrightarrow\; t = u_1 \vee \ldots \vee t = u_n$$

  $$t \in \textit{insert}\ u\ S \;\Leftrightarrow\; t = u \vee t \in S \qquad \texttt{insert\_iff}$$

- Extensionality

  $$A = B \;\Leftrightarrow\; (\forall\, x.\, (x \in A) \;\Leftrightarrow\; (x \in B)) \qquad \texttt{set\_eq\_iff}$$

- Subset

  $$A \subseteq B \;\Leftrightarrow\; \forall\, x \in A.\, x \in B \qquad \texttt{subset\_eq}$$

  $$A = B \;\Leftrightarrow\; A \subseteq B \wedge B \subseteq A \qquad \texttt{set\_eq\_subset}$$

- Empty set $\{\}$ (mathematically $\emptyset$):

  $$(c \in \{\}) = \textit{False} \qquad \texttt{empty\_iff}$$

Set Deduction Rules (Selection)

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

$$\frac{\Gamma \vdash t \in A \qquad t \in B, \Gamma \vdash P}{A \subseteq B \vdash P}\ \texttt{subsetD}$$

$$\frac{\Gamma \vdash A \subseteq B \qquad \Gamma \vdash B \subseteq A}{\Gamma \vdash A = B}\ \texttt{equalityI}$$

$$\frac{t \in A, t \in B, \Gamma \vdash P \qquad t \notin A, t \notin B, \Gamma \vdash P}{A = B, \Gamma \vdash P}\ \texttt{equalityCE}$$

- Subset and equality proofs can be automated with `blast` and `auto`.

Set Deduction Rules (Selection)

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

$$\frac{\Gamma \vdash t \in A \qquad t \in B, \Gamma \vdash P}{A \subseteq B \vdash P}\ \texttt{subsetD}$$

$$\frac{\Gamma \vdash A \subseteq B \qquad \Gamma \vdash B \subseteq A}{\Gamma \vdash A = B}\ \texttt{equalityI}$$

$$\frac{t \in A, t \in B, \Gamma \vdash P \qquad t \notin A, t \notin B, \Gamma \vdash P}{A = B, \Gamma \vdash P}\ \texttt{equalityCE}$$

- Subset and equality proofs can be automated with `blast` and `auto`.

Set Deduction Rules (Selection)

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

$$\frac{\Gamma \vdash t \in A \qquad t \in B, \Gamma \vdash P}{A \subseteq B \vdash P}\ \texttt{subsetD}$$

$$\frac{\Gamma \vdash A \subseteq B \qquad \Gamma \vdash B \subseteq A}{\Gamma \vdash A = B}\ \texttt{equalityI}$$

$$\frac{t \in A, t \in B, \Gamma \vdash P \qquad t \notin A, t \notin B, \Gamma \vdash P}{A = B, \Gamma \vdash P}\ \texttt{equalityCE}$$

- Subset and equality proofs can be automated with `blast` and `auto`.

Set Deduction Rules (Selection)

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

$$\frac{\Gamma \vdash t \in A \qquad t \in B, \Gamma \vdash P}{A \subseteq B \vdash P}\ \texttt{subsetD}$$

$$\frac{\Gamma \vdash A \subseteq B \qquad \Gamma \vdash B \subseteq A}{\Gamma \vdash A = B}\ \texttt{equalityI}$$

$$\frac{t \in A, t \in B, \Gamma \vdash P \qquad t \notin A, t \notin B, \Gamma \vdash P}{A = B, \Gamma \vdash P}\ \texttt{equalityCE}$$

- Subset and equality proofs can be automated with `blast` and `auto`.

Set Deduction Rules (Selection)

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

$$\frac{\Gamma \vdash t \in A \qquad t \in B, \Gamma \vdash P}{A \subseteq B \vdash P}\ \texttt{subsetD}$$

$$\frac{\Gamma \vdash A \subseteq B \qquad \Gamma \vdash B \subseteq A}{\Gamma \vdash A = B}\ \texttt{equalityI}$$

$$\frac{t \in A, t \in B, \Gamma \vdash P \qquad t \notin A, t \notin B, \Gamma \vdash P}{A = B, \Gamma \vdash P}\ \texttt{equalityCE}$$

- Subset and equality proofs can be automated with `blast` and `auto`.

Set Deduction Rules (Selection)

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

$$\frac{\Gamma \vdash t \in A \qquad t \in B, \Gamma \vdash P}{A \subseteq B \vdash P}\ \texttt{subsetD}$$

$$\frac{\Gamma \vdash A \subseteq B \qquad \Gamma \vdash B \subseteq A}{\Gamma \vdash A = B}\ \texttt{equalityI}$$

$$\frac{t \in A, t \in B, \Gamma \vdash P \qquad t \notin A, t \notin B, \Gamma \vdash P}{A = B, \Gamma \vdash P}\ \texttt{equalityCE}$$

- Subset and equality proofs can be automated with `blast` and `auto`.

Bounded Quantifiers

- Sets can be used to bound the quantifiers.
- $\forall\, x \in A.\, P(x)$ – for every element of A predicate P holds.
- $\exists\, x \in A.\, P(x)$ – there is an element of A such that P holds.
- In HOL, these are syntactic sugar for regular quantification:

$$(\forall\, x \in A.\, P(x)) \quad \equiv \quad (\forall\, x.\, x \in A \longrightarrow P(x))$$

$$(\exists\, x \in A.\, P(x)) \quad \equiv \quad (\exists\, x.\, x \in A \wedge P(x))$$

Bounded Quantifiers

- Sets can be used to bound the quantifiers.
- $\forall x \in A.\, P(x)$ – for every element of A predicate P holds.
- $\exists x \in A.\, P(x)$ – there is an element of A such that P holds.
- In HOL, these are syntactic sugar for regular quantification:

$$(\forall x \in A.\, P(x)) \quad \equiv \quad (\forall x.\, x \in A \longrightarrow P(x))$$

$$(\exists x \in A.\, P(x)) \quad \equiv \quad (\exists x.\, x \in A \wedge P(x))$$

Bounded Quantifiers

- Sets can be used to bound the quantifiers.
- $\forall\, x \in A.\, P(x)$ – for every element of A predicate P holds.
- $\exists\, x \in A.\, P(x)$ – there is an element of A such that P holds.
- In HOL, these are syntactic sugar for regular quantification:

$$(\forall\, x \in A.\, P(x)) \quad \equiv \quad (\forall\, x.\, x \in A \longrightarrow P(x))$$

$$(\exists\, x \in A.\, P(x)) \quad \equiv \quad (\exists\, x.\, x \in A \wedge P(x))$$

Bounded Quantifiers

- Sets can be used to bound the quantifiers.
- $\forall\, x \in A.\, P(x)$ – for every element of A predicate P holds.
- $\exists\, x \in A.\, P(x)$ – there is an element of A such that P holds.
- In HOL, these are syntactic sugar for regular quantification:

$$(\forall\, x \in A.\, P(x)) \quad \equiv \quad (\forall\, x.\, x \in A \longrightarrow P(x))$$

$$(\exists\, x \in A.\, P(x)) \quad \equiv \quad (\exists\, x.\, x \in A \wedge P(x))$$

Bounded Quantifiers

- Sets can be used to bound the quantifiers.
- $\forall\, x \in A.\, P(x)$ – for every element of A predicate P holds.
- $\exists\, x \in A.\, P(x)$ – there is an element of A such that P holds.
- In HOL, these are syntactic sugar for regular quantification:

$$(\forall\, x \in A.\, P(x)) \equiv (\forall\, x.\, x \in A \longrightarrow P(x))$$

$$(\exists\, x \in A.\, P(x)) \equiv (\exists\, x.\, x \in A \wedge P(x))$$

Bounded Quantifiers

- Sets can be used to bound the quantifiers.
- $\forall\, x \in A.\, P(x)$ – for every element of A predicate P holds.
- $\exists\, x \in A.\, P(x)$ – there is an element of A such that P holds.
- In HOL, these are syntactic sugar for regular quantification:

$$(\forall\, x \in A.\, P(x)) \quad \equiv \quad (\forall\, x.\, x \in A \longrightarrow P(x))$$

$$(\exists\, x \in A.\, P(x)) \quad \equiv \quad (\exists\, x.\, x \in A \wedge P(x))$$

Bounded Quantifiers

- Sets can be used to bound the quantifiers.
- $\forall\, x \in A.\, P(x)$ – for every element of A predicate P holds.
- $\exists\, x \in A.\, P(x)$ – there is an element of A such that P holds.
- In HOL, these are syntactic sugar for regular quantification:

$$(\forall\, x \in A.\, P(x)) \quad \equiv \quad (\forall\, x.\, x \in A \longrightarrow P(x))$$

$$(\exists\, x \in A.\, P(x)) \quad \equiv \quad (\exists\, x.\, x \in A \wedge P(x))$$

Set comprehension

- Elements of set S satisfying property P (maths: $\{x \in S \mid P(x)\}$):

  $t \in \{x \in S.\ P(x)\} \Longleftrightarrow t \in S \wedge P(t)$

- Term comprehension: set constructed from particular terms:

  $t \in \{f(x) \mid x.\ P(x)\} \Longleftrightarrow (\exists x.\ P(x) \wedge t = f(x))$

- Set comprehension is the axiomatic constructor for sets:

  `Collect :: ('a ⇒ bool) ⇒ 'a set`

  `{x. P(x)} = Collect P`
  `a ∈ (Collect P) = P a`   `mem_Collect_eq`
  `Collect (λx. x ∈ A) = A`   `Collect_mem_eq`

Set comprehension

- Elements of set S satisfying property P (maths: $\{x \in S \mid P(x)\}$):

  $$t \in \{x \in S.\, P(x)\} \Leftrightarrow t \in S \wedge P(t)$$

- Term comprehension: set constructed from particular terms:

  $$t \in \{f(x) \mid x.\, P(x)\} \Leftrightarrow (\exists x.\, P(x) \wedge t = f(x))$$

- Set comprehension is the axiomatic constructor for sets:

  $\texttt{Collect} :: (\texttt{'a} \Rightarrow \texttt{bool}) \Rightarrow \texttt{'a set}$

  $\{x.\, P(x)\} = \texttt{Collect } P$

  $a \in (\texttt{Collect } P) \Leftrightarrow P\ a$ — mem_Collect_eq

  $\texttt{Collect}\,(\lambda x.\, x \in A) = A$ — Collect_mem_eq

Set comprehension

- Elements of set S satisfying property P (maths: $\{x \in S \mid P(x)\}$):

$$t \in \{\, x \in S.\, P(x)\} \;\Leftrightarrow\; t \in S \wedge P(t)$$

- Term comprehension: set constructed from particular terms:

$$t \in \{f(x) \mid x.\, P(x)\} \;\Leftrightarrow\; (\exists x.\, P(x) \wedge t = f(x))$$

- Set comprehension is the axiomatic constructor for sets:

```
Collect :: ('a ⇒ bool) ⇒ 'a set
```

```
{x. P(x)} = Collect P
a ∈ (Collect P) ⟷ P a                      mem_Collect_eq
Collect (λx. x ∈ A) = A                    Collect_mem_eq
```

Set comprehension

- Elements of set S satisfying property P (maths: $\{x \in S \mid P(x)\}$):

$$t \in \{\, x \in S.\, P(x) \} \;\Leftrightarrow\; t \in S \wedge P(t)$$

- Term comprehension: set constructed from particular terms:

$$t \in \{\, f(x) \mid x.\, P(x) \,\} \;\Leftrightarrow\; (\,\exists\, x.\, P(x) \wedge t = f(x)\,)$$

- Set comprehension is the axiomatic constructor for sets:

Collect :: ('a ⇒ bool) ⇒ 'a set

$\{x.\ P(x)\}$ = Collect P

a ∈ (Collect P) ⟷ P a   mem_Collect_eq

Collect($\lambda x.\ x \in A$) = A   Collect_mem_eq

Set comprehension

- Elements of set S satisfying property P (maths: $\{x \in S \mid P(x)\}$):

$$t \in \{\, x \in S.\, P(x) \} \;\Leftrightarrow\; t \in S \wedge P(t)$$

- Term comprehension: set constructed from particular terms:

$$t \in \{\, f(x) \mid x.\, P(x)\,\} \;\Leftrightarrow\; (\,\exists\, x.\, P(x) \wedge t = f(x)\,)$$

- Set comprehension is the axiomatic constructor for sets:

Set comprehension

- Elements of set S satisfying property P (maths: $\{x \in S \mid P(x)\}$):

$$t \in \{x \in S.\, P(x)\} \;\Leftrightarrow\; t \in S \wedge P(t)$$

- Term comprehension: set constructed from particular terms:

$$t \in \{\, f(x) \mid x.\, P(x)\,\} \;\Leftrightarrow\; (\,\exists\, x.\, P(x) \wedge t = f(x)\,)$$

- Set comprehension is the axiomatic constructor for sets:

$\texttt{Collect} :: (\texttt{'a} \Rightarrow \texttt{bool}) \Rightarrow \texttt{'a set}$

$\{x.\, P(x)\} = \texttt{Collect P}$

$\texttt{a} \in (\texttt{Collect P}) \;\Leftrightarrow\; \texttt{P a}$ mem_Collect_eq

$\texttt{Collect}\,(\lambda\,\texttt{x}.\,\texttt{x} \in \texttt{A}) = \texttt{A}$ Collect_mem_eq

Set comprehension

- Elements of set S satisfying property P (maths: $\{x \in S \mid P(x)\}$):

$$t \in \{x \in S.\, P(x)\} \;\Leftrightarrow\; t \in S \wedge P(t)$$

- Term comprehension: set constructed from particular terms:

$$t \in \{\, f(x) \mid x.\, P(x)\,\} \;\Leftrightarrow\; (\,\exists\, x.\, P(x) \wedge t = f(x)\,)$$

- Set comprehension is the axiomatic constructor for sets:

$$\texttt{Collect} :: (\texttt{'a} \Rightarrow \texttt{bool}) \Rightarrow \texttt{'a set}$$

$$\{x.\, P(x)\} = \texttt{Collect P}$$

$\texttt{a} \in (\texttt{Collect P}) \;\Leftrightarrow\; \texttt{P a}$ — mem_Collect_eq

$\texttt{Collect}\,(\lambda\,\texttt{x}.\,\texttt{x} \in \texttt{A}) = \texttt{A}$ — Collect_mem_eq

Set comprehension

- Elements of set S satisfying property P (maths: $\{x \in S \mid P(x)\}$):

$$t \in \{\, x \in S.\, P(x)\} \;\Leftrightarrow\; t \in S \wedge P(t)$$

- Term comprehension: set constructed from particular terms:

$$t \in \{\, f(x) \mid x.\, P(x)\,\} \;\Leftrightarrow\; (\,\exists\, x.\, P(x) \wedge t = f(x)\,)$$

- Set comprehension is the axiomatic constructor for sets:

$\texttt{Collect} :: (\texttt{'a} \Rightarrow \texttt{bool}) \Rightarrow \texttt{'a set}$

$\{x.\, P(x)\} = \texttt{Collect P}$

$\texttt{a} \in (\texttt{Collect P}) \;\Leftrightarrow\; \texttt{P a}$ mem_Collect_eq

$\texttt{Collect}\,(\lambda\, \texttt{x}.\, \texttt{x} \in \texttt{A}) = \texttt{A}$ Collect_mem_eq

Set comprehension

- Elements of set S satisfying property P (maths: $\{x \in S \mid P(x)\}$):

$$t \in \{\, x \in S.\, P(x)\} \;\Leftrightarrow\; t \in S \wedge P(t)$$

- Term comprehension: set constructed from particular terms:

$$t \in \{\, f(x) \mid x.\, P(x)\,\} \;\Leftrightarrow\; (\,\exists\, x.\, P(x) \wedge t = f(x)\,)$$

- Set comprehension is the axiomatic constructor for sets:

$$\texttt{Collect} :: (\texttt{'a} \Rightarrow \texttt{bool}) \Rightarrow \texttt{'a set}$$

$$\{x.\, P(x)\} = \texttt{Collect P}$$

$$\texttt{a} \in (\texttt{Collect P}) \;\Leftrightarrow\; \texttt{P a} \qquad \texttt{mem\_Collect\_eq}$$

$$\texttt{Collect}\,(\lambda\,\texttt{x}.\, \texttt{x} \in \texttt{A}) = \texttt{A} \qquad \texttt{Collect\_mem\_eq}$$

Set comprehension

- Elements of set S satisfying property P (maths: $\{x \in S \mid P(x)\}$):

$$t \in \{\, x \in S.\, P(x)\} \;\Leftrightarrow\; t \in S \wedge P(t)$$

- Term comprehension: set constructed from particular terms:

$$t \in \{\, f(x) \mid x.\, P(x)\,\} \;\Leftrightarrow\; (\,\exists\, x.\, P(x) \wedge t = f(x)\,)$$

- Set comprehension is the axiomatic constructor for sets:

$$\texttt{Collect} :: (\texttt{'a} \Rightarrow \texttt{bool}) \Rightarrow \texttt{'a set}$$

$$\{x.\, P(x)\} = \texttt{Collect P}$$

$$\texttt{a} \in (\texttt{Collect P}) \;\Leftrightarrow\; \texttt{P a} \qquad \texttt{mem\_Collect\_eq}$$

$$\texttt{Collect}\,(\lambda\, \texttt{x}.\, \texttt{x} \in \texttt{A}) = \texttt{A} \qquad \texttt{Collect\_mem\_eq}$$

Example: Subset Proofs

```
lemma subset_ex: "{1::nat,3,7,9} ⊆ {x. 0 < x ∧ x < 100}"
proof (rule subsetI)
  fix x :: nat
  assume "x ∈ {1, 3, 7, 9}"
  hence xs: "x = 1 ∨ x = 3 ∨ x = 7 ∨ x = 9"
    by (simp add: insert_iff empty_iff)
  show "x ∈ {x. 0 < x ∧ x < 100}"
  proof (unfold mem_Collect_eq, rule conjI)
    from xs show "0 < x" by auto
    from xs show "x < 100" by auto
  qed
qed
```

Example: Subset Proofs

```
lemma subset_ex: "{1::nat,3,7,9} ⊆ {x. 0 < x ∧ x < 100}"
proof (rule subsetI)
  fix x :: nat
  assume "x ∈ {1, 3, 7, 9}"
  hence xs: "x = 1 ∨ x = 3 ∨ x = 7 ∨ x = 9"
    by (simp add: insert_iff empty_iff)
  show "x ∈ {x. 0 < x ∧ x < 100}"
  proof (unfold mem_Collect_eq, rule conjI)
    from xs show "0 < x" by auto
    from xs show "x < 100" by auto
  qed
qed
```

Example: Subset Proofs

```
lemma subset_ex: "{1::nat,3,7,9} ⊆ {x. 0 < x ∧ x < 100}"
proof (rule subsetI)
  fix x :: nat
  assume "x ∈ {1, 3, 7, 9}"
  hence xs: "x = 1 ∨ x = 3 ∨ x = 7 ∨ x = 9"
    by (simp add: insert_iff empty_iff)
  show "x ∈ {x. 0 < x ∧ x < 100}"
  proof (unfold mem_Collect_eq, rule conjI)
    from xs show "0 < x" by auto
    from xs show "x < 100" by auto
  qed
qed
```

Example: Subset Proofs

```
lemma subset_ex: "{1::nat,3,7,9} ⊆ {x. 0 < x ∧ x < 100}"
proof (rule subsetI)
  fix x :: nat
  assume "x ∈ {1, 3, 7, 9}"
  hence xs: "x = 1 ∨ x = 3 ∨ x = 7 ∨ x = 9"
    by (simp add: insert_iff empty_iff)
  show "x ∈ {x. 0 < x ∧ x < 100}"
  proof (unfold mem_Collect_eq, rule conjI)
    from xs show "0 < x" by auto
    from xs show "x < 100" by auto
  qed
qed
```

Example: Subset Proofs

```
lemma subset_ex: "{1::nat,3,7,9} ⊆ {x. 0 < x ∧ x < 100}"
proof (rule subsetI)
  fix x :: nat
  assume "x ∈ {1, 3, 7, 9}"
  hence xs: "x = 1 ∨ x = 3 ∨ x = 7 ∨ x = 9"
    by (simp add: insert_iff empty_iff)
  show "x ∈ {x. 0 < x ∧ x < 100}"
  proof (unfold mem_Collect_eq, rule conjI)
    from xs show "0 < x" by auto
    from xs show "x < 100" by auto
  qed
qed
```

Example: Subset Proofs

```
lemma subset_ex: "{1::nat,3,7,9} ⊆ {x. 0 < x ∧ x < 100}"
proof (rule subsetI)
  fix x :: nat
  assume "x ∈ {1, 3, 7, 9}"
  hence xs: "x = 1 ∨ x = 3 ∨ x = 7 ∨ x = 9"
    by (simp add: insert_iff empty_iff)
  show "x ∈ {x. 0 < x ∧ x < 100}"
  proof (unfold mem_Collect_eq, rule conjI)
    from xs show "0 < x" by auto
    from xs show "x < 100" by auto
  qed
qed
```

Example: Subset Proofs

```
lemma subset_ex: "{1::nat,3,7,9} ⊆ {x. 0 < x ∧ x < 100}"
proof (rule subsetI)
  fix x :: nat
  assume "x ∈ {1, 3, 7, 9}"
  hence xs: "x = 1 ∨ x = 3 ∨ x = 7 ∨ x = 9"
    by (simp add: insert_iff empty_iff)
  show "x ∈ {x. 0 < x ∧ x < 100}"
  proof (unfold mem_Collect_eq, rule conjI)
    from xs show "0 < x" by auto
    from xs show "x < 100" by auto
  qed
qed
```

Example: Subset Proofs

```
lemma subset_ex: "{1::nat,3,7,9} ⊆ {x. 0 < x ∧ x < 100}"
proof (rule subsetI)
  fix x :: nat
  assume "x ∈ {1, 3, 7, 9}"
  hence xs: "x = 1 ∨ x = 3 ∨ x = 7 ∨ x = 9"
    by (simp add: insert_iff empty_iff)
  show "x ∈ {x. 0 < x ∧ x < 100}"
  proof (unfold mem_Collect_eq, rule conjI)
    from xs show "0 < x" by auto
    from xs show "x < 100" by auto
  qed
qed
```

Example: Subset Proofs

```
lemma subset_ex: "{1::nat,3,7,9} ⊆ {x. 0 < x ∧ x < 100}"
proof (rule subsetI)
  fix x :: nat
  assume "x ∈ {1, 3, 7, 9}"
  hence xs: "x = 1 ∨ x = 3 ∨ x = 7 ∨ x = 9"
    by (simp add: insert_iff empty_iff)
  show "x ∈ {x. 0 < x ∧ x < 100}"
  proof (unfold mem_Collect_eq, rule conjI)
    from xs show "0 < x" by auto
    from xs show "x < 100" by auto
  qed
qed
```

Example: Subset Proofs

```
lemma subset_ex: "{1::nat,3,7,9} ⊆ {x. 0 < x ∧ x < 100}"
proof (rule subsetI)
  fix x :: nat
  assume "x ∈ {1, 3, 7, 9}"
  hence xs: "x = 1 ∨ x = 3 ∨ x = 7 ∨ x = 9"
    by (simp add: insert_iff empty_iff)
  show "x ∈ {x. 0 < x ∧ x < 100}"
  proof (unfold mem_Collect_eq, rule conjI)
    from xs show "0 < x" by auto
    from xs show "x < 100" by auto
  qed
qed
```

Example: Subset Proofs

```
lemma subset_ex: "{1::nat,3,7,9} ⊆ {x. 0 < x ∧ x < 100}"
proof (rule subsetI)
  fix x :: nat
  assume "x ∈ {1, 3, 7, 9}"
  hence xs: "x = 1 ∨ x = 3 ∨ x = 7 ∨ x = 9"
    by (simp add: insert_iff empty_iff)
  show "x ∈ {x. 0 < x ∧ x < 100}"
  proof (unfold mem_Collect_eq, rule conjI)
    from xs show "0 < x" by auto
    from xs show "x < 100" by auto
  qed
qed
```

Example: Subset Proofs

```
lemma subset_ex: "{1::nat,3,7,9} ⊆ {x. 0 < x ∧ x < 100}"
proof (rule subsetI)
  fix x :: nat
  assume "x ∈ {1, 3, 7, 9}"
  hence xs: "x = 1 ∨ x = 3 ∨ x = 7 ∨ x = 9"
    by (simp add: insert_iff empty_iff)
  show "x ∈ {x. 0 < x ∧ x < 100}"
  proof (unfold mem_Collect_eq, rule conjI)
    from xs show "0 < x" by auto
    from xs show "x < 100" by auto
  qed
qed
```

Example: Subset Proofs

```
lemma subset_ex: "{1::nat,3,7,9} ⊆ {x. 0 < x ∧ x < 100}"
proof (rule subsetI)
  fix x :: nat
  assume "x ∈ {1, 3, 7, 9}"
  hence xs: "x = 1 ∨ x = 3 ∨ x = 7 ∨ x = 9"
    by (simp add: insert_iff empty_iff)
  show "x ∈ {x. 0 < x ∧ x < 100}"
  proof (unfold mem_Collect_eq, rule conjI)
    from xs show "0 < x" by auto
    from xs show "x < 100" by auto
  qed
qed
```

Outline

Set Operators

- Set union:

$$x \in (A \cup B) \longleftrightarrow x \in A \lor x \in B \qquad \text{Un\_iff}$$

- Set intersection:

$$x \in (A \cap B) \longleftrightarrow x \in A \land x \in B \qquad \text{Int\_iff}$$

- Set difference (maths: $A \setminus B$):

$$x \in (A - B) \longleftrightarrow x \in A \land x \notin B \qquad \text{Diff\_iff}$$

Example

$\{Atlantic, Indian\} \cup \{Indian, Pacific\} = \{Atlantic, Indian, Pacific\}$
$\{Atlantic, Indian\} \cap \{Indian, Pacific\} = \{Indian\}$
$\{Atlantic, Indian\} - \{Indian, Pacific\} = \{Atlantic\}$

Set Operators

- Set union:

$$x \in (A \cup B) \Leftrightarrow x \in A \vee x \in B$$ `Un_iff`

- Set intersection:

$$x \in (A \cap B) \Leftrightarrow x \in A \wedge x \in B$$ `Int_iff`

- Set difference (maths: $A \setminus B$):

$$x \in (A - B) \Leftrightarrow x \in A \wedge x \notin B$$ `Diff_iff`

Example

$\{Atlantic, Indian\} \cup \{Indian, Pacific\} = \{Atlantic, Indian, Pacific\}$

$\{Atlantic, Indian\} \cap \{Indian, Pacific\} = \{Indian\}$

$\{Atlantic, Indian\} - \{Indian, Pacific\} = \{Atlantic\}$

Set Operators

- Set union:

$$x \in (A \cup B) \iff x \in A \lor x \in B \qquad \texttt{Un\_iff}$$

- Set intersection:

$$x \in (A \cap B) \iff x \in A \land x \in B$$

- Set difference (maths: $A \setminus B$):

$$x \in (A - B) \iff x \in A \land x \notin B$$

Example

$\{Atlantic, Indian\} \cup \{Indian, Pacific\} = \{Atlantic, Indian, Pacific\}$

$\{Atlantic, Indian\} \cap \{Indian, Pacific\} = \{Indian\}$

$\{Atlantic, Indian\} - \{Indian, Pacific\} = \{Atlantic\}$

Set Operators

- Set union:

$$x \in (A \cup B) \;\Leftrightarrow\; x \in A \vee x \in B \qquad \texttt{Un\_iff}$$

- Set intersection:

$$x \in (A \cap B) \;\Leftrightarrow\; x \in A \wedge x \in B \qquad \texttt{Int\_iff}$$

- Set difference (maths: $A \setminus B$):

$$x \in (A - B) \;\Leftrightarrow\; x \in A \wedge x \notin B \qquad \texttt{Diff\_iff}$$

Example

$$\{Atlantic, Indian\} \cup \{Indian, Pacific\} = \{Atlantic, Indian, Pacific\}$$

$$\{Atlantic, Indian\} \cap \{Indian, Pacific\} = \{Indian\}$$

$$\{Atlantic, Indian\} - \{Indian, Pacific\} = \{Atlantic\}$$

Set Operators

- Set union:

$$x \in (A \cup B) \iff x \in A \lor x \in B \qquad \texttt{Un\_iff}$$

- Set intersection:

$$x \in (A \cap B) \iff x \in A \land x \in B \qquad \texttt{Int\_iff}$$

- Set difference (maths: $A \setminus B$):

$$x \in (A - B) \iff x \in A \land x \notin B$$

Example

$$\{Atlantic, Indian\} \cup \{Indian, Pacific\} = \{Atlantic, Indian, Pacific\}$$
$$\{Atlantic, Indian\} \cap \{Indian, Pacific\} = \{Indian\}$$
$$\{Atlantic, Indian\} - \{Indian, Pacific\} = \{Atlantic\}$$

Set Operators

- Set union:

$$x \in (A \cup B) \;\Leftrightarrow\; x \in A \vee x \in B \qquad \texttt{Un\_iff}$$

- Set intersection:

$$x \in (A \cap B) \;\Leftrightarrow\; x \in A \wedge x \in B \qquad \texttt{Int\_iff}$$

- Set difference (maths: $A \setminus B$):

$$x \in (A - B) \;\Leftrightarrow\; x \in A \wedge x \notin B \qquad \texttt{Diff\_iff}$$

Example

$$\{Atlantic, Indian\} \cup \{Indian, Pacific\} = \{Atlantic, Indian, Pacific\}$$

$$\{Atlantic, Indian\} \cap \{Indian, Pacific\} = \{Indian\}$$

$$\{Atlantic, Indian\} - \{Indian, Pacific\} = \{Atlantic\}$$

Set Operators

- Set union:

$$x \in (A \cup B) \ \Leftrightarrow \ x \in A \vee x \in B \qquad \texttt{Un\_iff}$$

- Set intersection:

$$x \in (A \cap B) \ \Leftrightarrow \ x \in A \wedge x \in B \qquad \texttt{Int\_iff}$$

- Set difference (maths: $A \setminus B$):

$$x \in (A - B) \ \Leftrightarrow \ x \in A \wedge x \notin B \qquad \texttt{Diff\_iff}$$

Example

$$\{Atlantic, Indian\} \cup \{Indian, Pacific\} = \{Atlantic, Indian, Pacific\}$$

$$\{Atlantic, Indian\} \cap \{Indian, Pacific\} = \{Indian\}$$

$$\{Atlantic, Indian\} - \{Indian, Pacific\} = \{Atlantic\}$$

Set Operators

- Set union:

$$x \in (A \cup B) \;\Leftrightarrow\; x \in A \vee x \in B \qquad \texttt{Un\_iff}$$

- Set intersection:

$$x \in (A \cap B) \;\Leftrightarrow\; x \in A \wedge x \in B \qquad \texttt{Int\_iff}$$

- Set difference (maths: $A \setminus B$):

$$x \in (A - B) \;\Leftrightarrow\; x \in A \wedge x \notin B \qquad \texttt{Diff\_iff}$$

Example

- $\{Atlantic, Indian\} \cup \{Indian, Pacific\} = \{Atlantic, Indian, Pacific\}$
- $\{Atlantic, Indian\} \cap \{Indian, Pacific\} = \{Indian\}$
- $\{Atlantic, Indian\} - \{Indian, Pacific\} = \{Atlantic\}$

Set Operators

- Set union:

$$x \in (A \cup B) \;\Leftrightarrow\; x \in A \lor x \in B \qquad \texttt{Un\_iff}$$

- Set intersection:

$$x \in (A \cap B) \;\Leftrightarrow\; x \in A \land x \in B \qquad \texttt{Int\_iff}$$

- Set difference (maths: $A \setminus B$):

$$x \in (A - B) \;\Leftrightarrow\; x \in A \land x \notin B \qquad \texttt{Diff\_iff}$$

Example

- $\{Atlantic, Indian\} \cup \{Indian, Pacific\} = \{Atlantic, Indian, Pacific\}$
- $\{Atlantic, Indian\} \cap \{Indian, Pacific\} = \{Indian\}$
- $\{Atlantic, Indian\} - \{Indian, Pacific\} = \{Atlantic\}$

Set Operators

- Set union:

$$x \in (A \cup B) \;\Leftrightarrow\; x \in A \vee x \in B \qquad \texttt{Un\_iff}$$

- Set intersection:

$$x \in (A \cap B) \;\Leftrightarrow\; x \in A \wedge x \in B \qquad \texttt{Int\_iff}$$

- Set difference (maths: $A \setminus B$):

$$x \in (A - B) \;\Leftrightarrow\; x \in A \wedge x \notin B \qquad \texttt{Diff\_iff}$$

Example

- $\{Atlantic, Indian\} \cup \{Indian, Pacific\} = \{Atlantic, Indian, Pacific\}$
- $\{Atlantic, Indian\} \cap \{Indian, Pacific\} = \{Indian\}$
- $\{Atlantic, Indian\} - \{Indian, Pacific\} = \{Atlantic\}$

Set Operators

- Set union:

$$x \in (A \cup B) \;\Leftrightarrow\; x \in A \lor x \in B \qquad \texttt{Un\_iff}$$

- Set intersection:

$$x \in (A \cap B) \;\Leftrightarrow\; x \in A \land x \in B \qquad \texttt{Int\_iff}$$

- Set difference (maths: $A \setminus B$):

$$x \in (A - B) \;\Leftrightarrow\; x \in A \land x \notin B \qquad \texttt{Diff\_iff}$$

Example

- $\{Atlantic, Indian\} \cup \{Indian, Pacific\} = \{Atlantic, Indian, Pacific\}$
- $\{Atlantic, Indian\} \cap \{Indian, Pacific\} = \{Indian\}$
- $\{Atlantic, Indian\} - \{Indian, Pacific\} = \{Atlantic\}$

Distributed Operators

- Distributed union:

  $$\bigcup\{A, B, C, \cdots\} = A \cup B \cup C \cup \cdots$$

  $$x \in \bigcup S \Leftrightarrow (\exists A \in S \bullet x \in A)$$ `Union_iff`

- Distributed intersection:

  $$\bigcap\{A, B, C, \cdots\} = A \cap B \cap C \cap \cdots$$

  $$x \in \bigcap S \Leftrightarrow (\forall A \in S \bullet x \in A)$$ `Inter_iff`

Distributed Operators

- Distributed union:

$$\bigcup\{A, B, C, \cdots\} = A \cup B \cup C \cup \cdots$$

$$x \in \bigcup S \Leftrightarrow (\exists A \in S \bullet x \in A) \qquad \texttt{Union\_iff}$$

- Distributed intersection:

$$\bigcap\{A, B, C, \cdots\} = A \cap B \cap C \cap \cdots$$

$$x \in \bigcap S \Leftrightarrow (\forall A \in S \bullet x \in A) \qquad \texttt{Inter\_iff}$$

Distributed Operators

- Distributed union:

$$\bigcup\{A, B, C, \cdots\} = A \cup B \cup C \cup \cdots$$

$$x \in \bigcup S \Leftrightarrow (\exists A \in S \bullet x \in A) \qquad \texttt{Union\_iff}$$

- Distributed intersection:

$$\bigcap\{A, B, C, \cdots\} = A \cap B \cap C \cap \cdots$$

$$x \in \bigcap S \Leftrightarrow (\forall A \in S \bullet x \in A) \qquad \texttt{Inter\_iff}$$

Distributed Operators

- Distributed union:

$$\bigcup\{A, B, C, \cdots\} \;=\; A \cup B \cup C \cup \cdots$$

$$x \in \bigcup S \;\Leftrightarrow\; (\,\exists\, A \in S \bullet x \in A\,) \qquad \texttt{Union\_iff}$$

- Distributed intersection:

Distributed Operators

- Distributed union:

$$\bigcup\{A, B, C, \cdots\} \; = \; A \cup B \cup C \cup \cdots$$

$$x \in \bigcup S \;\Leftrightarrow\; (\exists A \in S \bullet x \in A) \qquad \texttt{Union\_iff}$$

- Distributed intersection:

$$\bigcap\{A, B, C, \cdots\} \; = \; A \cap B \cap C \cap \cdots$$

$$x \in \bigcap S \;\Leftrightarrow\; (\forall A \in S \bullet x \in A) \qquad \texttt{Inter\_iff}$$

Distributed Operators

- Distributed union:

$$\bigcup\{A, B, C, \cdots\} \; = \; A \cup B \cup C \cup \cdots$$

$$x \in \bigcup S \;\Leftrightarrow\; (\exists A \in S \bullet x \in A) \qquad \texttt{Union\_iff}$$

- Distributed intersection:

$$\bigcap\{A, B, C, \cdots\} \; = \; A \cap B \cap C \cap \cdots$$

$$x \in \bigcap S \;\Leftrightarrow\; (\forall A \in S \bullet x \in A) \qquad \texttt{Inter\_iff}$$

Distributed Operators

- Distributed union:

$$\bigcup\{A, B, C, \cdots\} \;=\; A \cup B \cup C \cup \cdots$$

$$x \in \bigcup S \;\Leftrightarrow\; (\exists A \in S \bullet x \in A) \qquad \texttt{Union\_iff}$$

- Distributed intersection:

$$\bigcap\{A, B, C, \cdots\} \;=\; A \cap B \cap C \cap \cdots$$

$$x \in \bigcap S \;\Leftrightarrow\; (\forall A \in S \bullet x \in A) \qquad \texttt{Inter\_iff}$$

Intervals

- Interval between two endpoints: `{m..n}` (maths: $[m, n]$).
- Lower bound: `{m..}` ($[m, +\infty)$).
- Upper bound: `{..n}` ($(-\infty, n]$).
- Strictly between two endpoints: `{m<..<n}` ((m, n)).
- Strict lower bound: `{m<..}` ($(m, +\infty)$).
- Strict upper bound: `{..<n}` ($(-\infty, n)$).

Intervals

- Interval between two endpoints: `{m..n}` (maths: $[m, n]$).
- Lower bound: `{m..}` ($[m, +\infty)$).
- Upper bound: `{..n}` ($(-\infty, n]$).
- Strictly between two endpoints: `{m<..<n}` ((m, n)).
- Strict lower bound: `{m<..}` ($(m, +\infty)$).
- Strict upper bound: `{..<n}` ($(-\infty, n)$).

Intervals

- Interval between two endpoints: `{m..n}` (maths: $[m, n]$).
- Lower bound: `{m..}` ($[m, +\infty)$).
- Upper bound: `{..n}` ($(-\infty, n]$).
- Strictly between two endpoints: `{m<..<n}` ((m, n)).
- Strict lower bound: `{m<..}` ($(m, +\infty)$).
- Strict upper bound: `{..<n}` ($(-\infty, n)$).

Intervals

- Interval between two endpoints: `{m..n}` (maths: $[m, n]$).
- Lower bound: `{m..}` ($[m, +\infty)$).
- Upper bound: `{..n}` ($(-\infty, n]$).
- Strictly between two endpoints: `{m<..<n}` ((m, n)).
- Strict lower bound: `{m<..}` ($(m, +\infty)$).
- Strict upper bound: `{..<n}` ($(-\infty, n)$).

Intervals

- Interval between two endpoints: `{m..n}` (maths: $[m, n]$).
- Lower bound: `{m..}` ($[m, +\infty)$).
- Upper bound: `{..n}` ($(-\infty, n]$).
- Strictly between two endpoints: `{m<..<n}` ((m, n)).
- Strict lower bound: `{m<..}` ($(m, +\infty)$).
- Strict upper bound: `{..<n}` ($(-\infty, n)$).

Intervals

- Interval between two endpoints: `{m..n}` (maths: $[m, n]$).
- Lower bound: `{m..}` ($[m, +\infty)$).
- Upper bound: `{..n}` ($(-\infty, n]$).
- Strictly between two endpoints: `{m<..<n}` ((m, n)).
- Strict lower bound: `{m<..}` ($(m, +\infty)$).
- Strict upper bound: `{..<n}` ($(-\infty, n)$).

Intervals

- Interval between two endpoints: `{m..n}` (maths: $[m, n]$).
- Lower bound: `{m..}` ($[m, +\infty)$).
- Upper bound: `{..n}` ($(-\infty, n]$).
- Strictly between two endpoints: `{m<..<n}` ((m, n)).
- Strict lower bound: `{m<..}` ($(m, +\infty)$).
- Strict upper bound: `{..<n}` ($(-\infty, n)$).

Outline

1. Set theory in Isabelle/HOL
2. Operators on sets
3. Finite sets
4. Uncomputable objects

Finite sets

- Finite set has a finite number of elements, e.g. $\{u_1, u_2, \cdots, u_n\}$.
- Non-finite sets are infinite. For example, `{0::nat..}`.
- Finite characterised by `finite :: 'a set ⇒ bool` in HOL.

```
finite {}                                            finite.emptyI
finite (insert x A) ⟷ finite A                      finite_insert
finite (A ∪ B) ⟷ (finite A ∧ finite B)              finite_Un
```

- Lists can be converted to a finite set with `set :: 'a list ⇒ 'a set`.
- The resulting set ignores the occurrence and order of list elements :

```
finite (set xs)                                      finite_set
set [] = {}                                          empty_set
set (x # xs) = insert x (set xs)
```

Finite sets

- Finite set has a finite number of elements, e.g. $\{u_1, u_2, \cdots, u_n\}$.
- Non-finite sets are infinite. For example, `{0::nat..}`.
- Finite characterised by `finite :: 'a set ⇒ bool` in HOL.

```
finite {}                                      finite_empty
finite (insert x A) ⟷ finite A                finite_insert
finite (A ∪ B) ⟷ (finite A ∧ finite B)        finite_Un
```

- Lists can be converted to a finite set with `set :: 'a list ⇒ 'a set`.
- The resulting set ignores the occurrence and order of list elements :

```
finite (set xs)                                finite_set
set [] = {}                                    empty_set
set (x # xs) = insert x (set xs)               list_set
```

Finite sets

- Finite set has a finite number of elements, e.g. $\{u_1, u_2, \cdots, u_n\}$.
- Non-finite sets are infinite. For example, `{0::nat..}`.
- Finite characterised by `finite :: 'a set ⇒ bool` in HOL.

```
finite {}
finite (insert x A) = finite A
finite (A ∪ B) ⟷ (finite A ∧ finite B)
```

- Lists can be converted to a finite set with `set :: 'a list ⇒ 'a set`.
- The resulting set ignores the occurrence and order of list elements :

```
finite (set xs)
set [] = {}
set (x # xs) = insert x (set xs)
```

Finite sets

- Finite set has a finite number of elements, e.g. $\{u_1, u_2, \cdots, u_n\}$.
- Non-finite sets are infinite. For example, `{0::nat..}`.
- Finite characterised by `finite :: 'a set ⇒ bool` in HOL.

```
finite {}                                   finite.emptyI
finite (insert x A) ⇔ finite A              finite_insert
finite (A ∪ B) ⇔ (finite A ∧ finite B)         finite_Un
```

- Lists can be converted to a finite set with `set :: 'a list ⇒ 'a set`.
- The resulting set ignores the occurrence and order of list elements :

```
finite (set xs)
set [] = {}
set (x # xs) = insert x (set xs)
```

Finite sets

- Finite set has a finite number of elements, e.g. $\{u_1, u_2, \cdots, u_n\}$.
- Non-finite sets are infinite. For example, `{0::nat..}`.
- Finite characterised by `finite :: 'a set ⇒ bool` in HOL.

```
finite {}                                finite.emptyI
finite (insert x A) ⇔ finite A           finite_insert
finite (A ∪ B) ⇔ (finite A ∧ finite B)   finite_Un
```

- Lists can be converted to a finite set with `set :: 'a list ⇒ 'a set`.
- The resulting set ignores the occurrence and order of list elements :

```
finite (set xs)
set [] = {}
set (x # xs) = insert x (set xs)
```

Finite sets

- Finite set has a finite number of elements, e.g. $\{u_1, u_2, \cdots, u_n\}$.
- Non-finite sets are infinite. For example, `{0::nat..}`.
- Finite characterised by `finite :: 'a set ⇒ bool` in HOL.

`finite {}`	`finite.emptyI`
`finite (insert x A) ⇔ finite A`	`finite_insert`
`finite (A ∪ B) ⇔ (finite A ∧ finite B)`	`finite_Un`

- Lists can be converted to a finite set with `set :: 'a list ⇒ 'a set`.
- The resulting set ignores the occurrence and order of list elements :

`finite (set xs)`	`finite_set`
`set [] = {}`	`empty_set`
`set (x # xs) = insert x (set xs)`	`list.set`

Finite sets

- Finite set has a finite number of elements, e.g. $\{u_1, u_2, \cdots, u_n\}$.
- Non-finite sets are infinite. For example, `{0::nat..}`.
- Finite characterised by `finite :: 'a set ⇒ bool` in HOL.

```
finite {}                                   finite.emptyI
finite (insert x A) ⇔ finite A              finite_insert
finite (A ∪ B) ⇔ (finite A ∧ finite B)      finite_Un
```

- Lists can be converted to a finite set with `set :: 'a list ⇒ 'a set`.
- The resulting set ignores the occurrence and order of list elements :

```
finite (set xs)                             finite_set
set [] = {}                                 empty_set
set (x # xs) = insert x (set xs)            list.set
```

Finite sets

- Finite set has a finite number of elements, e.g. $\{u_1, u_2, \cdots, u_n\}$.
- Non-finite sets are infinite. For example, `{0::nat..}`.
- Finite characterised by `finite :: 'a set ⇒ bool` in HOL.

`finite {}`	`finite.emptyI`
`finite (insert x A) ⇔ finite A`	`finite_insert`
`finite (A ∪ B) ⇔ (finite A ∧ finite B)`	`finite_Un`

- Lists can be converted to a finite set with `set :: 'a list ⇒ 'a set`.
- The resulting set ignores the occurrence and order of list elements :

`finite (set xs)`	`finite_set`
`set [] = {}`	`empty_set`
`set (x # xs) = insert x (set xs)`	`list.set`

Finite sets

- Finite set has a finite number of elements, e.g. $\{u_1, u_2, \cdots, u_n\}$.
- Non-finite sets are infinite. For example, `{0::nat..}`.
- Finite characterised by `finite :: 'a set ⇒ bool` in HOL.

`finite {}`	`finite.emptyI`
`finite (insert x A) ⇔ finite A`	`finite_insert`
`finite (A ∪ B) ⇔ (finite A ∧ finite B)`	`finite_Un`

- Lists can be converted to a finite set with `set :: 'a list ⇒ 'a set`.
- The resulting set ignores the occurrence and order of list elements :

`finite (set xs)`	`finite_set`
`set [] = {}`	`empty_set`
`set (x # xs) = insert x (set xs)`	`list.set`

Finite sets

- Finite set has a finite number of elements, e.g. $\{u_1, u_2, \cdots, u_n\}$.
- Non-finite sets are infinite. For example, `{0::nat..}`.
- Finite characterised by `finite :: 'a set ⇒ bool` in HOL.

`finite {}`	`finite.emptyI`
`finite (insert x A) ⇔ finite A`	`finite_insert`
`finite (A ∪ B) ⇔ (finite A ∧ finite B)`	`finite_Un`

- Lists can be converted to a finite set with `set :: 'a list ⇒ 'a set`.
- The resulting set ignores the occurrence and order of list elements :

`finite (set xs)`	`finite_set`
`set [] = {}`	`empty_set`
`set (x # xs) = insert x (set xs)`	`list.set`

Finite sets

- Finite set has a finite number of elements, e.g. $\{u_1, u_2, \cdots, u_n\}$.
- Non-finite sets are infinite. For example, `{0::nat..}`.
- Finite characterised by `finite :: 'a set ⇒ bool` in HOL.

```
finite {}                                     finite.emptyI
finite (insert x A) ⇔ finite A                finite_insert
finite (A ∪ B) ⇔ (finite A ∧ finite B)        finite_Un
```

- Lists can be converted to a finite set with `set :: 'a list ⇒ 'a set`.
- The resulting set ignores the occurrence and order of list elements :

```
finite (set xs)                               finite_set
set [] = {}                                   empty_set
set (x # xs) = insert x (set xs)              list.set
```

Finite sets

- Finite set has a finite number of elements, e.g. $\{u_1, u_2, \cdots, u_n\}$.
- Non-finite sets are infinite. For example, `{0::nat..}`.
- Finite characterised by `finite :: 'a set ⇒ bool` in HOL.

```
finite {}                                   finite.emptyI
finite (insert x A) ⇔ finite A              finite_insert
finite (A ∪ B) ⇔ (finite A ∧ finite B)      finite_Un
```

- Lists can be converted to a finite set with `set :: 'a list ⇒ 'a set`.
- The resulting set ignores the occurrence and order of list elements :

```
finite (set xs)                             finite_set
set [] = {}                                 empty_set
set (x # xs) = insert x (set xs)            list.set
```

Power set

- Set of all subsets of A written as $\mathbb{P}\,A$:

$$S \in \mathbb{P}\,A \iff S \subseteq A \qquad \texttt{Pow\_iff}$$

- $\mathbb{P}\{1,2,3\} = \{\{\},\{1\},\{2\},\{3\},\{1,2\},\{1,3\},\{2,3\},\{1,2,3\}\}$.
- `Pow` :: `'a set` $\Rightarrow$ `'a set set` in Isabelle/HOL.
- Set of all finite powersets: $\mathbb{F}\,A$.
- `Fpow` :: `'a set` $\Rightarrow$ `'a set set` in Isabelle/HOL

$$\mathbb{F}\,A = \{X.\ X \subseteq A \wedge \mathit{finite}\ X\} \qquad \texttt{Fpow\_def}$$

$$\mathbb{F}\,A \subseteq \mathbb{P}\,A \qquad \texttt{Fpow\_subset\_Pow}$$

Power set

- Set of all subsets of A written as $\mathbb{P}\,A$:

$$S \in \mathbb{P}\,A \;\Leftrightarrow\; S \subseteq A \qquad \texttt{Pow\_iff}$$

- $\mathbb{P}\{1,2,3\} = \{\{\},\{1\},\{2\},\{3\},\{1,2\},\{1,3\},\{2,3\},\{1,2,3\}\}$.
- `Pow` :: `'a set` $\Rightarrow$ `'a set set` in Isabelle/HOL.
- Set of all finite powersets: $\mathbb{F}\,A$.
- `Fpow` :: `'a set` $\Rightarrow$ `'a set set` in Isabelle/HOL

$$\mathbb{F}\,A = \{X.\; X \subseteq A \wedge \mathit{finite}\; X\} \qquad \texttt{Fpow\_def}$$

$$\mathbb{F}\,A \subseteq \mathbb{P}\,A \qquad \texttt{Fpow\_subset\_Pow}$$

Power set

- Set of all subsets of A written as $\mathbb{P}\,A$:

$$S \in \mathbb{P}\,A \;\Leftrightarrow\; S \subseteq A \qquad \texttt{Pow\_iff}$$

- $\mathbb{P}\{1,2,3\} = \{\{\},\{1\},\{2\},\{3\},\{1,2\},\{1,3\},\{2,3\},\{1,2,3\}\}$.
- `Pow` :: `'a set` $\Rightarrow$ `'a set set` in Isabelle/HOL.
- Set of all finite powersets: $\mathbb{F}\,A$.
- `Fpow` :: `'a set` $\Rightarrow$ `'a set set` in Isabelle/HOL

$$\mathbb{F}\,A = \{X.\; X \subseteq A \land \mathit{finite}\; X\} \qquad \texttt{Fpow\_def}$$

$$\mathbb{F}\,A \subseteq \mathbb{P}\,A \qquad \texttt{Fpow\_subset\_Pow}$$

Power set

- Set of all subsets of A written as $\mathbb{P}\,A$:

$$S \in \mathbb{P}\,A \;\Leftrightarrow\; S \subseteq A \qquad \texttt{Pow\_iff}$$

- $\mathbb{P}\{1,2,3\} = \{\{\},\{1\},\{2\},\{3\},\{1,2\},\{1,3\},\{2,3\},\{1,2,3\}\}$.
- `Pow` :: `'a set` $\Rightarrow$ `'a set set` in Isabelle/HOL.
- Set of all finite powersets: $\mathbb{F}\,A$.
- `Fpow` :: `'a set` $\Rightarrow$ `'a set set` in Isabelle/HOL

$$\mathbb{F}\,A = \{X.\ X \subseteq A \wedge \mathit{finite}\ X\} \qquad \texttt{Fpow\_def}$$

$$\mathbb{F}\,A \subseteq \mathbb{P}\,A \qquad \texttt{Fpow\_subset\_Pow}$$

Power set

- Set of all subsets of A written as $\mathbb{P}\,A$:

$$S \in \mathbb{P}\,A \;\Leftrightarrow\; S \subseteq A \qquad \texttt{Pow\_iff}$$

- $\mathbb{P}\{1,2,3\} = \{\{\},\{1\},\{2\},\{3\},\{1,2\},\{1,3\},\{2,3\},\{1,2,3\}\}$.
- `Pow` :: `'a set` $\Rightarrow$ `'a set set` in Isabelle/HOL.
- Set of all finite powersets: $\mathbb{F}\,A$.
- `Fpow` :: `'a set` $\Rightarrow$ `'a set set` in Isabelle/HOL

Power set

- Set of all subsets of A written as $\mathbb{P}\,A$:

$$S \in \mathbb{P}\,A \;\Leftrightarrow\; S \subseteq A \qquad \texttt{Pow\_iff}$$

- $\mathbb{P}\{1,2,3\} = \{\{\},\{1\},\{2\},\{3\},\{1,2\},\{1,3\},\{2,3\},\{1,2,3\}\}$.
- `Pow` :: `'a set` $\Rightarrow$ `'a set set` in Isabelle/HOL.
- Set of all finite powersets: $\mathbb{F}\,A$.
- `Fpow` :: `'a set` $\Rightarrow$ `'a set set` in Isabelle/HOL

$$\mathbb{F}\,A = \{X.\ X \subseteq A \wedge \mathit{finite}\ X\} \qquad \texttt{Fpow\_def}$$

$$\mathbb{F}\,A \subseteq \mathbb{P}\,A \qquad \texttt{Fpow\_subset\_Pow}$$

Power set

- Set of all subsets of A written as $\mathbb{P}\,A$:

$$S \in \mathbb{P}\,A \;\Leftrightarrow\; S \subseteq A \qquad \texttt{Pow\_iff}$$

- $\mathbb{P}\{1,2,3\} = \{\{\},\{1\},\{2\},\{3\},\{1,2\},\{1,3\},\{2,3\},\{1,2,3\}\}$.
- `Pow` :: `'a set` $\Rightarrow$ `'a set set` in Isabelle/HOL.
- Set of all finite powersets: $\mathbb{F}\,A$.
- `Fpow` :: `'a set` $\Rightarrow$ `'a set set` in Isabelle/HOL

$$\mathbb{F}\,A \;=\; \{X.\, X \subseteq A \wedge \mathit{finite}\; X\} \qquad \texttt{Fpow\_def}$$

$$\mathbb{F}\,A \;\subseteq\; \mathbb{P}\,A \qquad \texttt{Fpow\_subset\_Pow}$$

Power set

- Set of all subsets of A written as $\mathbb{P}\,A$:

$$S \in \mathbb{P}\,A \;\Leftrightarrow\; S \subseteq A \qquad \texttt{Pow\_iff}$$

- $\mathbb{P}\{1,2,3\} = \{\{\},\{1\},\{2\},\{3\},\{1,2\},\{1,3\},\{2,3\},\{1,2,3\}\}$.
- `Pow` :: `'a set` $\Rightarrow$ `'a set set` in Isabelle/HOL.
- Set of all finite powersets: $\mathbb{F}\,A$.
- `Fpow` :: `'a set` $\Rightarrow$ `'a set set` in Isabelle/HOL

$$\mathbb{F}\,A \;=\; \{X.\, X \subseteq A \wedge \mathit{finite}\ X\} \qquad \texttt{Fpow\_def}$$

$$\mathbb{F}\,A \;\subseteq\; \mathbb{P}\,A \qquad \texttt{Fpow\_subset\_Pow}$$

Power set

- Set of all subsets of A written as $\mathbb{P}\,A$:

$$S \in \mathbb{P}\,A \;\Leftrightarrow\; S \subseteq A \qquad \texttt{Pow\_iff}$$

- $\mathbb{P}\{1,2,3\} = \{\{\},\{1\},\{2\},\{3\},\{1,2\},\{1,3\},\{2,3\},\{1,2,3\}\}$.
- $\texttt{Pow} :: \texttt{'a set} \Rightarrow \texttt{'a set set}$ in Isabelle/HOL.
- Set of all finite powersets: $\mathbb{F}\,A$.
- $\texttt{Fpow} :: \texttt{'a set} \Rightarrow \texttt{'a set set}$ in Isabelle/HOL

$$\mathbb{F}\,A \;=\; \{X.\, X \subseteq A \wedge \mathit{finite}\; X\} \qquad \texttt{Fpow\_def}$$

$$\mathbb{F}\,A \;\subseteq\; \mathbb{P}\,A \qquad \texttt{Fpow\_subset\_Pow}$$

Cartesian product

- Suppose A and B are sets.
- The cartesian product $A \times B$ is the set of all tuples (x, y).
- x is an element of A and y is an element of B.

$$(x, y) \in A \times B \Leftrightarrow x \in A \wedge y \in B$$

- Membership:

$$(x_1, \ldots, x_n) \in A_1 \times \ldots \times A_n \Leftrightarrow x_1 \in A_1 \wedge \ldots \wedge x_n \in A_n$$

- Equality:

$$(x_1, \ldots, x_n) = (y_1, \ldots, y_n) \Leftrightarrow x_1 = y_1 \wedge \ldots \wedge x_n = y_n$$

Cartesian product

- Suppose A and B are sets.
- The cartesian product $A \times B$ is the set of all tuples (x, y).
- x is an element of A and y is an element of B.
- Membership:
- Equality:

Cartesian product

- Suppose A and B are sets.
- The cartesian product $A \times B$ is the set of all tuples (x, y).
- x is an element of A and y is an element of B.
- Membership:
- Equality:

Cartesian product

- Suppose A and B are sets.
- The cartesian product $A \times B$ is the set of all tuples (x, y).
- x is an element of A and y is an element of B.

$$(x, y) \in A \times B \;\Leftrightarrow\; x \in A \wedge y \in B \qquad \texttt{mem\_Sigma\_iff}$$

- Membership:

$$(x_1, \ldots, x_n) \in A_1 \times \ldots \times A_n \;\Leftrightarrow\; x_1 \in A_1 \wedge \ldots \wedge x_n \in A_n$$

- Equality:

$$(x_1, \ldots, x_n) = (y_1, \ldots, y_n) \;\Leftrightarrow\; x_1 = y_1 \wedge \ldots \wedge x_n = y_n$$

Cartesian product

- Suppose A and B are sets.
- The cartesian product $A \times B$ is the set of all tuples (x, y).
- x is an element of A and y is an element of B.

$$(x, y) \in A \times B \iff x \in A \wedge y \in B \qquad \texttt{mem\_Sigma\_iff}$$

- Membership:
- Equality:

Cartesian product

- Suppose A and B are sets.
- The cartesian product $A \times B$ is the set of all tuples (x, y).
- x is an element of A and y is an element of B.

$$(x, y) \in A \times B \ \Leftrightarrow \ x \in A \wedge y \in B$$ `mem_Sigma_iff`

- Membership:

$$(x_1, \ldots, x_n) \in A_1 \times \ldots \times A_n \ \Leftrightarrow \ x_1 \in A_1 \wedge \ldots \wedge x_n \in A_n$$

- Equality:

$$(x_1, \ldots, x_n) = (y_1, \ldots, y_n) \ \Leftrightarrow \ x_1 = y_1 \wedge \ldots \wedge x_n = y_n$$

Cartesian product

- Suppose A and B are sets.
- The cartesian product $A \times B$ is the set of all tuples (x, y).
- x is an element of A and y is an element of B.

$$(x, y) \in A \times B \;\Leftrightarrow\; x \in A \land y \in B \qquad \texttt{mem\_Sigma\_iff}$$

- Membership:

$$(x_1, \ldots, x_n) \in A_1 \times \ldots \times A_n \;\Leftrightarrow\; x_1 \in A_1 \land \ldots \land x_n \in A_n$$

- Equality:

$$(x_1, \ldots, x_n) = (y_1, \ldots, y_n) \;\Leftrightarrow\; x_1 = y_1 \land \ldots \land x_n = y_n$$

Cartesian product

- Suppose A and B are sets.
- The cartesian product $A \times B$ is the set of all tuples (x, y).
- x is an element of A and y is an element of B.

$$(x, y) \in A \times B \;\Leftrightarrow\; x \in A \wedge y \in B \qquad \texttt{mem\_Sigma\_iff}$$

- Membership:

$$(x_1, \ldots, x_n) \in A_1 \times \ldots \times A_n \;\Leftrightarrow\; x_1 \in A_1 \wedge \ldots \wedge x_n \in A_n$$

- Equality:

$$(x_1, \ldots, x_n) = (y_1, \ldots, y_n) \;\Leftrightarrow\; x_1 = y_1 \wedge \ldots \wedge x_n = y_n$$

Cartesian product

- Suppose A and B are sets.
- The cartesian product $A \times B$ is the set of all tuples (x, y).
- x is an element of A and y is an element of B.

$$(x, y) \in A \times B \;\Leftrightarrow\; x \in A \wedge y \in B \qquad \texttt{mem\_Sigma\_iff}$$

- Membership:

$$(x_1, \ldots, x_n) \in A_1 \times \ldots \times A_n \;\Leftrightarrow\; x_1 \in A_1 \wedge \ldots \wedge x_n \in A_n$$

- Equality:

$$(x_1, \ldots, x_n) = (y_1, \ldots, y_n) \;\Leftrightarrow\; x_1 = y_1 \wedge \ldots \wedge x_n = y_n$$

Outline

Computability

- Sets can be uncomputable, unlike lists and other algebraic datatypes.
- Distinguishes HOL from programming languages.
- HOL has mathematical real numbers, not just fixed- or floating-point.
- Can't compute `{0::nat..}`: it's unbounded and so infinite (try `value`).
- Can't compute `{0::real..1}` as real numbers aren't enumerable.

  $$\texttt{\{0::real..1\}} = \{0,\ 1,\ 0.1,\ 0.01,\ 0.001,\ \sqrt{2}/2,\ \pi/4,\ \cdots\}$$

- Reason symbolically using theorems.
- Can usually compute with finite sets.

Computability

- Sets can be uncomputable, unlike lists and other algebraic datatypes.
- Distinguishes HOL from programming languages.
- HOL has mathematical real numbers, not just fixed- or floating-point.
- Can't compute `{0::nat..}`: it's unbounded and so infinite (try `value`).
- Can't compute `{0::real..1}` as real numbers aren't enumerable.

$$\texttt{\{0::real..1\}} = \{0,\ 1,\ 0.1,\ 0.01,\ 0.001,\ \sqrt{2}/2,\ \pi/4,\ \cdots\}$$

- Reason symbolically using theorems.
- Can usually compute with finite sets.

Computability

- Sets can be uncomputable, unlike lists and other algebraic datatypes.
- Distinguishes HOL from programming languages.
- HOL has mathematical real numbers, not just fixed- or floating-point.
- Can't compute `{0::nat..}`: it's unbounded and so infinite (try `value`).
- Can't compute `{0::real..1}` as real numbers aren't enumerable.

  $$\texttt{\{0::real..1\}} = \{0,\ 1,\ 0.1,\ 0.01,\ 0.001,\ \sqrt{2}/2,\ \pi/4,\ \cdots\}$$

- Reason symbolically using theorems.
- Can usually compute with finite sets.

Computability

- Sets can be uncomputable, unlike lists and other algebraic datatypes.
- Distinguishes HOL from programming languages.
- HOL has mathematical real numbers, not just fixed- or floating-point.
- Can't compute `{0::nat..}`: it's unbounded and so infinite (try `value`).
- Can't compute `{0::real..1}` as real numbers aren't enumerable.

$$\texttt{\{0::real..1\}} = \{0,\ 1,\ 0.1,\ 0.01,\ 0.001,\ \sqrt{2}/2,\ \pi/4,\ \cdots\}$$

- Reason symbolically using theorems.
- Can usually compute with finite sets.

Computability

- Sets can be uncomputable, unlike lists and other algebraic datatypes.
- Distinguishes HOL from programming languages.
- HOL has mathematical real numbers, not just fixed- or floating-point.
- Can't compute `{0::nat..}`: it's unbounded and so infinite (try **`value`**).
- Can't compute `{0::real..1}` as real numbers aren't enumerable.

  $$\texttt{\{0::real..1\}} = \{0,\ 1,\ 0.1,\ 0.01,\ 0.001,\ \sqrt{2}/2,\ \pi/4,\ \cdots\}$$

- Reason symbolically using theorems.
- Can usually compute with finite sets.

Computability

- Sets can be uncomputable, unlike lists and other algebraic datatypes.
- Distinguishes HOL from programming languages.
- HOL has mathematical real numbers, not just fixed- or floating-point.
- Can't compute `{0::nat..}`: it's unbounded and so infinite (try **value**).
- Can't compute `{0::real..1}` as real numbers aren't enumerable.

  $$\texttt{\{0::real..1\}} = \{0,\ 1,\ 0.1,\ 0.01,\ 0.001,\ \sqrt{2}/2,\ \pi/4,\ \cdots\}$$

- Reason symbolically using theorems.
- Can usually compute with finite sets.

Computability

- Sets can be uncomputable, unlike lists and other algebraic datatypes.
- Distinguishes HOL from programming languages.
- HOL has mathematical real numbers, not just fixed- or floating-point.
- Can't compute `{0::nat..}`: it's unbounded and so infinite (try **`value`**).
- Can't compute `{0::real..1}` as real numbers aren't enumerable.

$$\texttt{\{0::real..1\}} = \{0,\ 1,\ 0.1,\ 0.01,\ 0.001,\ \sqrt{2}/2,\ \pi/4,\ \cdots\}$$

- Reason symbolically using theorems.
- Can usually compute with finite sets.

Computability

- Sets can be uncomputable, unlike lists and other algebraic datatypes.
- Distinguishes HOL from programming languages.
- HOL has mathematical real numbers, not just fixed- or floating-point.
- Can't compute `{0::nat..}`: it's unbounded and so infinite (try **`value`**).
- Can't compute `{0::real..1}` as real numbers aren't enumerable.

  $$\texttt{\{0::real..1\}} = \{0,\ 1,\ 0.1,\ 0.01,\ 0.001,\ \sqrt{2}/2,\ \pi/4,\ \cdots\}$$

- Reason symbolically using theorems.
- Can usually compute with finite sets.

Computability

- Sets can be uncomputable, unlike lists and other algebraic datatypes.
- Distinguishes HOL from programming languages.
- HOL has mathematical real numbers, not just fixed- or floating-point.
- Can't compute `{0::nat..}`: it's unbounded and so infinite (try **value**).
- Can't compute `{0::real..1}` as real numbers aren't enumerable.

$$\texttt{\{0::real..1\}} = \{0,\ 1,\ 0.1,\ 0.01,\ 0.001,\ \sqrt{2}/2,\ \pi/4,\ \cdots\}$$

- Reason symbolically using theorems.
- Can usually compute with finite sets.

Conclusion

This Lecture

- Set theory in Isabelle/HOL
- Finite sets
- Uncomputable objects

Next Lecture

- Foundations of types

Conclusion

This Lecture

- Set theory in Isabelle/HOL.
- Finite sets.
- Uncomputable objects.

Next Lecture

- Foundations of types.

Conclusion

This Lecture

- Set theory in Isabelle/HOL.
- Finite sets.
- Uncomputable objects.

Next Lecture

- Foundations of types.

Conclusion

This Lecture

- Set theory in Isabelle/HOL.
- Finite sets.
- Uncomputable objects.

Next Lecture

- Foundations of types.

Conclusion

This Lecture

- Set theory in Isabelle/HOL.
- Finite sets.
- Uncomputable objects.

Next Lecture

- Foundations of types

Conclusion

This Lecture

- Set theory in Isabelle/HOL.
- Finite sets.
- Uncomputable objects.

Next Lecture

- Foundations of types.

Conclusion

This Lecture

- Set theory in Isabelle/HOL.
- Finite sets.
- Uncomputable objects.

Next Lecture

- Foundations of types.