

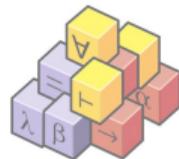
🍷 Sets in Isabelle/HOL 🍷

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 {})))
```

```
insert (insert A) = insert A insert (insert A) = insert A
```

```
insert (insert A) = insert (insert A) insert (insert A) = insert (insert 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 {})))
```

```
parts (insert A) = parts A ∪ {A}      insert A B = B
```

```
insert (insert A) = insert (insert A)  insert A B = B
```

- 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 {})))
```

```
inserts (insert A) = inserts A ∪ {A} insert A ∪ B = insert A (insert B A)
```

```
insert (insert A) = insert (insert A) insert (insert A) = insert A ∪ {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 {})))
```

- 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 {})))
```

- 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\_absorb2*

*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 {})))
```

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

`insert_absorb2`

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

`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 {})))
```

$insert\ x\ (insert\ x\ A) = insert\ x\ A$  `insert_absorb2`

$insert\ x\ (insert\ y\ A) = insert\ y\ (insert\ x\ A)$  `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  $x \notin S$ .

$$t \in \{u_1, \dots, u_n\} \leftrightarrow \exists i = 1, \dots, n. t = u_i$$

$$t \in \text{insert } a \text{ } B \leftrightarrow t = a \vee t \in B \quad \text{insert\_iff}$$

- Extensionality

$$A = B \leftrightarrow (\forall x. (x \in A) \leftrightarrow (x \in B)) \quad \text{set\_eq\_iff}$$

- Subset

$$A \subseteq B \leftrightarrow \forall x. (x \in A) \rightarrow (x \in B) \quad \text{subset\_eq}$$

$$A = B \leftrightarrow A \subseteq B \wedge B \subseteq A \quad \text{set\_eq\_subset}$$

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

$$(x \in \{\}) \equiv \text{False} \quad \text{empty\_iff}$$

# Membership and extension

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

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

$$t \in \text{insert } u \ S \Leftrightarrow t = u \vee t \in S \quad \text{insert\_iff}$$

- Extensionality

$$A = B \Leftrightarrow (\forall x (x \in A \rightarrow x \in B) \wedge (\forall x (x \in B \rightarrow x \in A))) \quad \text{set\_eq\_iff}$$

- Subset

$$A \subseteq B \Leftrightarrow \forall x (x \in A \rightarrow x \in B) \quad \text{subset\_eq}$$

$$A = B \Leftrightarrow A \subseteq B \wedge B \subseteq A \quad \text{set\_eq\_subset}$$

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

$$(x \in \{\}) \Leftrightarrow \text{false} \quad \text{empty\_iff}$$

# Membership and extension

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

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

$$t \in \text{insert } u \ S \Leftrightarrow t = u \vee t \in S \quad \text{insert\_iff}$$

- Extensionality

$$A = B \Leftrightarrow (\forall x (x \in A) \rightarrow (x \in B)) \wedge (\forall x (x \in B) \rightarrow (x \in A)) \quad \text{set\_eq\_iff}$$

- Subset

$$A \subseteq B \Leftrightarrow \forall x (x \in A \rightarrow x \in B) \quad \text{subset\_eq}$$

$$A = B \Leftrightarrow A \subseteq B \wedge B \subseteq A \quad \text{set\_eq\_subset}$$

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

$$(x \in \{\}) \Leftrightarrow \text{False} \quad \text{empty\_iff}$$

# Membership and extension

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

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

$$t \in \mathit{insert} \ u \ S \Leftrightarrow t = u \vee t \in S$$

`insert_iff`

- Extensionality

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

`set_eq_iff`

- Subset

$$A \subseteq B \Leftrightarrow \forall x. x \in A \Rightarrow x \in B$$

`subset_eq`

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

`set_eq_subset`

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

$$(\forall x. x \notin \{\})$$

`empty_iff`

# Membership and extension

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

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

$$t \in \text{insert } u S \Leftrightarrow t = u \vee t \in S$$

`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  $\emptyset$ ):

$$(\forall x. x \notin \{\})$$

`empty_iff`

# Membership and extension

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

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

$$t \in \text{insert } u S \Leftrightarrow t = u \vee t \in S$$

`insert_iff`

- **Extensionality**

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

`set_eq_iff`

- Subset

$$A \subseteq B \Leftrightarrow (\forall x. x \in A \Rightarrow x \in B)$$

`subset_eq`

$$A \subseteq B \Leftrightarrow (A \cap B = A)$$

`set_eq_subset`

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

$$\{x \in \{\} \} = \{\}$$

`empty_iff`

# Membership and extension

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

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

$$t \in \text{insert } u S \Leftrightarrow t = u \vee t \in S$$

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  $\emptyset$ ):

$$(x \in \{\}) \Leftrightarrow \text{false}$$

empty\_iff

# Membership and extension

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

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

$$t \in \mathit{insert} \ u \ S \Leftrightarrow t = u \vee t \in S \quad \mathit{insert\_iff}$$

- **Extensionality**

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

- **Subset**

$$A \subseteq B \Leftrightarrow \forall x \in A. x \in B \quad \mathit{subset\_eq}$$

$$A = B \Leftrightarrow A \subseteq B \wedge B \subseteq A \quad \mathit{set\_eq\_subset}$$

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

$$\{x\} \neq \{\} \quad \mathit{empty\_iff}$$

# Membership and extension

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

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

$$t \in \text{insert } u S \Leftrightarrow t = u \vee t \in S$$

`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  $\emptyset$ ):

$$\{x\} \cap \{\} = \{\}$$

`empty_iff`

# Membership and extension

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

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

$$t \in \text{insert } u S \Leftrightarrow t = u \vee t \in S \quad \text{insert\_iff}$$

- **Extensionality**

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

- **Subset**

$$A \subseteq B \Leftrightarrow \forall x \in A. x \in B \quad \text{subset\_eq}$$

$$A = B \Leftrightarrow A \subseteq B \wedge B \subseteq A \quad \text{set\_eq\_subset}$$

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

$$(c \in \{\}) = \text{False} \quad \text{empty\_iff}$$

# Membership and extension

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

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

$$t \in \text{insert } u S \Leftrightarrow t = u \vee t \in S \quad \text{insert\_iff}$$

- **Extensionality**

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

- **Subset**

$$A \subseteq B \Leftrightarrow \forall x \in A. x \in B \quad \text{subset\_eq}$$

$$A = B \Leftrightarrow A \subseteq B \wedge B \subseteq A \quad \text{set\_eq\_subset}$$

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

$$(c \in \{\}) = \text{False} \quad \text{empty\_iff}$$

## Set Deduction Rules (Selection)

$$x \notin \mathcal{V}(\Gamma) \frac{x \in A, \Gamma \vdash x \in B}{\Gamma \vdash A \subseteq B} \text{subsetI}$$

$$\frac{\Gamma \vdash t \in A \quad t \in B, \Gamma \vdash P}{A \subseteq B \vdash P} \text{subsetD}$$

$$\frac{\Gamma \vdash A \subseteq B \quad \Gamma \vdash B \subseteq A}{\Gamma \vdash A = B} \text{equalityI}$$

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

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

# Set Deduction Rules (Selection)

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

$$\frac{\Gamma \vdash t \in A \quad t \in B, \Gamma \vdash P}{A \subseteq B \vdash P} \text{subsetD}$$

$$\frac{\Gamma \vdash A \subseteq B \quad \Gamma \vdash B \subseteq A}{\Gamma \vdash A = B} \text{equalityI}$$

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

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

# Set Deduction Rules (Selection)

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

$$\frac{\Gamma \vdash t \in A \quad t \in B, \Gamma \vdash P}{A \subseteq B \vdash P} \text{subsetD}$$

$$\frac{\Gamma \vdash A \subseteq B \quad \Gamma \vdash B \subseteq A}{\Gamma \vdash A = B} \text{equalityI}$$

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

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

# Set Deduction Rules (Selection)

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

$$\frac{\Gamma \vdash t \in A \quad t \in B, \Gamma \vdash P}{A \subseteq B \vdash P} \text{subsetD}$$

$$\frac{\Gamma \vdash A \subseteq B \quad \Gamma \vdash B \subseteq A}{\Gamma \vdash A = B} \text{equalityI}$$

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

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

# Set Deduction Rules (Selection)

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

$$\frac{\Gamma \vdash t \in A \quad t \in B, \Gamma \vdash P}{A \subseteq B \vdash P} \text{subsetD}$$

$$\frac{\Gamma \vdash A \subseteq B \quad \Gamma \vdash B \subseteq A}{\Gamma \vdash A = B} \text{equalityI}$$

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

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

# Set Deduction Rules (Selection)

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

$$\frac{\Gamma \vdash t \in A \quad t \in B, \Gamma \vdash P}{A \subseteq B \vdash P} \text{subsetD}$$

$$\frac{\Gamma \vdash A \subseteq B \quad \Gamma \vdash B \subseteq A}{\Gamma \vdash A = B} \text{equalityI}$$

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

- 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:

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

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

# 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:

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

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

# 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:

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

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

# 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:

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

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

# 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:

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

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

# 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:

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

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

# 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:

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

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

# Set comprehension

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

$$x \in \{x \in S \mid P(x)\} \Leftrightarrow x \in S \wedge P(x)$$

- Term comprehension: set constructed from particular terms:

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

- Set comprehension is the **axiomatic** constructor for sets:

Collects:  $(\lambda a \Rightarrow \text{bool}) \Rightarrow \lambda a \text{ set}$

$\{x \mid P(x)\} = \text{Collect } P$

$x \in \{\text{Collect } P\} \Leftrightarrow P \ x$

$\text{Collect } P \subseteq A \Leftrightarrow P \ x \Rightarrow x \in A$

$\text{mem\_Collect\_eq}$

$\text{Collect\_mem\_eq}$

# 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:

`Collects ('a → bool) → 'a set`

`{x. P(x)} = Collect P`

`a ∈ Collect P` w/ `P a`

`Collect P x ∈ A` → `A`

`mem_Collect_eq`

`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:

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

- Set comprehension is the **axiomatic** constructor for sets:

`Collects ('a -> bool) -> 'a set`

`{x. P(x)} = Collect P`

`x ∈ Collect P`  $\Leftrightarrow$  `P x`

`Collect P`  $\subseteq$  `A`

`mem_Collect_eq`

`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`

`x ∈ Collect P` iff `P x`

`Collect P = {x. P(x)}`

`mem_Collect_eq`

`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 => b) : 'a set`

`forall Collect`

`set Collect`

`Collect_eq`

`forall Collect`

`set Collect_eq`

`Collect_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  $\Rightarrow$  bool)  $\Rightarrow$  'a set`

`{x. P(x)} = Collect P`

`a  $\in$  (Collect P)  $\Leftrightarrow$  P a`

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

`mem_Collect_eq`

`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  $\Rightarrow$  bool)  $\Rightarrow$  'a set`

`{x. P(x)} = Collect P`

`a  $\in$  (Collect P)  $\Leftrightarrow$  P a`

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

`mem_Collect_eq`

`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  $\Rightarrow$  bool)  $\Rightarrow$  'a set`

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

`$a \in (\text{Collect } P) \Leftrightarrow P\ a$`

`$\text{Collect } (\lambda x. x \in A) = A$`

`mem_Collect_eq`

`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  $\Rightarrow$  bool)  $\Rightarrow$  'a set`

`{x. P(x)} = Collect P`

`a  $\in$  (Collect P)  $\Leftrightarrow$  P a`

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

`mem_Collect_eq`

`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  $\Rightarrow$  bool)  $\Rightarrow$  'a set`

`{x. P(x)} = Collect P`

`a  $\in$  (Collect P)  $\Leftrightarrow$  P a`

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

`mem_Collect_eq`

`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} \subseteq {x. 0 < x \wedge x < 100}"
proof (rule subsetI)
 fix x :: nat
 assume "x \in {1, 3, 7, 9}"
 hence xs: "x = 1 \vee x = 3 \vee x = 7 \vee x = 9"
 by (simp add: insert_iff empty_iff)
 show "x \in {x. 0 < x \wedge 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
```

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

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

# Set Operators

- Set union:

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

Or iff

- Set intersection:

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

And iff

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

$$x \in (A \setminus B) \Leftrightarrow x \in A \wedge x \notin B$$

Diff iff

## Example

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

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

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

# 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:  $A \setminus B$ ):

$$x \in (A \setminus B) \Leftrightarrow x \in A \wedge x \notin B$$

Diff\_iff

## Example

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

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

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

# 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:  $A \setminus B$ ):

$$x \in (A \setminus B) \Leftrightarrow x \in A \wedge x \notin B$$

Diff\_iff

## Example

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

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

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

# 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:  $A \setminus B$ ):

$$x \in (A \setminus B) \Leftrightarrow x \in A \wedge x \notin B$$

Diff\_iff

## Example

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

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

$$\{\text{Atlantic}, \text{Indian}\} \setminus \{\text{Indian}, \text{Pacific}\} = \{\text{Atlantic}\}$$

# Set Operators

- Set union:

$$x \in (A \cup B) \Leftrightarrow x \in A \vee x \in B \quad \text{Un\_iff}$$

- Set intersection:

$$x \in (A \cap B) \Leftrightarrow x \in A \wedge x \in B \quad \text{Int\_iff}$$

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

$$x \in (A \setminus B) \Leftrightarrow x \in A \wedge x \notin B \quad \text{diff\_iff}$$

## Example

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

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

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

# Set Operators

- Set union:

$$x \in (A \cup B) \Leftrightarrow x \in A \vee x \in B \quad \text{Un\_iff}$$

- Set intersection:

$$x \in (A \cap B) \Leftrightarrow x \in A \wedge x \in B \quad \text{Int\_iff}$$

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

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

## Example

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

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

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

# Set Operators

- Set union:

$$x \in (A \cup B) \Leftrightarrow x \in A \vee x \in B \quad \text{Un\_iff}$$

- Set intersection:

$$x \in (A \cap B) \Leftrightarrow x \in A \wedge x \in B \quad \text{Int\_iff}$$

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

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

## Example

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

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

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

# Set Operators

- Set union:

$$x \in (A \cup B) \Leftrightarrow x \in A \vee x \in B \quad \text{Un\_iff}$$

- Set intersection:

$$x \in (A \cap B) \Leftrightarrow x \in A \wedge x \in B \quad \text{Int\_iff}$$

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

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

## Example

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

# Set Operators

- Set union:

$$x \in (A \cup B) \Leftrightarrow x \in A \vee x \in B \quad \text{Un\_iff}$$

- Set intersection:

$$x \in (A \cap B) \Leftrightarrow x \in A \wedge x \in B \quad \text{Int\_iff}$$

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

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

## Example

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

# Set Operators

- Set union:

$$x \in (A \cup B) \Leftrightarrow x \in A \vee x \in B \quad \text{Un\_iff}$$

- Set intersection:

$$x \in (A \cap B) \Leftrightarrow x \in A \wedge x \in B \quad \text{Int\_iff}$$

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

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

## Example

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

# Set Operators

- Set union:

$$x \in (A \cup B) \Leftrightarrow x \in A \vee x \in B \quad \text{Un\_iff}$$

- Set intersection:

$$x \in (A \cap B) \Leftrightarrow x \in A \wedge x \in B \quad \text{Int\_iff}$$

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

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

## Example

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

# Distributed Operators

- Distributed union:

$$\begin{aligned} \Pi(A, B, C, \dots) &= A \cup B \cup C \cup \dots \\ \pi(\pi(A) \cup \pi(B) \cup \dots) &= \pi(A \cup B \cup \dots) \end{aligned} \quad \text{Union\_iff}$$

- Distributed intersection:

$$\begin{aligned} \Pi(A, B, C, \dots) &= A \cap B \cap C \cap \dots \\ \pi(\pi(A) \cap \pi(B) \cap \dots) &= \pi(A \cap B \cap \dots) \end{aligned} \quad \text{Inter\_iff}$$

# Distributed Operators

- Distributed union:

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

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

Union\_iff

- Distributed intersection:

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

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

Inter\_iff

# Distributed Operators

- Distributed union:

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

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

Union\_iff

- Distributed intersection:

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

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

Inter\_iff

# Distributed Operators

- Distributed union:

$$\bigcup\{A, B, C, \dots\} = A \cup B \cup C \cup \dots$$
$$x \in \bigcup S \Leftrightarrow (\exists A \in S \bullet x \in A)$$

Union\_iff

- Distributed intersection:

$$\bigcap\{A, B, C, \dots\} = A \cap B \cap C \cap \dots$$
$$x \in \bigcap S \Leftrightarrow (\forall A \in S \bullet x \in A)$$

Inter\_iff

# Distributed Operators

- Distributed union:

$$\bigcup\{A, B, C, \dots\} = A \cup B \cup C \cup \dots$$
$$x \in \bigcup S \Leftrightarrow (\exists A \in S \bullet x \in A)$$

Union\_iff

- Distributed intersection:

$$\bigcap\{A, B, C, \dots\} = A \cap B \cap C \cap \dots$$
$$x \in \bigcap S \Leftrightarrow (\forall A \in S \bullet x \in A)$$

Inter\_iff

# Distributed Operators

- Distributed union:

$$\bigcup\{A, B, C, \dots\} = A \cup B \cup C \cup \dots$$
$$x \in \bigcup S \Leftrightarrow (\exists A \in S \bullet x \in A)$$

Union\_iff

- Distributed intersection:

$$\bigcap\{A, B, C, \dots\} = A \cap B \cap C \cap \dots$$
$$x \in \bigcap S \Leftrightarrow (\forall A \in S \bullet x \in A)$$

Inter\_iff

# Distributed Operators

- Distributed union:

$$\begin{aligned}\bigcup\{A, B, C, \dots\} &= A \cup B \cup C \cup \dots \\ x \in \bigcup S &\Leftrightarrow (\exists A \in S \bullet x \in A)\end{aligned}$$

Union\_iff

- Distributed intersection:

$$\begin{aligned}\bigcap\{A, B, C, \dots\} &= A \cap B \cap C \cap \dots \\ x \in \bigcap S &\Leftrightarrow (\forall A \in S \bullet x \in A)\end{aligned}$$

Inter\_iff

# Intervals

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

# 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.  $\{u_1, u_2, \dots, u_n\}$ .
- Non-finite sets are *infinite*. For example,  $\{0 :: \text{nat}.. \}$ .
- Finite characterised by `finite :: 'a set  $\Rightarrow$  bool` in HOL.

```
finite [] \Rightarrow true finite.emptyI
finite (insert x A) \Leftrightarrow finite A finite.insert
finite (A \cup B) \Leftrightarrow (finite A \wedge finite B) finite.Un
```

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

```
finite (set xs) \Leftrightarrow finite 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.  $\{u_1, u_2, \dots, u_n\}$ .
- Non-finite sets are *infinite*. For example,  $\{0 :: \text{nat}.. \}$ .
- Finite characterised by `finite :: 'a set  $\Rightarrow$  bool` in HOL.

```
finite [] = true
finite (insert x A) = finite A
finite (A U B) = (finite A \wedge finite B)
```

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

```
finite (set xs) = finite xs
set [] = {}
set (repeat x n) = {x}
set (repeat x n) = set (repeat x m)
```

# Finite sets

- **Finite set** has a finite number of elements, e.g.  $\{u_1, u_2, \dots, u_n\}$ .
- Non-finite sets are **infinite**. For example,  $\{0 :: \text{nat}.. \}$ .

- Finite characterised by `finite :: 'a set  $\Rightarrow$  bool` in HOL.

```
finite [] finite.emptyI
finite (insert x A) \Leftrightarrow finite A finite.insert
finite (A \cup B) \Leftrightarrow (finite A \wedge finite B) finite.Un
```

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

```
finite (set xs) finite.set
set [] = {} empty_set
finite (set xs) \Leftrightarrow finite (set xs) list.set
```

# Finite sets

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

```
finite {}
```

```
finite.emptyI
```

```
finite (insert x A) \Leftrightarrow finite A
```

```
finite_insert
```

```
finite (A \cup B) \Leftrightarrow (finite A \wedge finite B)
```

```
finite_Un
```

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

```
finite (set xs)
```

```
finite_set
```

```
set [] = {}
```

```
empty_set
```

```
finite (set (repeat 1000 1))
```

```
list_eqI
```

# Finite sets

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

```
finite {}
```

```
finite.emptyI
```

```
finite (insert x A) \Leftrightarrow finite A
```

```
finite_insert
```

```
finite (A \cup B) \Leftrightarrow (finite A \wedge finite B)
```

```
finite_Un
```

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

```
finite (set xs)
```

```
finite_set
```

```
set [] = {}
```

```
empty_set
```

```
set [x, x, x] = {x}
```

```
list_set
```

# Finite sets

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

`finite {}`

`finite.emptyI`

`finite (insert x A)  $\Leftrightarrow$  finite A`

`finite_insert`

`finite (A  $\cup$  B)  $\Leftrightarrow$  (finite A  $\wedge$  finite B)`

`finite_Un`

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

`set [1, 2, 1, 2, 1]`

`finite_set`

`set []`

`empty_set`

`set [1, 2, 1, 2, 1] = set [1, 2]`

`list_of_set`

# Finite sets

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

|                                                                                                                  |                            |
|------------------------------------------------------------------------------------------------------------------|----------------------------|
| <code>finite {}</code>                                                                                           | <code>finite.emptyI</code> |
| <code>finite (insert x A) <math>\Leftrightarrow</math> finite A</code>                                           | <code>finite_insert</code> |
| <code>finite (A <math>\cup</math> B) <math>\Leftrightarrow</math> (finite A <math>\wedge</math> finite B)</code> | <code>finite_Un</code>     |

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

|                                  |                         |
|----------------------------------|-------------------------|
| <code>set [1, 2, 3, 2, 1]</code> | <code>finite_set</code> |
| <code>set []</code>              | <code>empty_set</code>  |
| <code>set [1, 2, 3]</code>       | <code>finite_set</code> |

# Finite sets

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

`finite {}` `finite.emptyI`

`finite (insert x A)  $\Leftrightarrow$  finite A` `finite_insert`

`finite (A  $\cup$  B)  $\Leftrightarrow$  (finite A  $\wedge$  finite B)` `finite_Un`

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

# Finite sets

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

|                                                                                                                  |                            |
|------------------------------------------------------------------------------------------------------------------|----------------------------|
| <code>finite {}</code>                                                                                           | <code>finite.emptyI</code> |
| <code>finite (insert x A) <math>\Leftrightarrow</math> finite A</code>                                           | <code>finite_insert</code> |
| <code>finite (A <math>\cup</math> B) <math>\Leftrightarrow</math> (finite A <math>\wedge</math> finite B)</code> | <code>finite_Un</code>     |

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

|                                               |                         |
|-----------------------------------------------|-------------------------|
| <code>finite (set xs)</code>                  | <code>finite_set</code> |
| <code>set [] = {}</code>                      | <code>empty_set</code>  |
| <code>set (x # xs) = insert x (set xs)</code> | <code>list.set</code>   |

# Finite sets

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

|                                                                                                                  |                            |
|------------------------------------------------------------------------------------------------------------------|----------------------------|
| <code>finite {}</code>                                                                                           | <code>finite.emptyI</code> |
| <code>finite (insert x A) <math>\Leftrightarrow</math> finite A</code>                                           | <code>finite_insert</code> |
| <code>finite (A <math>\cup</math> B) <math>\Leftrightarrow</math> (finite A <math>\wedge</math> finite B)</code> | <code>finite_Un</code>     |

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

|                                               |                         |
|-----------------------------------------------|-------------------------|
| <code>finite (set xs)</code>                  | <code>finite_set</code> |
| <code>set [] = {}</code>                      | <code>empty_set</code>  |
| <code>set (x # xs) = insert x (set xs)</code> | <code>list.set</code>   |

# Finite sets

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

|                                                                                                                  |                            |
|------------------------------------------------------------------------------------------------------------------|----------------------------|
| <code>finite {}</code>                                                                                           | <code>finite.emptyI</code> |
| <code>finite (insert x A) <math>\Leftrightarrow</math> finite A</code>                                           | <code>finite_insert</code> |
| <code>finite (A <math>\cup</math> B) <math>\Leftrightarrow</math> (finite A <math>\wedge</math> finite B)</code> | <code>finite_Un</code>     |

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

|                                               |                         |
|-----------------------------------------------|-------------------------|
| <code>finite (set xs)</code>                  | <code>finite_set</code> |
| <code>set [] = {}</code>                      | <code>empty_set</code>  |
| <code>set (x # xs) = insert x (set xs)</code> | <code>list.set</code>   |

# Finite sets

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

|                                                                                                                  |                            |
|------------------------------------------------------------------------------------------------------------------|----------------------------|
| <code>finite {}</code>                                                                                           | <code>finite.emptyI</code> |
| <code>finite (insert x A) <math>\Leftrightarrow</math> finite A</code>                                           | <code>finite_insert</code> |
| <code>finite (A <math>\cup</math> B) <math>\Leftrightarrow</math> (finite A <math>\wedge</math> finite B)</code> | <code>finite_Un</code>     |

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

|                                               |                         |
|-----------------------------------------------|-------------------------|
| <code>finite (set xs)</code>                  | <code>finite_set</code> |
| <code>set [] = {}</code>                      | <code>empty_set</code>  |
| <code>set (x # xs) = insert x (set xs)</code> | <code>list.set</code>   |

# Power set

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

$x \in \mathbb{P} A \iff x \subseteq A$

`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

$\mathbb{F} A = \{X, X \subseteq A \mid \text{finite } X\}$

`Fpow_def`

$A \subseteq \mathbb{P} A$

`Fpow_subset_Pow`

# Power set

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

$$S \in \mathbb{P} A \Leftrightarrow S \subseteq A$$

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

$$\mathbb{F} A = \{X \mid X \subseteq A \text{ finite}\}$$

Fpow\_def

$$\mathbb{F} A \subseteq \mathbb{P} A$$

Fpow\_subset\_Pow

# Power set

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

$$S \in \mathbb{P} A \Leftrightarrow S \subseteq A$$

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

$$\mathbb{F} A = \{X \mid X \subseteq A \text{ finite}\}$$

$$\mathbb{F} A \subseteq \mathbb{P} A$$

Fpow\_def

Fpow\_subset\_Pow

# Power set

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

$$S \in \mathbb{P} A \Leftrightarrow S \subseteq A$$

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

$$\mathbb{F} A = \{X \mid X \subseteq A \text{ finite}\}$$

$$X \subseteq \mathbb{F} A$$

Fpow\_def

Pow\_subset\_Pow

# Power set

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

$$S \in \mathbb{P} A \Leftrightarrow S \subseteq A$$

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

$$\mathbb{F} A = \bigcup_{n \in \mathbb{N}} \mathbb{P}^n A$$

$$A \subseteq \mathbb{P} A$$

$$\mathbb{F} \text{pow\_def}$$

$$\text{Pow\_subset\_Pow}$$

# Power set

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

$$S \in \mathbb{P} A \Leftrightarrow S \subseteq A$$

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

`F A =  $\mathbb{F} A$  Pow F`

`Fpow_def`

`F A  $\subseteq$  Pow F A`

`Pow_subset_Pow`

# Power set

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

$$S \in \mathbb{P} A \Leftrightarrow S \subseteq A$$

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

$$\mathbb{F} A = \{X. X \subseteq A \wedge \text{finite } X\}$$

Fpow\_def

$$\mathbb{F} A \subseteq \mathbb{P} A$$

Fpow\_subset\_Pow

# Power set

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

$$S \in \mathbb{P} A \Leftrightarrow S \subseteq A$$

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

$$\mathbb{F} A = \{X. X \subseteq A \wedge \text{finite } X\}$$

Fpow\_def

$$\mathbb{F} A \subseteq \mathbb{P} A$$

Fpow\_subset\_Pow

# Power set

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

$$S \in \mathbb{P} A \Leftrightarrow S \subseteq A$$

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

$$\mathbb{F} A = \{X. X \subseteq A \wedge \text{finite } X\}$$

Fpow\_def

$$\mathbb{F} A \subseteq \mathbb{P} A$$

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  $B$ .

$$(x, y) \in A \times B \Leftrightarrow x \in A \wedge y \in B \quad \text{non\_Sigma\_iff}$$

- Membership:

$$(x_1, y_1) \in A_1 \times B_1 \wedge (x_2, y_2) \in A_2 \times B_2 \Leftrightarrow x_1 \in A_1 \wedge y_1 \in B_1 \wedge x_2 \in A_2 \wedge y_2 \in B_2$$

- Equality:

$$(x_1, y_1, z_1) = (x_2, y_2, z_2) \Leftrightarrow x_1 = x_2 \wedge y_1 = y_2 \wedge z_1 = z_2$$

# 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  $B$ .

$$(x, y) \in A \times B \Leftrightarrow x \in A \wedge y \in B \quad \text{non\_Sigma\_iff}$$

- Membership:

$$(x_1, y_1) \in A_1 \times B_1 \wedge (x_2, y_2) \in A_2 \times B_2 \Leftrightarrow x_1 \in A_1 \wedge y_1 \in B_1 \wedge x_2 \in A_2 \wedge y_2 \in B_2$$

- Equality:

$$(x_1, y_1) = (x_2, y_2) \Leftrightarrow x_1 = x_2 \wedge y_1 = y_2$$

# 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  $B$ .

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

non-Sigma\_1 FF

- Membership:

$$(x_1, y_1) \in A_1 \times B_1 \wedge (x_2, y_2) \in A_2 \times B_2 \Leftrightarrow x_1 \in A_1 \wedge x_2 \in A_2 \wedge y_1 \in B_1 \wedge y_2 \in B_2$$

- Equality:

$$(x_1, y_1) = (x_2, y_2) \Leftrightarrow x_1 = x_2 \wedge y_1 = y_2$$

# 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  $B$ .

$$(x, y) \in A \times B \Leftrightarrow x \in A \wedge y \in B \quad \text{mem\_Sigma\_iff}$$

- Membership:

$$(a_1, a_2) \in A_1 \times A_2 \Leftrightarrow a_1 \in A_1 \wedge a_2 \in A_2$$

- Equality:

$$(a_1, a_2) = (a_3, a_4) \Leftrightarrow a_1 = a_3 \wedge a_2 = a_4$$

# 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:

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

- Equality:

$$A \times B = C \times D \Leftrightarrow A = C \wedge B = D$$

# 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  $B$ .

$$(x, y) \in A \times B \Leftrightarrow x \in A \wedge y \in B \quad \text{mem\_Sigma\_iff}$$

- **Membership:**

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

- **Equality:**

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

# 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  $B$ .

$$(x, y) \in A \times B \Leftrightarrow x \in A \wedge y \in B \quad \text{mem\_Sigma\_iff}$$

- **Membership:**

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

- 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  $B$ .

$$(x, y) \in A \times B \Leftrightarrow x \in A \wedge y \in B \quad \text{mem\_Sigma\_iff}$$

- **Membership:**

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

- **Equality:**

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

# 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  $B$ .

$$(x, y) \in A \times B \Leftrightarrow x \in A \wedge y \in B \quad \text{mem\_Sigma\_iff}$$

- **Membership:**

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

- **Equality:**

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

# Outline

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

# 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  $\{0::\text{nat}.. \}$ : it's unbounded and so infinite (try **value**).
- Can't compute  $\{0::\text{real}..1\}$  as real numbers aren't **enumerable**.  
$$\{0::\text{real}..1\} = \{0, 1, 0.1, 0.01, 0.001, \sqrt{2}/2, \pi/4, \dots\}$$
- 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  $\{0::\text{nat}.. \}$ : it's unbounded and so infinite (try **value**).
- Can't compute  $\{0::\text{real}..1\}$  as real numbers aren't **enumerable**.  
$$\{0::\text{real}..1\} = \{0, 1, 0.1, 0.01, 0.001, \sqrt{2}/2, \pi/4, \dots\}$$
- 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  $\{0::\text{nat}.. \}$ : it's unbounded and so infinite (try **value**).
- Can't compute  $\{0::\text{real}..1\}$  as real numbers aren't **enumerable**.  
$$\{0::\text{real}..1\} = \{0, 1, 0.1, 0.01, 0.001, \sqrt{2}/2, \pi/4, \dots\}$$
- 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  $\{0::\text{nat}.. \}$ : it's unbounded and so infinite (try **value**).
- Can't compute  $\{0::\text{real}..1\}$  as real numbers aren't **enumerable**.  
 $\{0::\text{real}..1\} = \{0, 1, 0.1, 0.01, 0.001, \sqrt{2}/2, \pi/4, \dots\}$
- 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  $\{0::\text{nat}.. \}$ : it's unbounded and so infinite (try **value**).
- Can't compute  $\{0::\text{real}..1\}$  as real numbers aren't **enumerable**.  
 $\{0::\text{real}..1\} = \{0, 1, 0.1, 0.01, 0.001, \sqrt{2}/2, \pi/4, \dots\}$
- 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  $\{0::\text{nat}.. \}$ : it's unbounded and so infinite (try **value**).
- Can't compute  $\{0::\text{real}..1\}$  as real numbers aren't **enumerable**.

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

- 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  $\{0::\text{nat}.. \}$ : it's unbounded and so infinite (try **value**).
- Can't compute  $\{0::\text{real}..1\}$  as real numbers aren't **enumerable**.

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

- 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  $\{0::\text{nat}.. \}$ : it's unbounded and so infinite (try **value**).
- Can't compute  $\{0::\text{real}..1\}$  as real numbers aren't **enumerable**.

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

- 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  $\{0::\text{nat}..\}$ : it's unbounded and so infinite (try **value**).
- Can't compute  $\{0::\text{real}..1\}$  as real numbers aren't **enumerable**.

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

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

# Conclusion

## This Lecture

- Semantics in Isabelle/HOL
- Finite sets
- Unconquered 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

□ [Uncomputable objects](#)

# 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.