Interactive Formal Verification

6: Sets

Tjark Weber
(Slides: Lawrence C Paulson)
Computer Laboratory
University of Cambridge
Set Notation in Isabelle
Set Notation in Isabelle

• Set notation is crucial to mathematical discourse.
Set Notation in Isabelle

• Set notation is crucial to mathematical discourse.

• Set-theoretic abstractions naturally express many complex constructions.
Set Notation in Isabelle

- Set notation is crucial to mathematical discourse.
- Set-theoretic abstractions naturally express many complex constructions.
- A set in higher-order logic is a boolean-valued map.
Set Notation in Isabelle

- Set notation is crucial to mathematical discourse.
- Set-theoretic abstractions naturally express many complex constructions.
- A set in higher-order logic is a boolean-valued map.
- Its elements must all have the same type.
Set Theory Primitives
Set Theory Primitives

- The type $\alpha \text{ set}$, which abbreviates $\alpha \Rightarrow \text{bool}$
Set Theory Primitives

- The type $\alpha \text{set}$, which abbreviates $\alpha \Rightarrow \text{bool}$
- The membership relation: $\in$
Set Theory Primitives

- The type $\alpha \text{ set}$, which abbreviates $\alpha \Rightarrow \text{bool}$
- The membership relation: $\in$
- The subset relation: $\subseteq$
  - Reflexive, anti-symmetric, transitive
Set Theory Primitives

- The type $\alpha$ set, which abbreviates $\alpha \Rightarrow \text{bool}$
- The membership relation: $\in$
- The subset relation: $\subseteq$
  - Reflexive, anti-symmetric, transitive
- The empty set: $\{\}$
Set Theory Primitives

- The type $\alpha \text{ set}$, which abbreviates $\alpha \Rightarrow \text{bool}$
- The membership relation: $\in$
- The subset relation: $\subseteq$
  - Reflexive, anti-symmetric, transitive
- The empty set: $\{}$
- The universal set: $\text{UNIV}$
Basic Set Theory Operations
Basic Set Theory Operations

\[ e \in \{x. P(x)\} \iff P(e) \]
Basic Set Theory Operations

\[ e \in \{ x. \ P(x) \} \iff P(e) \]
\[ e \in \{ x \in A. \ P(x) \} \iff e \in A \land P(e) \]
Basic Set Theory Operations

\[ e \in \{ x \mid P(x) \} \iff P(e) \]
\[ e \in \{ x \in A \mid P(x) \} \iff e \in A \land P(e) \]
\[ e \in \neg A \iff e \notin A \]
Basic Set Theory Operations

\[ e \in \{ x. \ P(x) \} \iff P(e) \]
\[ e \in \{ x \in A. \ P(x) \} \iff e \in A \land P(e) \]
\[ e \in -A \iff e \notin A \]
\[ e \in A \cup B \iff e \in A \lor e \in B \]
Basic Set Theory Operations

\[ e \in \{x. \, P(x)\} \iff P(e) \]
\[ e \in \{x \in A. \, P(x)\} \iff e \in A \land P(e) \]
\[ e \in -A \iff e \notin A \]
\[ e \in A \cup B \iff e \in A \lor e \in B \]
\[ e \in A \cap B \iff e \in A \land e \in B \]
Basic Set Theory Operations

\[ e \in \{x. P(x)\} \iff P(e) \]
\[ e \in \{x \in A. P(x)\} \iff e \in A \land P(e) \]
\[ e \in -A \iff e \notin A \]
\[ e \in A \cup B \iff e \in A \lor e \in B \]
\[ e \in A \cap B \iff e \in A \land e \in B \]
\[ e \in \text{Pow}(A) \iff e \subseteq A \]
Big Union and Intersection
Big Union and Intersection

\[ e \in \left( \bigcup x. B(x) \right) \iff \exists x. e \in B(x) \]
Big Union and Intersection

\[ e \in \left( \bigcup x. B(x) \right) \iff \exists x. e \in B(x) \]

\[ e \in \left( \bigcup_{x \in A} B(x) \right) \iff \exists x \in A. e \in B(x) \]
Big Union and Intersection

\[ e \in \left( \bigcup x. B(x) \right) \iff \exists x. e \in B(x) \]
\[ e \in \left( \bigcup x \in A. B(x) \right) \iff \exists x \in A. e \in B(x) \]
\[ e \in \bigcup A \iff \exists x \in A. e \in x \]
Big Union and Intersection

\[ e \in \left( \bigcup x. B(x) \right) \iff \exists x. e \in B(x) \]
\[ e \in \left( \bigcup x \in A. B(x) \right) \iff \exists x \in A. e \in B(x) \]
\[ e \in \bigcup A \iff \exists x \in A. e \in x \]

And the analogous forms of intersections...
A Simple Set Theory Proof

lemma 

"\((\bigcap x \in A \cup B. \ C \ x \cup D) = (\bigcap x \in A. \ C \ x) \cap (\bigcap x \in B. \ C \ x) \cup D)"\n
apply auto

proof (prove): step 0

goal (1 subgoal):
1. \((\bigcap x \in A \cup B. \ C \ x \cup D) = (\bigcap x \in A. \ C \ x) \cap (\bigcap x \in B. \ C \ x) \cup D\n
*goals*
A Simple Set Theory Proof

plain ASCII syntax is an alternative to special symbols.
A Simple Set Theory Proof

lemma "(INT x: A Un B. C x Un D) = ((INT x: A. C x) Int (INT x: B. C x)) Un D"
apply auto

proof (prove): step 1

goal:
No subgoals!
Functions
Functions

\[ e \in (f' \ A) \iff \exists x \in A. \ e = f(x) \]
Functions

\[ e \in (f^\prime A) \iff \exists x \in A. \ e = f(x) \]

\[ e \in (f^{-1} A) \iff f(e) \in A \]
Functions

\[ e \in (f^\prime A) \iff \exists x \in A. \ e = f(x) \]

\[ e \in (f^{-\prime} A) \iff f(e) \in A \]

\[ f(x:=y) = (\lambda z. \text{if } z = x \text{ then } y \text{ else } f(z)) \]
Functions

\[ e \in (f^\prime A) \iff \exists x \in A. \ e = f(x) \]

\[ e \in (f^{-1} A) \iff f(e) \in A \]

\[ f(x:=y) = (\lambda z. \text{if } z = x \text{ then } y \text{ else } f(z)) \]

• Also inj, surj, bij, inv, etc. (injective,...)
Functions

\[ e \in (f^{\uparrow}A) \iff \exists x \in A. e = f(x) \]

\[ e \in (f^{-\uparrow}A) \iff f(e) \in A \]

\[ f(x:=y) = (\lambda z. \text{if } z = x \text{ then } y \text{ else } f(z)) \]

- Also inj, surj, bij, inv, etc. (injective,...)
- Don’t re-invent image and inverse image!!
Finite Set Notation
Finite Set Notation

\[ \{a_1, \ldots, a_n\} = \text{insert}(a_1, \ldots, \text{insert}(a_n, \{\})) \]
Finite Set Notation

\[ \{a_1, \ldots, a_n\} = \text{insert}(a_1, \ldots, \text{insert}(a_n, \{\})) \]

\[ e \in \text{insert}(a, B) \iff e = a \lor e \in B \]
Finite Sets

A finite set is defined inductively in terms of \{
} and insert
Finite Sets

A finite set is defined \textit{inductively} in terms of \{\} and \texttt{insert}

\[
\text{finite}(A \cup B) = (\text{finite } A \land \text{finite } B)
\]
Finite Sets

A finite set is defined \textit{inductively} in terms of \{\} and \textit{insert}

\[
\text{finite}(A \cup B) = (\text{finite } A \land \text{finite } B)
\]

\[
\text{finite } A \iff \text{card}(\text{Pow } A) = 2^{\text{card } A}
\]
Intervals, Sums and Products
Intervals, Sums and Products

\[
\{..<u\} \leftrightarrow \{x. \ x < u\} \\
\{..u\} \leftrightarrow \{x. \ x \leq u\} \\
\{l<..\} \leftrightarrow \{x. \ l < x\} \\
\{l..\} \leftrightarrow \{x. \ l \leq x\} \\
\{l<..<u\} \leftrightarrow \{l<..\} \cap \{..<u\} \\
\{l..<u\} \leftrightarrow \{l..\} \cap \{..<u\}
\]
Intervals, Sums and Products

\{..<u\} == \{x. x < u\}
\{..u\} == \{x. x \leq u\}
\{l<..\} == \{x. l<x\}
\{l..\} == \{x. l\leq x\}
\{l<..<u\} == \{l<..\} \cap \{..<u\}
\{l..<u\} == \{l..\} \cap \{..<u\}

\text{setsum } f \ A \text{ and } \text{setprod } f \ A
Intervals, Sums and Products

\{..<u\} \equiv \{x. \ x < \ u\}
\{..u\} \equiv \{x. \ x \leq \ u\}
\{l<..\} \equiv \{x. \ l<x\}
\{l..\} \equiv \{x. \ l\leq x\}
\{l<..<u\} \equiv \{l<..\} \cap \{..<u\}
\{l..<u\} \equiv \{l..\} \cap \{..<u\}

\text{setsum } f \ A \ and \ \text{setprod } f \ A
\Sigma_{i \in I.} f \ and \ \Pi_{i \in I.} f
A Harder Proof Involving Sets

```isar
lemma
  fixes c :: "real"
  shows "finite A \implies \text{setsum} (\%x. c \times f x) A = c \times \text{setsum} f A"
apply (induct A rule: finite_induct)
apply auto
apply (auto simp add: algebra_simps)
done
```

-proof (prove): step 0-

-goal (1 subgoal):
  1. finite A \implies (\sum_{x \in A} c \times f x) = c \times \text{setsum} f A
```

**goals**
tool-bar undo
A Harder Proof Involving Sets

```plaintext
lemma
gives c :: "real"
says "finite A \implies \text{setsum} (\%x. c * f x) A = c * \text{setsum} f A"

apply (induct A rule: finite_induct)
apply auto
apply (auto simp add: algebra_simps)
done

proof (prove): step 0

goal (1 subgoal):
  1. finite A \implies (\Sigma x \in A. c * f x) = c * \text{setsum} f A
```
A Harder Proof Involving Sets

- lemma fixes c :: "real"
  shows "finite A ⟹ setsum (%x. c * f x) A = c * setsum f A"

- apply (induct A rule: finite_induct)
- apply auto
- apply (auto simp add: algebra_simps)
- done

proof (prove): step 0

goal (1 subgoal):
1. finite A ⟹ (∑x∈A. c * f x) = c * setsum f A
Outcome of the Induction

```plaintext
lemma
  fixes c :: "real"
  shows "finite A \implies \text{setsum} (\%x. c \cdot f x) A = c \cdot \text{setsum} f A"
apply (induct A rule: finite_induct)
apply auto
apply (auto simp add: algebra_simps)
done
```

```
proof (prove): step 1

goal (2 subgoals):
1. \(\sum_{x \in \{\}} c \cdot f x = c \cdot \text{setsum} f \{\}\)
2. \(\forall x. F \Rightarrow (\text{finite } F; x \notin F; (\sum_{x \in F} c \cdot f x) = c \cdot \text{setsum} f F)\)
   \(\Rightarrow (\sum_{x \in \text{insert } x F} c \cdot f x) = c \cdot \text{setsum} f (\text{insert } x F)\)
```

```
Outcome of the Induction

base case: A is empty
Outcome of the Induction

base case: A is empty

inductive step: A = insert x F
Almost There!
Almost There!

```
lemma
    fixes c :: "real"
    shows "finite A \implies setsum (%x. c * f x) A = c * setsum f A"
apply (induct A rule: finite_induct)
apply auto
  apply (auto simp add: algebra_simps)
done
```

proof (prove): step 2

goal (1 subgoal):
1. \( \forall x. F \Rightarrow \left( \frac{\sum x \in F. c \cdot f x}{c \cdot \left(f x + \text{setsum } f\text{ F}\right)} \right) \)

need to apply a distributive law
Almost There!

need to apply a distributive law
Finished!

```
lemma
  fixes c :: "real"
  shows "finite A \implies setsum (%x. c * f x) A = c * setsum f A"
apply (induct A rule: finite_induct)
apply auto
apply (auto simp add: algebra_simps)
done
```

```
proof (prove): step 3

goal:
No subgoals!
```
Finished!

No need for the first "auto"...
Proving Theorems about Sets
Proving Theorems about Sets

- It is not practical to learn all the built-in lemmas.
Proving Theorems about Sets

• It is not practical to learn all the built-in lemmas.
• Instead, try an automatic proof method:
  • auto
  • force
  • blast
Proving Theorems about Sets

• It is not practical to learn all the built-in lemmas.

• Instead, try an automatic proof method:
  • auto
  • force
  • blast

• Each uses the built-in library, comprising hundreds of facts, with powerful heuristics.
Finding Theorems about Sets

```plaintext
lemma fixes c :: "real"
  shows "finite A ==> setsum (%x. c * f x) A = c * setsum f A"
apply (induct A rule: finite_induct)
apply auto
apply (auto simp add: algebra_simps)
done
```

```plaintext
```

```plaintext
*response*  All L1 (Isar Messages Utoks Abbrev;)
```
Finding Theorems about Sets

Step 1: click this button!

```
lemma
  fixes c :: "real"
  shows "finite A --> setsum (%x. c * f x) A = c * setsum f A"
apply (induct A rule: finite_induct)
apply auto
apply (auto simp add: algebra_simps)
done
```

```
lemma finite ?A --> (\sum x\in?A. ?c * ?f x) = ?c * setsum ?f ?A
```

```
*response* All L1 (Isar Messages Utoks Abbrev;)
```
Finding Theorems about Sets

Step 2: type some patterns
What Theorems Were Found?

searched for:
"_ ∪ _"
"_ ∩ _"
"card"

found 2 theorems in 0.120 secs:

Finite_Set.card_Un_Int:
  [finite ?A; finite ?B]

Finite_Set.card_Un_disjoint: