COMP4161: Advanced Topics in Software Verification

Gerwin Klein, Johannes Åman Pohjola, Christine Rizkallah, Miki Tanaka
T3/2020
Content

→ Foundations & Principles
  • Intro, Lambda calculus, natural deduction [1,2]
  • Higher Order Logic, Isar (part 1) [2,3a]
  • Term rewriting [3,4]

→ Proof & Specification Techniques
  • Inductively defined sets, rule induction, datatype induction, primitive recursion [4,5]
  • General recursive functions, termination proofs [7b]
  • Proof automation, Hoare logic, proofs about programs, invariants [8]
  • C verification [9,10]
  • Practice, questions, exam prep [10c]

a1 due; a2 due; a3 due
Last Time

- Conditional term rewriting
Last Time

- Conditional term rewriting
- Case Splitting with the simplifier
Last Time

- Conditional term rewriting
- Case Splitting with the simplifier
- Congruence rules
Last Time

- Conditional term rewriting
- Case Splitting with the simplifier
- Congruence rules
- AC Rules
Last Time

- Conditional term rewriting
- Case Splitting with the simplifier
- Congruence rules
- AC Rules
- Knuth-Bendix Completion (Waldmeister)
Last Time

- Conditional term rewriting
- Case Splitting with the simplifier
- Congruence rules
- AC Rules
- Knuth-Bendix Completion (Waldmeister)
- Orthogonal Rewrite Systems
Specification Techniques

Sets
Sets in Isabelle

Type 'a set: sets over type 'a
Sets in Isabelle

Type 'a set: sets over type 'a

→ {}, {e₁, ..., eₙ}, {x. P x}
Sets in Isabelle

Type 'a set: sets over type 'a

\[ \{\}, \{e_1, \ldots, e_n\}, \{x. P x\} \]

\[ e \in A, \quad A \subseteq B \]
Sets in Isabelle

Type 'a set: sets over type 'a

- \{\}, \{e_1, \ldots, e_n\}, \{x. P x\}
- e \in A, A \subseteq B
- A \cup B, A \cap B, A - B, -A
Sets in Isabelle

Type 'a set: sets over type 'a

→ \{\}, \{e_1, \ldots, e_n\}, \{x. P x\}
→ e \in A, A \subseteq B
→ A \cup B, A \cap B, A - B, -A
→ \bigcup x \in A. B x, \bigcap x \in A. B x, \bigcap A, \bigcup A
Sets in Isabelle

Type 'a set: sets over type 'a

- \{\}, \{e_1, \ldots, e_n\}, \{x. P x\}
- e \in A, \ A \subseteq B
- A \cup B, \ A \cap B, \ A - B, \ -A
- \bigcup_{x \in A. B} x, \ \bigcap_{x \in A. B} x, \ \bigcap A, \ \bigcup A
- \{i..j\}
Sets in Isabelle

Type 'a set: sets over type 'a

→ \{\}, \{e_1, \ldots, e_n\}, \{x. P x\}
→ e \in A, \quad A \subseteq B
→ A \cup B, \quad A \cap B, \quad A \setminus B, \quad \neg A
→ \bigcup x \in A. B \ x, \quad \bigcap x \in A. B \ x, \quad \bigcap A, \quad \bigcup A
→ \{i..j\}
→ insert :: \alpha \Rightarrow \alpha set \Rightarrow \alpha set
Sets in Isabelle

Type 'a set: sets over type 'a

→ \{\}, \{e_1, \ldots, e_n\}, \{x. P x\}
→ e \in A, \ A \subseteq B
→ A \cup B, \ A \cap B, \ A - B, \ -A
→ \bigcup x \in A. \ B \ x, \ \bigcap x \in A. \ B \ x, \ \bigcap A, \ \bigcup A
→ \{i..j\}
→ insert :: \alpha \Rightarrow \alpha \ set \Rightarrow \alpha \ set
→ f 'A \equiv \{y. \exists x \in A. \ y = f \ x\}
→ \ldots
Proofs about Sets

Natural deduction proofs:

→ equality\text{-}I: [A \subseteq B; B \subseteq A] \implies A = B
Proofs about Sets

Natural deduction proofs:

→ equalityI: \([A \subseteq B; B \subseteq A] \implies A = B\)

→ subsetI: \((\forall x. x \in A \implies x \in B) \implies A \subseteq B\)
Proofs about Sets

Natural deduction proofs:

→ equalityI: \[ A \subseteq B; \ B \subseteq A \] \implies A = B
→ subsetI: (\bigwedge x. x \in A \implies x \in B) \implies A \subseteq B
→ ... find_theorems
Bounded Quantifiers

\[ \forall x \in A. \ P x \]
Bounded Quantifiers

\[ \forall x \in A. \ P x \equiv \forall x. \ x \in A \rightarrow P x \]
Bounded Quantifiers

→ ∀x ∈ A. P x ≡ ∀x. x ∈ A → P x
→ ∃x ∈ A. P x
Bounded Quantifiers

\[ \forall x \in A. \ P \ x \equiv \forall x. \ x \in A \rightarrow P \ x \]

\[ \exists x \in A. \ P \ x \equiv \exists x. \ x \in A \land P \ x \]
Bounded Quantifiers

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

$$\exists x \in A. \ P x \equiv \exists x. \ x \in A \land P x$$

$$\text{balll}: \ (\land x. \ x \in A \Rightarrow P x) \Rightarrow \forall x \in A. \ P x$$

$$\text{bspec}: \ [\forall x \in A. \ P x; \ x \in A] \Rightarrow P x$$
Bounded Quantifiers

- \( \forall x \in A. \ P \ x \equiv \forall x. \ x \in A \rightarrow P \ x \)
- \( \exists x \in A. \ P \ x \equiv \exists x. \ x \in A \land P \ x \)
- \( \text{ballI}: (\bigwedge x. \ x \in A \implies P \ x) \implies \forall x \in A. \ P \ x \)
- \( \text{bspec}: \left[ \forall x \in A. \ P \ x; x \in A \right] \implies P \ x \)
- \( \text{bexI}: \left[ P \ x; x \in A \right] \implies \exists x \in A. \ P \ x \)
- \( \text{bexE}: \left[ \exists x \in A. \ P \ x; \bigwedge x. \left[ x \in A; P \ x \right] \implies Q \right] \implies Q \)
Demo

Sets
The Three Basic Ways of Introducing Theorems

→ Axioms:

Example: axiomatization where refl: ”$t = t$”

→ Definitions:

Example: definition inj where

 $\text{inj } f \equiv \forall x y. f x = f y \rightarrow x = y$

Introduces a new lemma called inj def.

→ Proofs:

Example: lemma ”$\lambda x. x + 1$”
The Three Basic Ways of Introducing Theorems

→ Axioms:

Example: axiomatization where refl: "t = t"

Do not use. Evil. Can make your logic inconsistent.
The Three Basic Ways of Introducing Theorems

→ Axioms:
  Example: axiomatization where refl: "t = t"
  Do not use. Evil. Can make your logic inconsistent.

→ Definitions:
  Example: definition inj where "inj f ≡ ∀x y. f x = f y → x = y"

---

9 | COMP4161 | CC-BY-4.0 License
The Three Basic Ways of Introducing Theorems

→ Axioms:
   Example: axiomatization where refl: "t = t"
   Do not use. Evil. Can make your logic inconsistent.

→ Definitions:
   Example: definition inj where "inj \( f \equiv \forall x \ y. \ f \ x = f \ y \longrightarrow x = y \)"
   Introduces a new lemma called inj_def.
The Three Basic Ways of Introducing Theorems

→ Axioms:
  Example: \( \text{axiomatization where refl: } t = t \)
  Do not use. Evil. Can make your logic inconsistent.

→ Definitions:
  Example: \( \text{definition inj where } \forall x y. f x = f y \rightarrow x = y \)
  Introduces a new lemma called inj_def.

→ Proofs:
  Example: \( \text{lemma } \forall x. x + 1 \)
The Three Basic Ways of Introducing Theorems

→ Axioms:
  Example: axiomatization where refl: "t = t"
  Do not use. Evil. Can make your logic inconsistent.

→ Definitions:
  Example: definition inj where "inj f ≡ ∀x y. f x = f y → x = y"
  Introduces a new lemma called inj_def.

→ Proofs:
  Example: lemma "inj (λx. x + 1)"
  The harder, but safe choice.
The Three Basic Ways of Introducing Types

→ **typedecl**: by name only

Example:  
`typedecl names`  
Introduces new type *names* without any further assumptions
The Three Basic Ways of Introducing Types

→ **typedec**: by name only
  Example: `typedec names`
  Introduces new type *names* without any further assumptions

→ **type_synonym**: by abbreviation
  Example: `type_synonym α rel = "α ⇒ α ⇒ bool"`
  Introduces abbreviation *rel* for existing type *α ⇒ α ⇒ bool*
  Type abbreviations are immediately expanded internally
The Three Basic Ways of Introducing Types

- **typedef**: by name only
  
  Example: `typedef names`
  Introduces new type `names` without any further assumptions

- **type synonym**: by abbreviation
  
  Example: `type synonym α rel = "α ⇒ α ⇒ bool"`
  Introduces abbreviation `rel` for existing type `α ⇒ α ⇒ bool`
  Type abbreviations are immediately expanded internally

- **typedef**: by definition as a set
  
  Example: `typedef new_type = "{some set}" <proof>`
  Introduces a new type as a subset of an existing type.
  The proof shows that the set on the rhs in non-empty.
How typedef works

new type
How typedef works

new type

existing type
How typedef works

new type

existing type
How typedef works

new type

existing type

Rep

Abs
How typedef works

new type

existing type

Rep

Abs
Example: Pairs

\[(\alpha, \beta) \text{ Prod}\]

① Pick existing type:
Example: Pairs

\[(\alpha, \beta) \text{ Prod}\]

1. Pick existing type: \(\alpha \Rightarrow \beta \Rightarrow \text{bool}\)
2. Identify subset:

We get from Isabelle:

- functions Abs\_Prod, Rep\_Prod
- both injective

We now can:

- define constants Pair, fst, snd in terms of Abs\_Prod and Rep\_Prod
- derive all characteristic theorems
- forget about Rep/Abs, use characteristic theorems instead
Example: Pairs

\[(\alpha, \beta) \text{ Prod}\]

1. Pick existing type: \(\alpha \Rightarrow \beta \Rightarrow \text{bool}\)
2. Identify subset:
   \[(\alpha, \beta) \text{ Prod} = \{f. \exists a b. f = \lambda(x :: \alpha) (y :: \beta). x = a \land y = b\}\]
3. We get from Isabelle:

13 | COMP4161 | CC-BY-4.0 License
Example: Pairs

$((\alpha, \beta)) \text{ Prod}$

1. Pick existing type: $\alpha \Rightarrow \beta \Rightarrow \text{bool}$
2. Identify subset:
   $((\alpha, \beta)) \text{ Prod} = \{f. \exists a b. f = \lambda (x :: \alpha) (y :: \beta). x = a \land y = b\}$
3. We get from Isabelle:
   - functions Abs_Prod, Rep_Prod
   - both injective
   - $\text{Abs}_\text{Prod} (\text{Rep}_\text{Prod} x) = x$
4. We now can:
Example: Pairs

$$(\alpha, \beta) \text{ Prod}$$

1. Pick existing type: $\alpha \Rightarrow \beta \Rightarrow \text{bool}$
2. Identify subset:
   
   $$(\alpha, \beta) \text{ Prod} = \{ f. \exists a b. f = \lambda(x :: \alpha) (y :: \beta). x = a \land y = b\}$$
3. We get from Isabelle:
   - functions Abs_Prod, Rep_Prod
   - both injective
   - $\text{Abs\textunderscore Prod}(\text{Rep\textunderscore Prod} \ x) = x$
4. We now can:
   - define constants Pair, fst, snd in terms of Abs_Prod and Rep_Prod
   - derive all characteristic theorems
   - forget about Rep/Abs, use characteristic theorems instead
Demo

Introducing new Types
Inductive Definitions
Example

\[
\langle \text{skip}, \sigma \rangle \rightarrow \sigma \quad \langle x := e, \sigma \rangle \rightarrow \sigma[x \mapsto v] \\
\langle c_1, \sigma \rangle \rightarrow \sigma' \quad \langle c_2, \sigma' \rangle \rightarrow \sigma'' \\
\langle c_1; c_2, \sigma \rangle \rightarrow \sigma''
\]

\[
\langle \text{while } b\text{ do } c, \sigma \rangle \rightarrow \sigma \\
\langle b \rangle \sigma = \text{False}
\]

\[
\langle c, \sigma \rangle \rightarrow \sigma' \quad \langle \text{while } b\text{ do } c, \sigma' \rangle \rightarrow \sigma''
\]

\[
\langle \text{while } b\text{ do } c, \sigma \rangle \rightarrow \sigma'' \\
\langle b \rangle \sigma = \text{True}
\]
What does this mean?

\[\langle c, \sigma \rangle \rightarrow \sigma'\]

Fancy syntax for a relation \((c, \sigma, \sigma') \in E\)

Relations are sets:

\[E ::= (com \times \text{state} \times \text{state}) \text{ set}\]

The rules define a set inductively.
What does this mean?

\[ \langle c, \sigma \rangle \rightarrow \sigma' \quad \text{fancy syntax for a relation} \quad (c, \sigma, \sigma') \in E \]
What does this mean?

$\langle c, \sigma \rangle \rightarrow \sigma'$ fancy syntax for a relation $(c, \sigma, \sigma') \in E$

relations are sets: $E ::= (\text{com} \times \text{state} \times \text{state})$ set
What does this mean?

→ \langle c, \sigma \rangle \longrightarrow \sigma' \quad \text{fancy syntax for a relation} \quad (c, \sigma, \sigma') \in E
→ \text{relations are sets: } E :: (\text{com} \times \text{state} \times \text{state}) \text{ set}
→ \text{the rules define a set inductively}
What does this mean?

→ \langle c, \sigma \rangle \longrightarrow \sigma' \quad \text{fancy syntax for a relation} \quad (c, \sigma, \sigma') \in E
→ \text{relations are sets: } E :: (\text{com } \times \text{ state } \times \text{ state}) \text{ set}
→ \text{the rules define a set inductively}

But which set?
Simpler Example

\[
\begin{align*}
0 \in N & \quad n \in N \\
\hline
n + 1 \in N &
\end{align*}
\]

\(N\) is the set of natural numbers.

But why not the set of real numbers?

\(0 \in \mathbb{R}, n \in \mathbb{R} \implies n + 1 \in \mathbb{R}\)

\(\mathbb{N}\) is the smallest set that is consistent with the rules.

Why the smallest set?

Objective: no junk. Only what must be in \(X\) shall be in \(X\).

Gives rise to a nice proof principle (rule induction).

Alternative (greatest set) occasionally also useful: coinduction.
Simpler Example

\[
\begin{align*}
0 & \in \mathbb{N} \\
n & \in \mathbb{N} \quad \Rightarrow \\
n + 1 & \in \mathbb{N}
\end{align*}
\]

\(\Rightarrow\) \(\mathbb{N}\) is the set of natural numbers \(\mathbb{N}\)
Simpler Example

\[
\begin{align*}
0 & \in \mathbb{N} & n & \in \mathbb{N} \\
\implies n + 1 & \in \mathbb{N}
\end{align*}
\]

→ \( \mathbb{N} \) is the set of natural numbers \( \mathbb{N} \)

→ But why not the set of real numbers? \( 0 \in \mathbb{R}, n \in \mathbb{R} \implies n + 1 \in \mathbb{R} \)
Simpler Example

\[ 0 \in \mathbb{N} \quad n \in \mathbb{N} \quad n + 1 \in \mathbb{N} \]

\( \rightarrow \) \( \mathbb{N} \) is the set of natural numbers \( \mathbb{N} \)

\( \rightarrow \) But why not the set of real numbers? \( 0 \in \mathbb{R}, \ n \in \mathbb{R} \implies n + 1 \in \mathbb{R} \)

\( \rightarrow \) \( \mathbb{N} \) is the smallest set that is consistent with the rules.
Simpler Example

\[ \begin{align*}
0 \in \mathbb{N} & & n \in \mathbb{N} \\
n + 1 \in \mathbb{N} & & n + 1 \in \mathbb{N}
\end{align*} \]

→ \( \mathbb{N} \) is the set of natural numbers \( \mathbb{N} \)

→ But why not the set of real numbers? \( 0 \in \mathbb{R}, n \in \mathbb{R} \implies n + 1 \in \mathbb{R} \)

→ \( \mathbb{N} \) is the **smallest** set that is **consistent** with the rules.

**Why the smallest set?**
Simpler Example

\[
\begin{align*}
0 \in N & \quad \quad \quad n \in N \\
n + 1 \in N &
\end{align*}
\]

$\rightarrow$ $N$ is the set of natural numbers $\mathbb{N}$

$\rightarrow$ But why not the set of real numbers? $0 \in \mathbb{R}$, $n \in \mathbb{R} \implies n + 1 \in \mathbb{R}$

$\rightarrow$ $\mathbb{N}$ is the \textbf{smallest} set that is \textbf{consistent} with the rules.

Why the smallest set?

$\rightarrow$ Objective: \textbf{no junk}. Only what must be in $X$ shall be in $X$. 
Simpler Example

\[
\begin{align*}
0 & \in \mathbb{N} \\
n & \in \mathbb{N} \\
n + 1 & \in \mathbb{N}
\end{align*}
\]

\[\rightarrow \] \( \mathbb{N} \) is the set of natural numbers \( \mathbb{N} \)

\[\rightarrow \] But why not the set of real numbers? \( 0 \in \mathbb{R}, n \in \mathbb{R} \implies n + 1 \in \mathbb{R} \)

\[\rightarrow \] \( \mathbb{N} \) is the smallest set that is consistent with the rules.

Why the smallest set?

\[\rightarrow \] Objective: no junk. Only what must be in \( X \) shall be in \( X \).

\[\rightarrow \] Gives rise to a nice proof principle (rule induction)
Simpler Example

\[
\begin{align*}
0 & \in \mathbb{N} \\
n & \in \mathbb{N} \\
n + 1 & \in \mathbb{N}
\end{align*}
\]

- \( \mathbb{N} \) is the set of natural numbers \( \mathbb{N} \)
- But why not the set of real numbers? \( 0 \in \mathbb{R}, n \in \mathbb{R} \implies n + 1 \in \mathbb{R} \)
- \( \mathbb{N} \) is the \textbf{smallest} set that is \textbf{consistent} with the rules.

Why the smallest set?

- Objective: \textbf{no junk}. Only what must be in \( X \) shall be in \( X \).
- Gives rise to a nice proof principle (rule induction)
- Alternative (greatest set) occasionally also useful: coinduction
Rule Induction

\[\frac{0 \in \mathbb{N}}{n \in \mathbb{N}} \quad \frac{n \in \mathbb{N}}{n + 1 \in \mathbb{N}}\]

induces induction principle

\[\left[ P \ 0; \ \forall n. \ P \ n \implies P \ (n + 1) \right] \implies \forall x \in \mathbb{N}. \ P \ x\]
Demo

Inductive Definitions
We have learned today ...

→ Sets
We have learned today ...

- Sets
- Type Definitions
We have learned today ...

- Sets
- Type Definitions
- Inductive Definitions