Interactive Formal Verification
9: Presenting Theories

Tjark Weber
Computer Laboratory
University of Cambridge
Concrete Syntax

• Suggestive textual representation is important:
  • $x+y$ instead of `plus x y`
  • `[1,2,3]` instead of `Cons 1 (Cons 2 (Cons 3 Nil))`
  • `'a × 'b` instead of `('a, 'b) prod`
Concrete Syntax

- Suggestive textual representation is important:
  - \( x+y \) instead of `plus x y`
  - \([1,2,3]\) instead of `Cons 1 (Cons 2 (Cons 3 Nil))`
  - `'a × 'b` instead of `('a, 'b) prod`

- Isabelle uses `mixfix annotations`. 
Concrete Syntax

- Suggestive textual representation is important:
  - \( x+y \) instead of \( \text{plus } x \ y \)
  - \([1,2,3]\) instead of \(\text{Cons 1 (Cons 2 (Cons 3 Nil))}\)
  - \('a \times 'b\) instead of \(('a, 'b) \text{ prod}\)

- Isabelle uses \textit{mixfix annotations}. 

\[\langle _, _\rangle \rightarrow _{\text{\_}}\]
Concrete Syntax

● Suggestive textual representation is important:
  ● x+y instead of plus x y
  ● [1,2,3] instead of Cons 1 (Cons 2 (Cons 3 Nil))
  ● 'a × 'b instead of ('a, 'b) prod

● Isabelle uses mixfix annotations.

● These can be specified wherever constants (or types) are defined: definition, fun, datatype, ...
Infix Annotations

definition xor :: "bool ⇒ bool ⇒ bool"
  (infixl "[+]" 60)
where "A [+] B ≡ (A ∧ ¬B) ∨ (¬A ∧ B)"
Infix Annotations

definition xor :: "bool ⇒ bool ⇒ bool"
  (infixl "[+]" 60)
where "A [+] B ≡ (A ∧ ¬B) ∨ (¬A ∧ B)"
Infix Annotations

Infix annotation

**definition** xor :: "bool ⇒ bool ⇒ bool"

(infixl "[++]") 60

where "A [+][+] B ≡ (A ∧ ¬B) ∨ (¬A ∧ B)"

Infix Annotations

definition xor :: "bool ⇒ bool ⇒ bool"
  (infixl "[+]" 60)
where "A [+] B ≡ (A ∧ ¬B) ∨ (¬A ∧ B)"

Left-associative: A [+] B [+] C = (A [+] B) [+] C

Other options: infixr, infix
Infix Annotations

definition xor :: "bool ⇒ bool ⇒ bool"
  (infixl "[+]" 60)
where "A [+ B = (A ∧ ¬B) ∨ (¬A ∧ B)"

Other options: infixr, infix

Left-associative: A [+ B [+ C = (A [+ B) [+ C

Infix annotation

Precedence
Infix Annotations

**definition xor :: "bool ⇒ bool ⇒ bool"
(infixl "[+]" 60)
where "A [+] B ≡ (A ∧ ¬B) ∨ (¬A ∧ B)"

Now xor A B and A [+] B mean the same.

Other options: infixr, infix

Left-associative: A [+] B [+] C = (A [+] B) [+] C
Infix Annotations

**Definition**

\[
xor :: 
\begin{align*}
\text{bool} \Rightarrow \text{bool} \Rightarrow \text{bool}
\end{align*}
\]

\[
(\text{infixl} \ "[++]\" \ 60)
\]

where

\[
A ++ B \equiv (A \land \neg B) \lor (\neg A \land B)
\]

- Now \( xor \ A \ B \) and \( A ++ B \) mean the same.
- Use \( \text{op} \) for partial application: \( \text{op} \ [++] \)

**Infix annotation**

**Precedence**

**Left-associative**:

\[
A ++ B ++ C = (A ++ B) ++ C
\]
Mathematical Symbols

• Isabelle has its own notion of symbols:
  • 7-bit ASCII characters
  • Named symbols: \(<ident>\)
  • Named control symbols: \(\langle^ident\rangle\)
Mathematical Symbols

- Isabelle has its own notion of symbols:
  - 7-bit ASCII characters
  - Named symbols: \(<\text{ident}>\)
  - Named control symbols: \(<^\text{ident}>\)
- Symbol / ASCII / internal name:
  e.g., ∀ / ALL, ! / \(<\text{forall}>\)
Mathematical Symbols

- Isabelle has its own notion of symbols:
  - 7-bit ASCII characters
  - Named symbols: `\<ident>`
  - Named control symbols: `\<^ident>`
- Symbol / ASCII / internal name:
  e.g., `∀` / `ALL`, `!` / `\forall`
Mathematical Symbols

• Isabelle has its own notion of symbols:
  • 7-bit ASCII characters
  • Named symbols: \<ident>
  • Named control symbols: \<^ident>

• Symbol / ASCII / internal name:
  e.g., \( \forall \) / ALL, \( ! \) / \<forall>

• Unicode (UTF-8)

See the Tutorial for other symbols
Mathematical Symbols

- Isabelle has its own notion of symbols:
  - 7-bit ASCII characters
  - Named symbols: \(<ident>\)
  - Named control symbols: \(<^ident>\)

- Symbol / ASCII / internal name:
  e.g., \(\forall / \text{ALL}, \neg / \text{forall}\)

- Unicode (UTF-8)

See the Tutorial for other symbols

Beware
Using Symbols

definition xor :: "bool ⇒ bool ⇒ bool"
  (infixl "⊕" 60)
where "A ⊕ B ≡ (A ∧ ¬B) ∨ (¬A ∧ B)"
Using Symbols

definition xor :: "bool ⇒ bool ⇒ bool"
  (infixl "⊕" 60)
where "A ⊕ B ≡ (A ∧ ¬B) ∨ (¬A ∧ B)"

\<oplus>
Using Symbols

definition xor :: "bool ⇒ bool ⇒ bool"
  (infixl "⊕" 60)
where "A ⊕ B ≡ (A ∧ ¬B) ∨ (¬A ∧ B)"

• Now xor A B and A ⊕ B mean the same.
Declaring Alternative Notation

definition xor :: "bool ⇒ bool ⇒ bool"
  (infixl "[+]") 60
where "A [+] B ≡ (A ∧ ¬B) ∨ (¬A ∧ B)"
Declaring Alternative Notation

definition xor :: "bool ⇒ bool ⇒ bool"
  (infixl "[+]" 60)
where "A [+] B ≡ (A ∧ ¬B) ∨ (¬A ∧ B)"

notation xor (infixl "⊕" 60)
Declaring Alternative Notation

definition xor :: "bool ⇒ bool ⇒ bool"
  (infixl "[+]" 60)
where "A [+] B ≡ (A ∧ ¬B) ∨ (¬A ∧ B)"

notation xor (infixl "⊕" 60)

Associates a mixfix annotation with a known constant
Declaring Alternative Notation

definition xor :: "bool ⇒ bool ⇒ bool"
    (infixl "[+]" 60)
where "A [+] B ≡ (A ∧ ¬B) ∨ (¬A ∧ B)"

 Associates a mixfix annotation with a known constant

notation xor (infixl "⊕" 60)

Now ⊕ also means xor.
Declaring Alternative Notation

definition xor :: "bool ⇒ bool ⇒ bool"
   (infixl "[+]" 60)
where "A [+ ] B ≡ (A ∧ ¬B) ∨ (¬A ∧ B)"

notation xor (infixl "⊕" 60)

Associates a mixfix annotation with a known constant

Now ⊕ also means xor.

Optional: a print mode
Prefix Annotations

• A simple form of mixfix annotations
Prefix Annotations

- A simple form of mixfix annotations
- No template arguments, no priorities
Prefix Annotations

- A simple form of mixfix annotations
- No template arguments, no priorities

```
datatype currency =
    Euro nat ("€")
  | Pounds nat ("£")
```
Prefix Annotations

• A simple form of mixfix annotations
• No template arguments, no priorities

```datatype currency =
  Euro nat ("€")
| Pounds nat ("£")
```
Prefix Annotations

- A simple form of mixfix annotations
- No template arguments, no priorities

```plaintext
datatype currency =
  Euro nat ("€")
| Pounds nat ("£")
```

Now € 10 means Euro 10.
Abbreviations

• Even more powerful than mixfix annotations
Abbreviations

• Even more powerful than mixfix annotations

abbreviation inS :: "'a ⇒ 'a ⇒ bool"
    (infix "≈" 50)
where "x ≈ y ≡ (x,y) ∈ S"
Abbreviations

• Even more powerful than mixfix annotations

Introduces a new constant as an abbreviation for a complex term

abbreviation inS :: "'a ⇒ 'a ⇒ bool"
  (infix "≈" 50)
where "x ≈ y ≡ (x,y) ∈ S"
Abbreviations

- Even more powerful than mixfix annotations

   ```haskell
   abbreviation inS :: "'a ⇒ 'a ⇒ bool"
   (infix "≈" 50)
   where "x ≈ y ≡ (x,y) ∈ S"
   ```

- Automatically folded/unfolded
Abbreviations

• Even more powerful than mixfix annotations

Introduces a new constant as an abbreviation for a complex term

abbreviation inS :: "'a ⇒ 'a ⇒ bool"
   (infix "≈" 50)
where "x ≈ y ≡ (x,y) ∈ S"

• Automatically folded/unfolded

Abbreviations do not replace definitions!
Document Preparation

- Two ways to turn Isabelle theories into PDF documents:
  - Isabelle > Commands > Display Draft
  - Document preparation via LaTeX
Document Preparation

- Two ways to turn Isabelle theories into PDF documents:
  - Isabelle > Commands > Display Draft
    - Prints the raw theory sources
  - Document preparation via LaTeX
Document Preparation

- Two ways to turn Isabelle theories into PDF documents:
  - Isabelle > Commands > Display Draft
    - Prints the raw theory sources
  - Document preparation via LaTeX
    - Proper typesetting of mathematical symbols
Isabelle Sessions

- Document preparation works in batch mode.
Isabelle Sessions

• Document preparation works in batch mode.

- Isabelle Emacs is just one tool – there are many others.
Isabelle Sessions

• Document preparation works in batch mode.

  isabelle emacs is just one tool – there are many others.

• Create a session: isabelle mkdir MySession
Isabelle Sessions

• Document preparation works in batch mode.

  isabelle emacs is just one tool – there are many others.

• Create a session: `isabelle mkdir MySession`

• Run a session: `isabelle make`
Isabelle Sessions

- Document preparation works in batch mode.
  
isabelle emacs is just one tool – there are many others.

- Create a session: `isabelle mkdir MySession`
- Run a session: `isabelle make`

- Uses LaTeX to produce `document.pdf`
Session Sources

- `isabelle mkdir MySession` generates
  - `MySession/`
  - `MySession/ROOT.ML`
  - `MySession/document`
  - `IsaMakefile`
Session Sources

- `isabelle mkdir MySession` generates
  - MySession/
  - MySession/ROOT.ML
  - MySession/document
  - IsaMakefile

Put your theory files here
Session Sources

- `isabelle mkdir MySession` generates
  - `MySession/`
  - `MySession/ROOT.ML`
  - `MySession/document`
  - `IsaMakefile`

  - Put your theory files here
  - Loads your theories
Session Sources

- `isabelle mkdir MySession` generates
  - `MySession/`
    - Put your theory files here
  - `MySession/ROOT.ML`
    - Loads your theories
  - `MySession/document`
    - LaTeX stage
  - `IsaMakefile`
Session Sources

- `isabelle mkdir MySession` generates
  - `MySession/`
    - Put your theory files here
  - `MySession/ROOT.ML`
    - Loads your theories
  - `MySession/document`
    - LaTeX stage
  - `IsaMakefile`
    - Dependencies, session management
header { * Some properties of Foo *}

theory Foo imports Main
begin

subsection { * Basic definitions 
\label{sec:basic-defs} *}

definition foo :: ...

end
header {* Some properties of Foo *}

theory Foo imports Main begin

subsection {* Basic definitions \label{sec:basic-defs} *}

definition foo :: ...

definition foo :: ...

end
theory Foo imports Main
begin
subsection {* Basic definitions \label{sec:basic-defs} *}
definition foo :: ...
end
header {* Some properties of Foo *}

theory Foo imports Main
begin
subsection {* Basic definitions \label{sec:basic-defs} *}
definition foo :: ...
end

... with arbitrary LaTeX commands
Antiquotations

- Antiquotations refer to formal theory content from informal text blocks.

```text
{*
  @{term "%x y. x"} is a well-typed term.
*}
```
Antiquotations

- Antiquotations refer to formal theory content from informal text blocks.

```
@{term "%x y. x"} is a well-typed term.
```
Antiquotations

- Antiquotations refer to formal theory content from informal text blocks.

```
text {*
  @{term "\%x y. x"} is a well-typed term.
*
```

Antiquotation that prints a term
Antiquotations

- Antiquotations refer to formal theory content from informal text blocks.

\texttt{\{@\text{term "\%x y. x"}\} is a well-typed term.}\n
- Output: \( \lambda x\ y.\ x \) is a well-typed term.
Antiquotations

- Antiquotations refer to formal theory content from informal text blocks.

```
\text{\{\text{@\{term "\textbf{\%}x \ y. \ x"}\} is a well-typed term.}\}}
```

- Output: \(\lambda x \ y. \ x\) is a well-typed term.

Note that \(\%\) is printed as \(\lambda\)
Useful Antiquotations

- @{typ "τ"}
- @{term "t"}
- @{const "c"}
- @{prop "ϕ"}
- @{thm name}
- @{text "s"}
Useful Antiquotations

- \( \text{@typ } \tau \text{ } \)
- \( \text{@term } t \text{ } \)
- \( \text{@const } c \text{ } \)
- \( \text{@prop } \phi \text{ } \)
- \( \text{@thm name} \text{ } \)
- \( \text{@text } s \text{ } \)

Prints a type
Useful Antiquotations

- @{typ "\tau"}
- @{term "t"}
- @{const "c"}
- @{prop "\phi"}
- @{thm name}
- @{text "s"}

Prints a type
Prints a term
Useful Antiquotations

- @{typ "τ"}
- @{term "t"}
- @{const "c"}
- @{prop "ϕ"}
- @{thm name}
- @{text "s"}
Useful Antiquotations

- \(@\{\text{typ} \ "\tau"\}\) Prints a type
- \(@\{\text{term} \ "t"\}\) Prints a term
- \(@\{\text{const} \ "c"\}\) Prints a constant
- \(@\{\text{prop} \ "\phi"\}\) Prints a proposition
- \(@\{\text{thm name}\}\)
- \(@\{\text{text} \ "s"\}\)
Useful Antiquotations

- `{typ "τ"}`: Prints a type
- `{term "t"}`: Prints a term
- `{const "c"}`: Prints a constant
- `{prop "φ"}`: Prints a proposition
- `{thm name}`: Prints a theorem
- `{text "s"}`
Useful Antiquotations

- @{typ "τ"}  Prints a type
- @{term "t"}  Prints a term
- @{const "c"}  Prints a constant
- @{prop "ϕ"}  Prints a proposition
- @{thm name}  Prints a theorem
- @{text "s"}  Prints uninterpreted text
Suppressing Output

- (* This is a comment. *)

- theory Foo (*<*)imports Main(*>*) begin
Suppressing Output

• (* This is a comment. *)

• `theory Foo (*<*>imports Main(*>*) begin`
Suppressing Output

- (* This is a comment. *)
  - Not processed, not printed

- `theory Foo (*<*>imports Main(*>*) begin`
  - Processed, but not printed