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

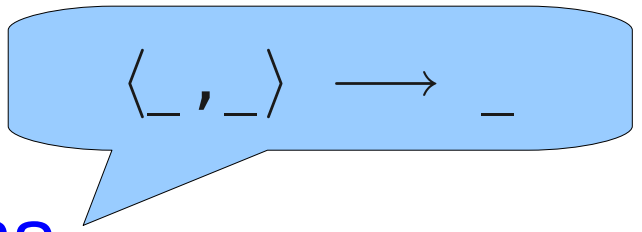
Concrete Syntax

- 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

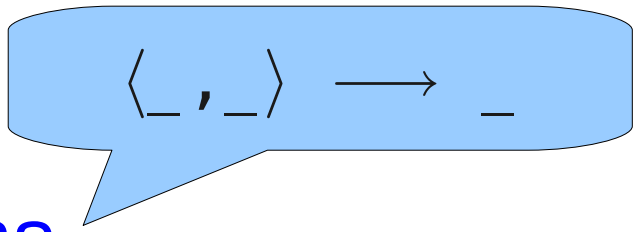
- Suggestive textual representation is important:
  - x+y instead of plus x y
  - [1,2,3] instead of Cons 1 (Cons 2 (Cons 3 Nil))
  - 'a × 'b instead of ('a, 'b) prod



- Isabelle uses mixfix annotations.

Concrete Syntax

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

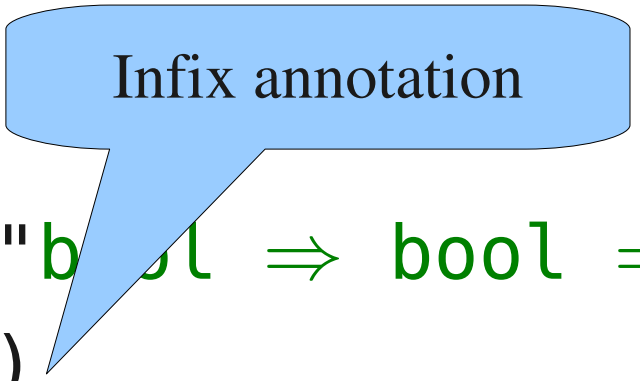


- Isabelle uses mixfix annotations.
- These can be specified wherever constants (or types) are defined: definition, fun, datatype, ...

Infix Annotations

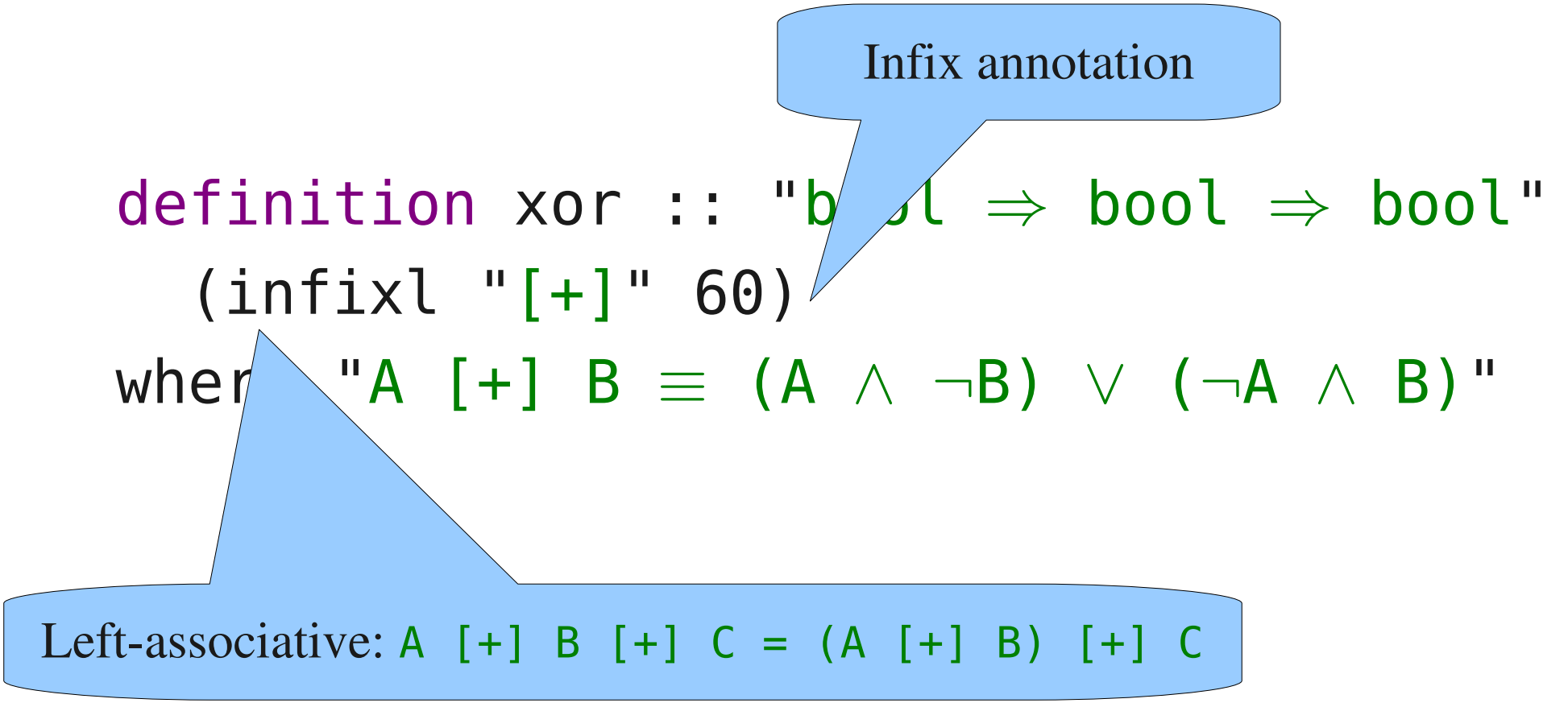
```
definition xor :: "bool ⇒ bool ⇒ bool"
  (infixl "[+]" 60)
where "A [+] B ≡ (A ∧ ¬B) ∨ (¬A ∧ B)"
```

Infix Annotations

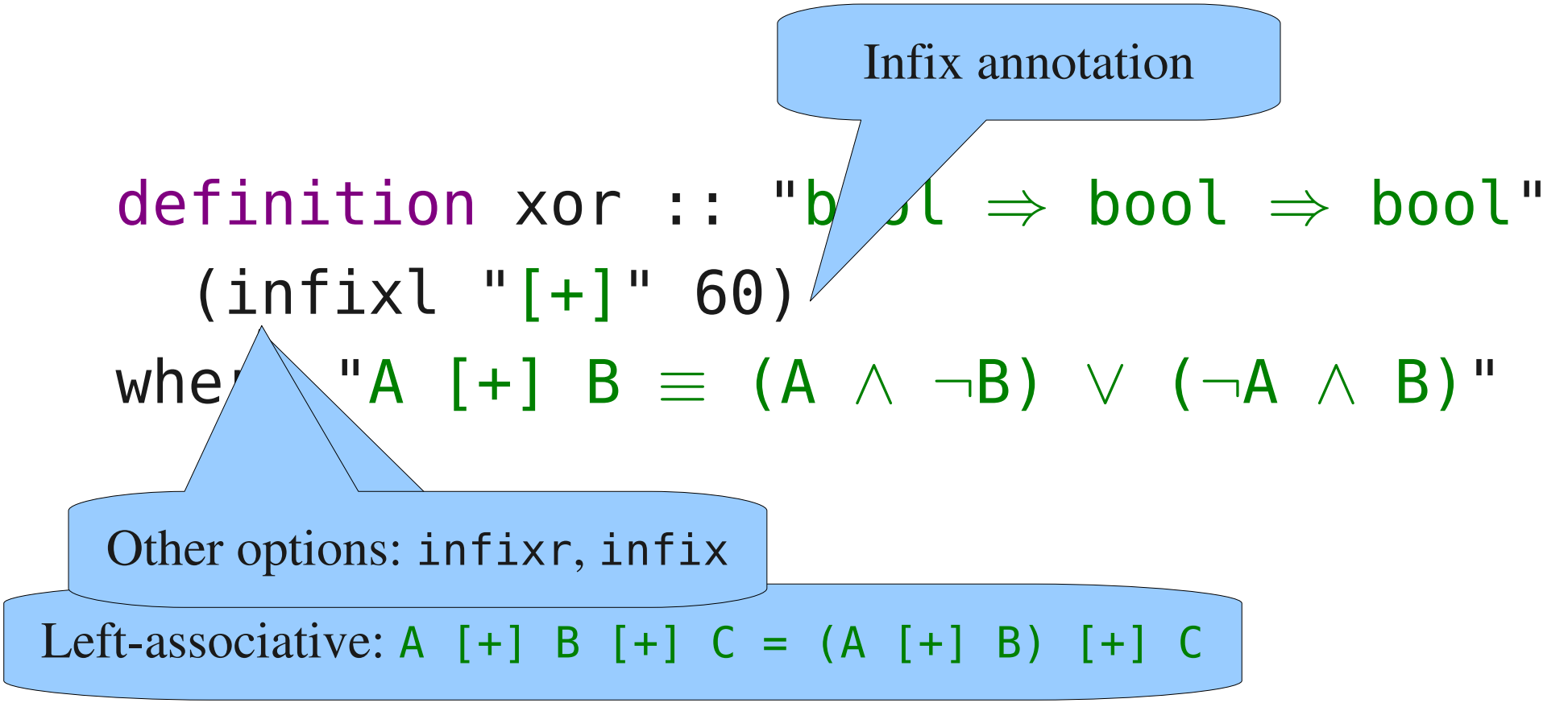



```
definition xor :: "bool ⇒ bool ⇒ bool"
  (infixl "[+]" 60)
where "A [+] B ≡ (A ∧ ¬B) ∨ (¬A ∧ B)"
```

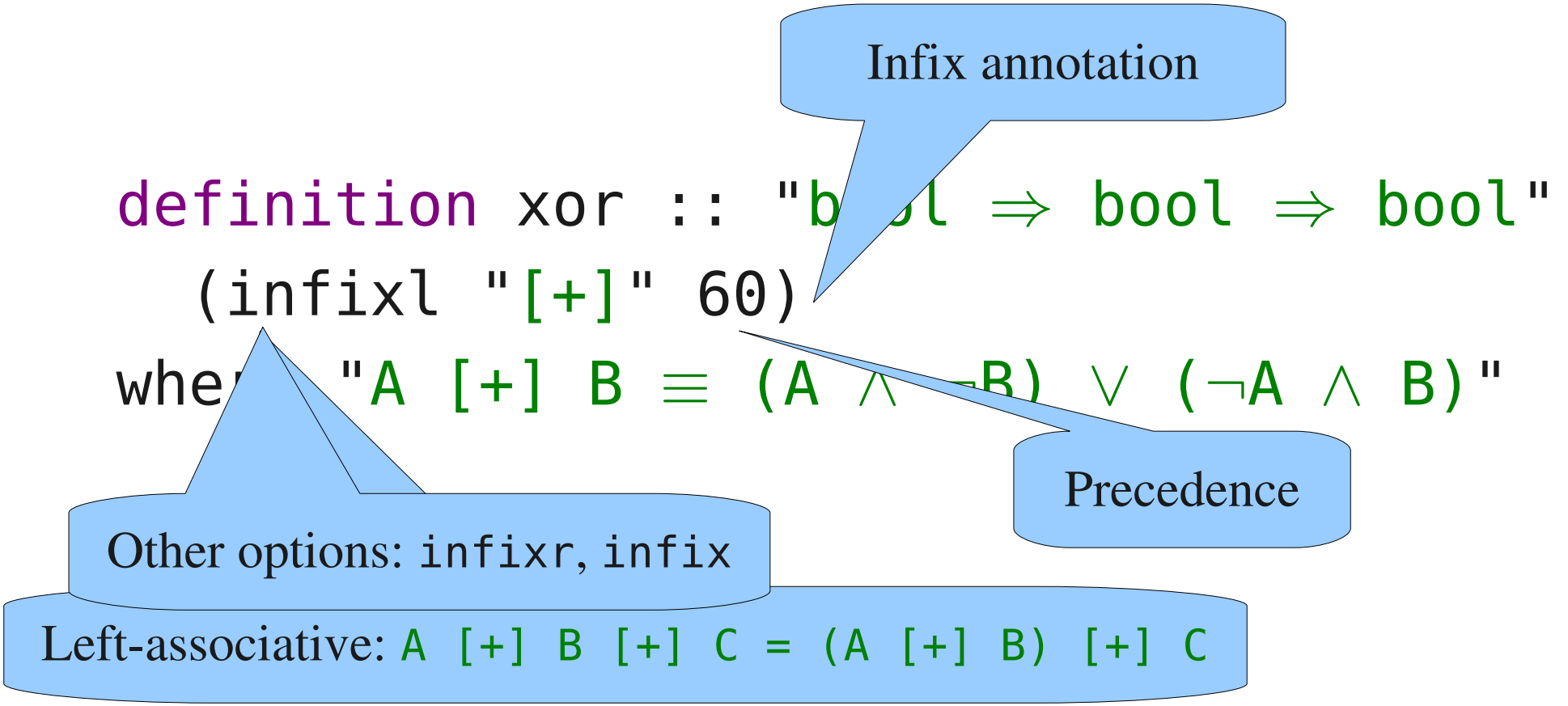
Infix Annotations



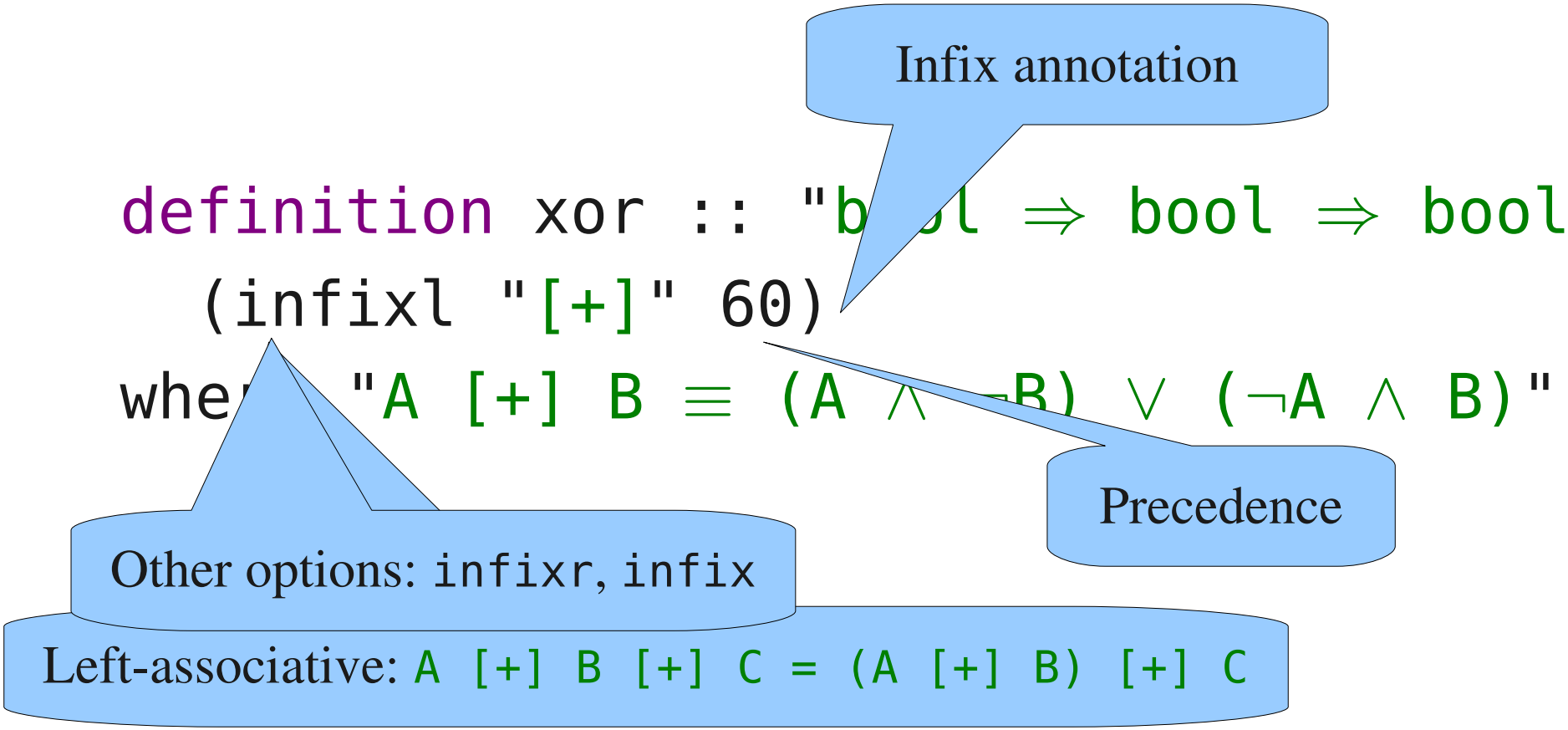
Infix Annotations



Infix Annotations

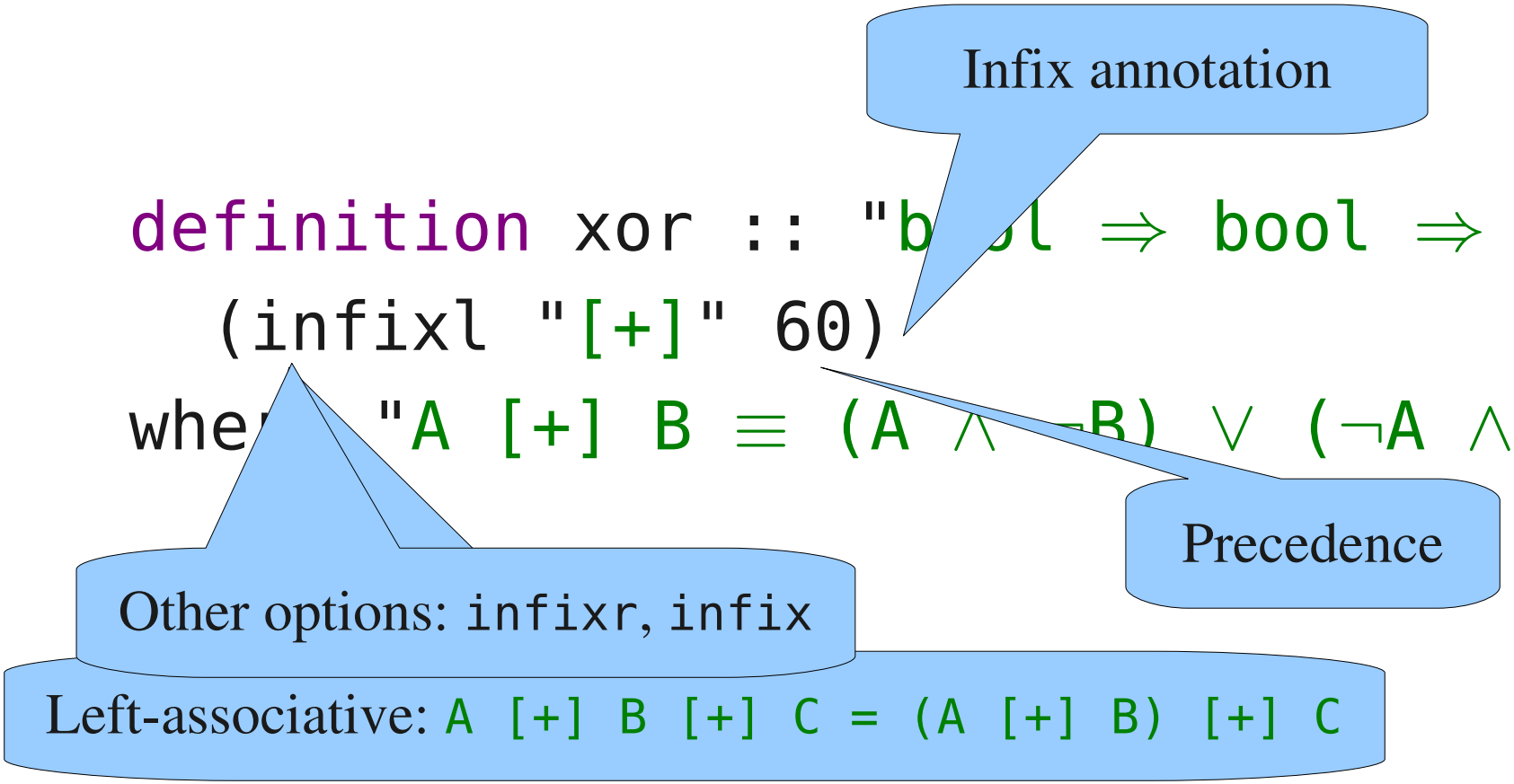


Infix Annotations



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

Infix Annotations



- Now `xor A B` and `A [+] B` mean the same.
- Use `op` for partial application: `op [+]`

Mathematical Symbols

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

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>`

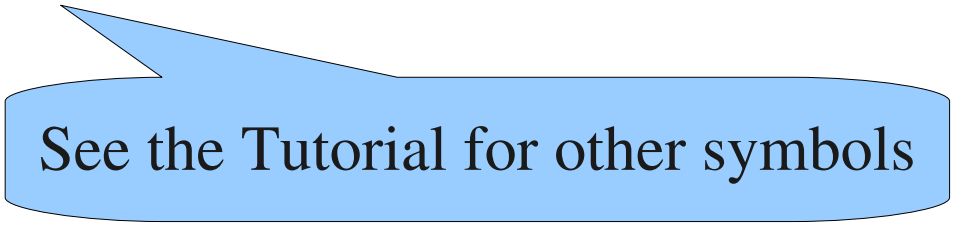
Mathematical Symbols

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

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., ∀ / `ALL`, `!` / `\<forall>`
- Unicode (UTF-8)



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>`
- Unicode (UTF-8)

Beware

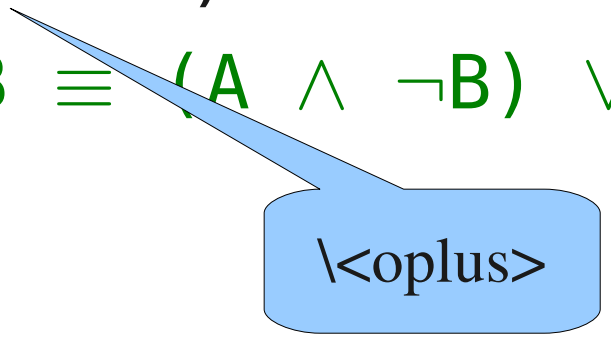
See the Tutorial for other symbols

Using Symbols

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

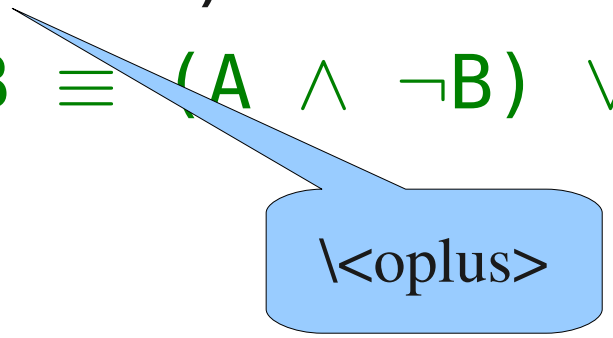
Using Symbols

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



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)"
```

Associates a mixfix annotation with a known constant

```
notation xor (infixl "⊕" 60)
```

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)"
```

Associates a mixfix annotation with a known constant

```
notation xor (infixl "⊕" 60)
```

Optional: a print mode

Now ⊕ also means `xor`.

Prefix Annotations

- A simple form of mixfix annotations

Prefix Annotations

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

Prefix Annotations

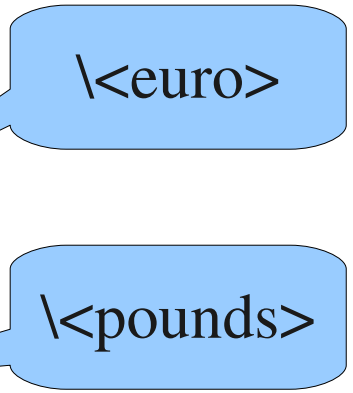
- A simple form of mixfix annotations
- No template arguments, no priorities

```
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

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

\<euro>

\<pounds>

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

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

- 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:

Prints the raw theory sources

  - Isabelle > Commands > Display Draft

  - Document preparation via LaTeX

Document Preparation

- Two ways to turn Isabelle theories into PDF documents:

Prints the raw theory sources

  - Isabelle > Commands > Display Draft

Proper typesetting of mathematical symbols

  - Document preparation via LaTeX

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`

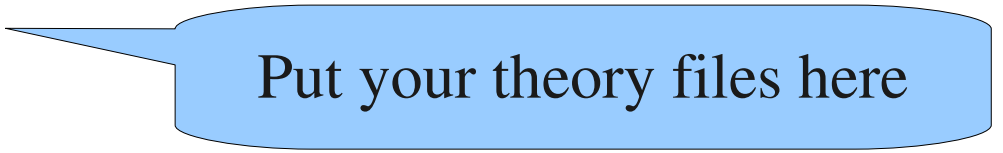
- Uses LaTeX to produce `document.pdf`

Session Sources

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

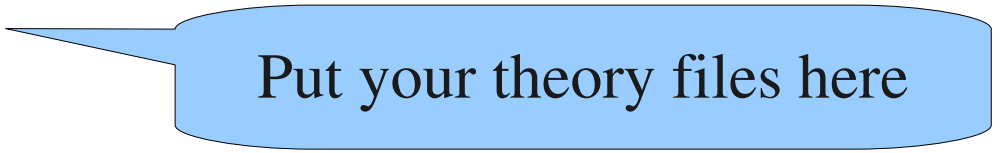
Session Sources

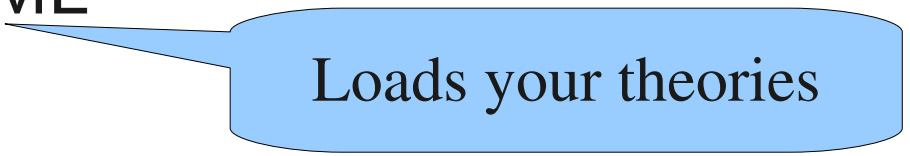
- `isabelle mkdir MySession` generates
  - MySession/
  - MySession/ROOT.ML
  - MySession/document
  - IsaMakefile



Session Sources

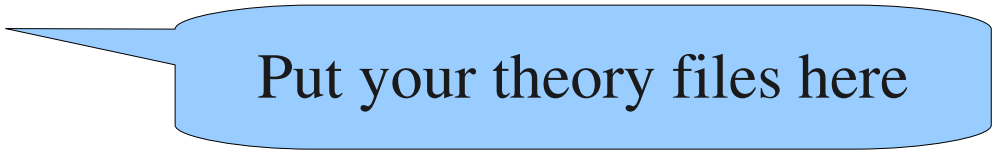
- `isabelle mkdir MySession` generates
  - MySession/
  - MySession/ROOT.ML
  - MySession/document
  - IsaMakefile

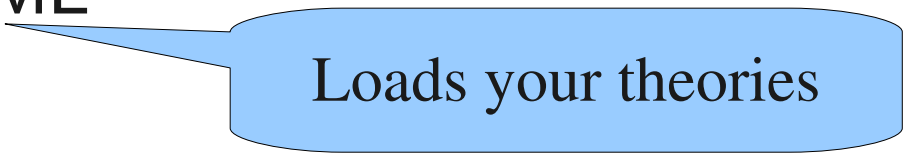


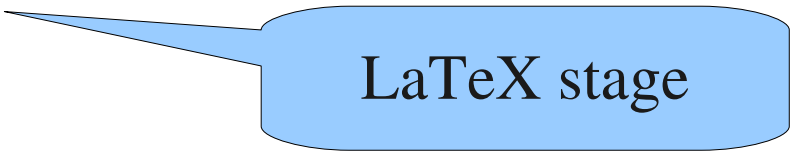


Session Sources

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

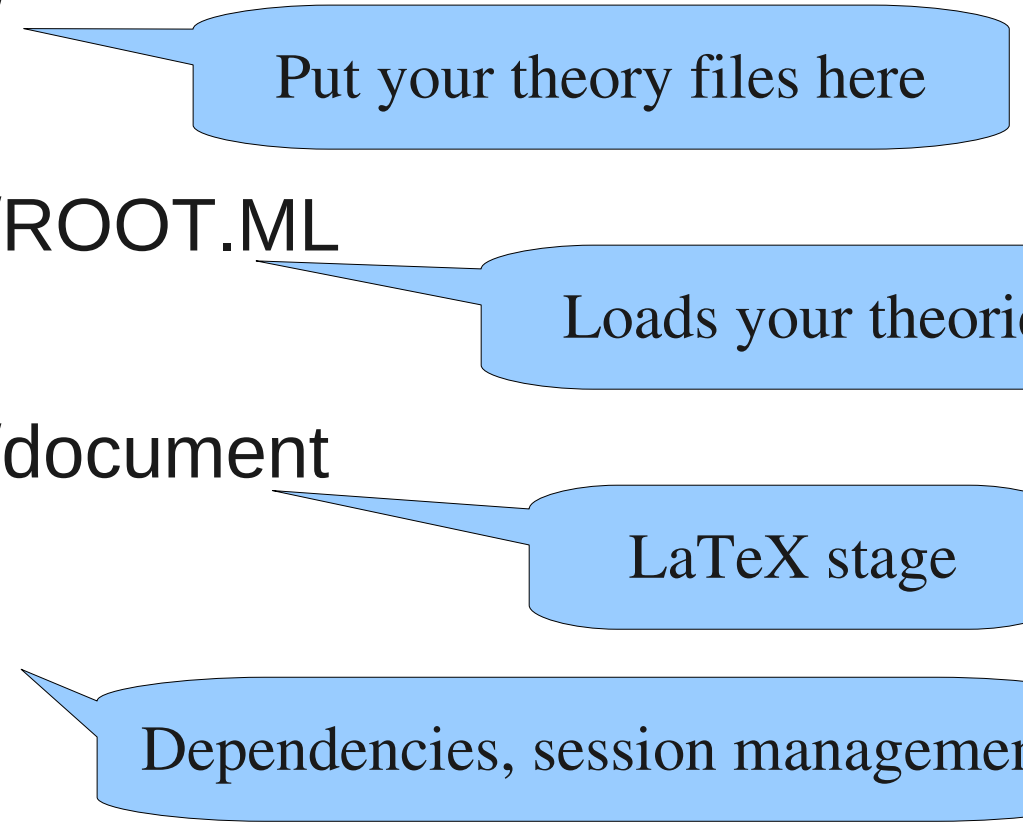






Session Sources

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



Structure Markup

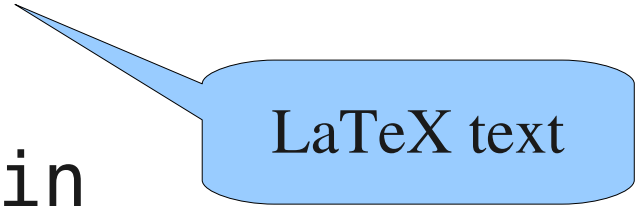
```
header {* Some properties of Foo *}

theory Foo imports Main
begin

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

definition foo :: ...

end
```

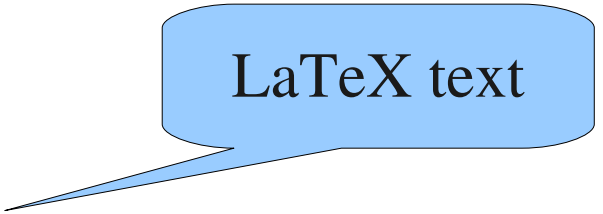
Structure Markup

```
header {* Some properties of Foo *}

theory Foo imports Main
begin

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

definition foo :: ...

end
```

Structure Markup

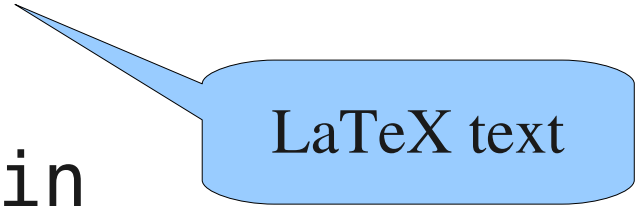
```
header {* Some properties of Foo *}

theory Foo imports Main
begin

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

definition foo :: ...

end
```

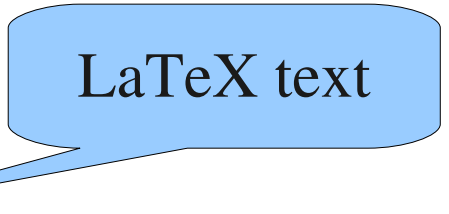


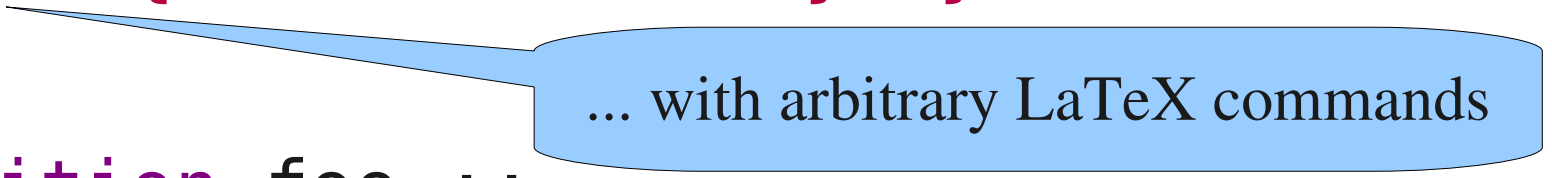


Structure Markup

```
header {* Some properties of Foo *}

theory Foo imports Main
begin

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

definition foo :: ...


end
```







Antiquotations

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

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

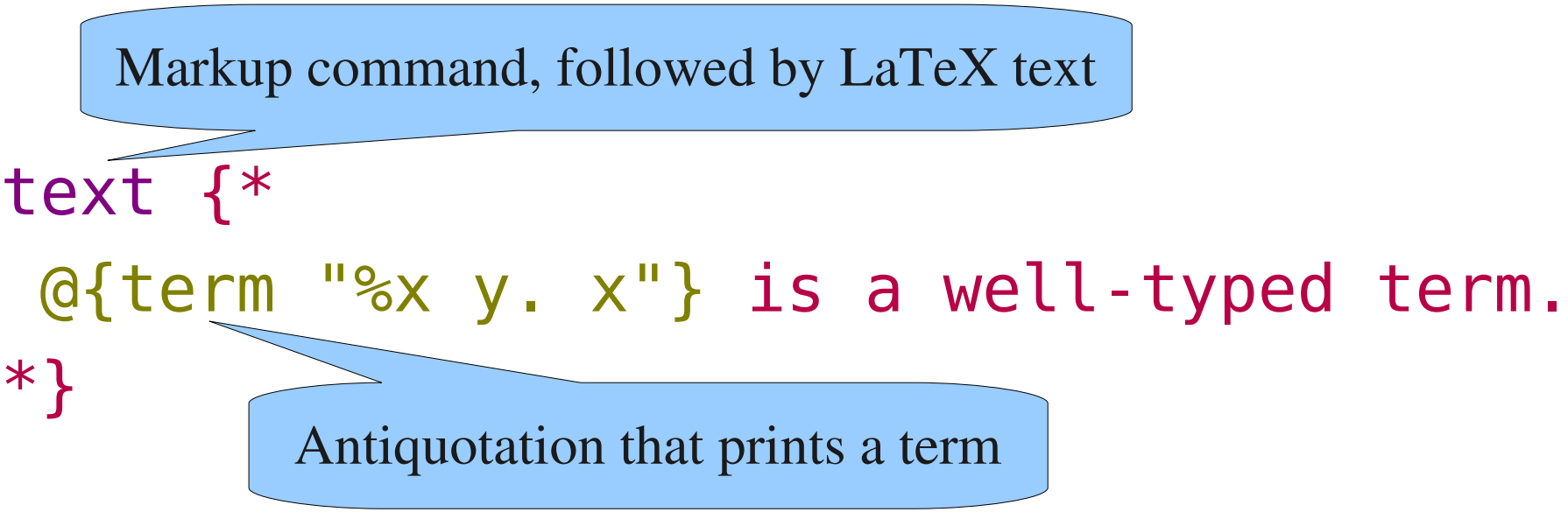
Antiquotations

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


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

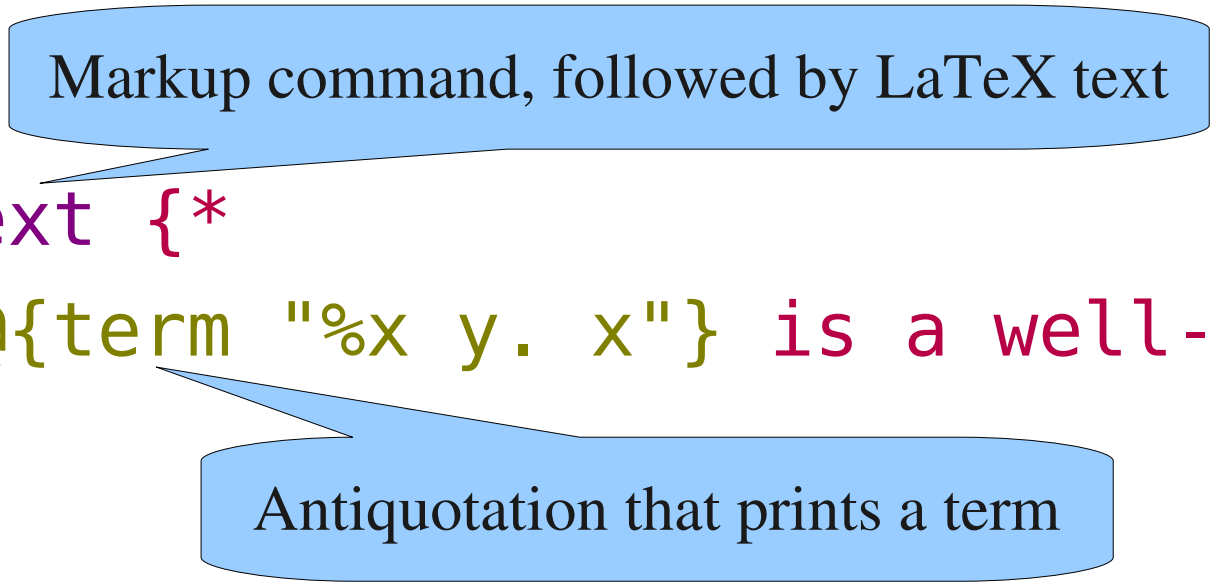
Antiquotations

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



Antiquotations

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

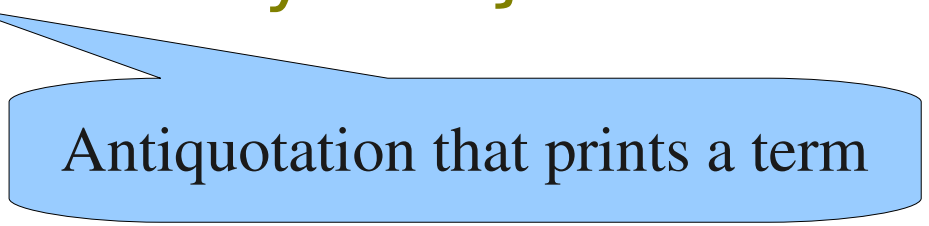


- Output: λx y. x is a well-typed term.

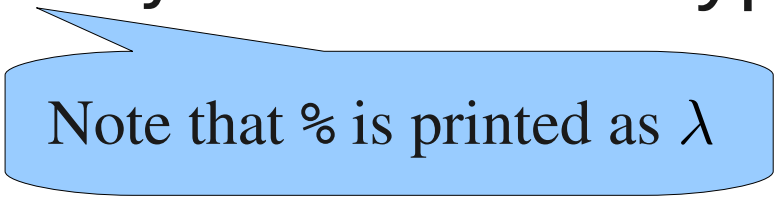
Antiquotations

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


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



- Output: λx y. x is a well-typed term.

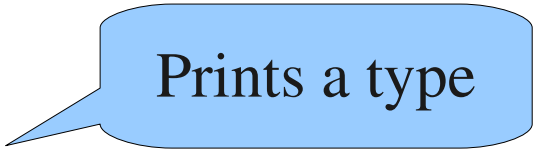


Useful Antiquotations

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

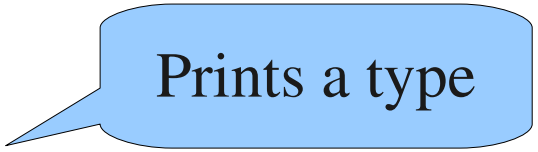
Useful Antiquotations

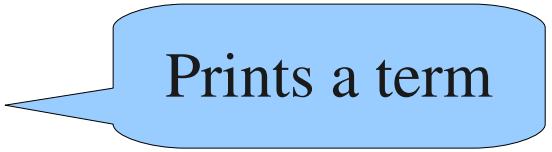
- `@{typ "τ"}`
- `@{term "t"}`
- `@{const "c"}`
- `@{prop "ϕ"}`
- `@{thm name}`
- `@{text "s"}`



Useful Antiquotations

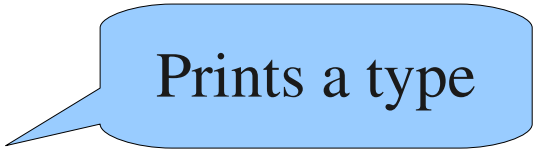
- `@{typ "τ"}`
- `@{term "t"}`
- `@{const "c"}`
- `@{prop "ϕ"}`
- `@{thm name}`
- `@{text "s"}`

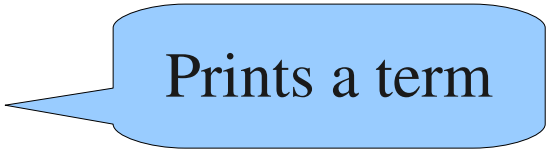


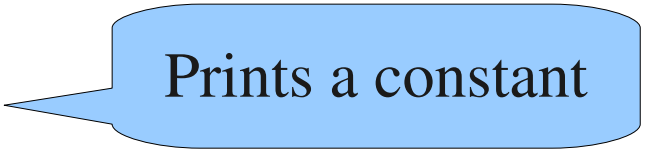


Useful Antiquotations

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

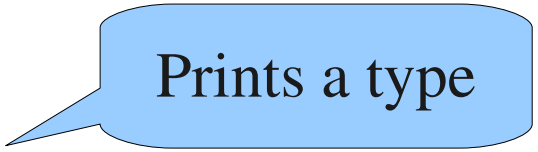


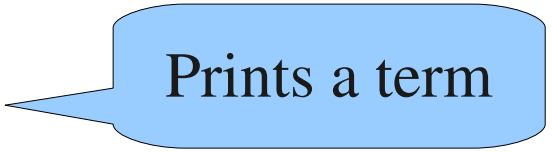


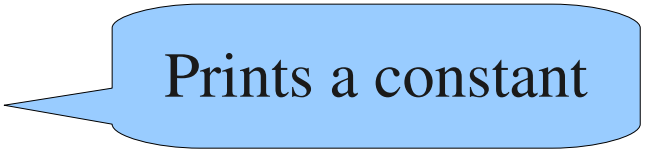


Useful Antiquotations

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

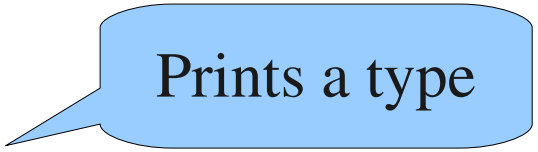


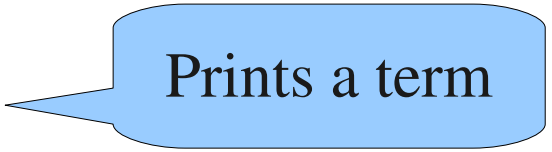


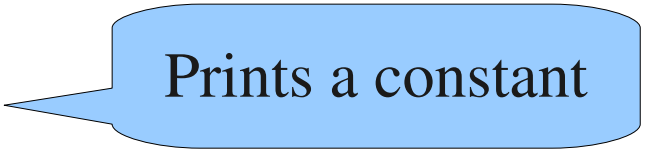


Useful Antiquotations

- `@{typ "τ"}`
- `@{term "t"}`
- `@{const "c"}`
- `@{prop "ϕ"}`
- `@{thm name}`
- `@{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

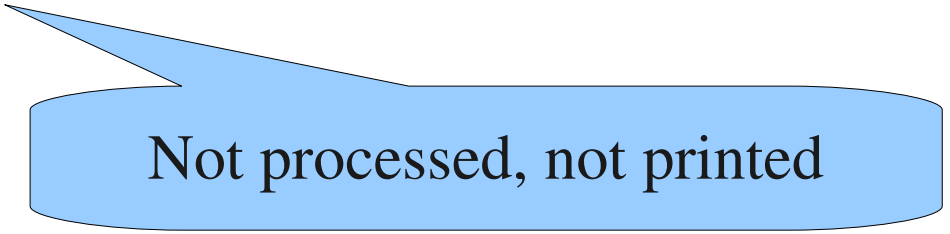
Suppressing Output

- `(* This is a comment. *)`

- `theory Foo (*<*)imports Main(*>*) begin`

Suppressing Output

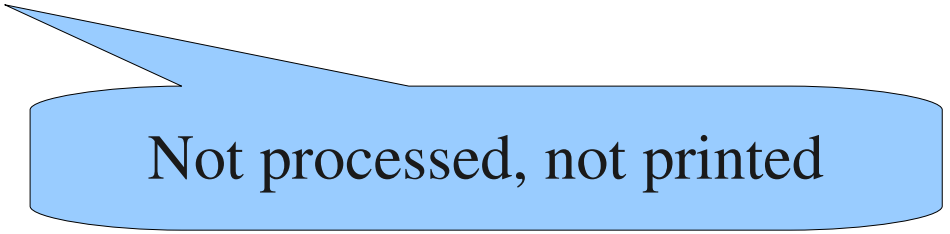
- `(* This is a comment. *)`



- `theory Foo (*<*)imports Main(*>*) begin`

Suppressing Output

- `(* This is a comment. *)`



- `theory Foo (*<*)imports Main(*>*) begin`