universität innsbruck

Summer Term 2023

Interactive Theorem Proving using Isabelle/HOL

Session 12

René Thiemann

Department of Computer Science

Outline

- Session Management
- Document Preparation
- Type Classes

Session Management

Isabelle Sessions

- session is "project" consisting of collection of theory files
- sessions can be defined hierarchically (import of other sessions); root of hierarchy: HOL
- sessions are defined in `ROOT` files
- processing sessions may take considerable time
- possible to capture state of sessions in persistent heap image/session image

Session Specifications

$$
\begin{array}{rcl}
\mathit{session} & ::= & \textbf{session}\ \mathit{name} = \mathit{name} + \mathit{description}^{?}\ \mathit{options}^{?}\ \mathit{sessions}^{?}\ \mathit{theories}^{*} \\
\mathit{description} & ::= & \textbf{description}\ \langle \mathit{text} \rangle \\
\mathit{options} & ::= & \textbf{options}\ [(\mathit{key}{=}\mathit{value} \mid \mathit{key})^{+}] \\
\mathit{sessions} & ::= & \textbf{sessions}\ \mathit{name}^{+} \\
\mathit{theories} & ::= & \textbf{theories}\ ([(\mathit{key}{=}\mathit{value} \mid \mathit{key})^{+}])^{?}\ \mathit{name}^{+}
\end{array}
$$

Example Session

```
session Test_Session = HOL +                    (* load and import HOL *)
  description ‹This is a test session›
  options [timeout = 600]                                (* 600 seconds *)
  sessions
    "HOL-Library"                                (* import HOL-Library *)
  theories
    Test        (* may import theories HOL.XXX and HOL-Library.XXX *)
```

Invoking the Build Process

use '`$ isabelle build [OPTIONS] S1 ... SN`' to run sessions `S1` to `SN` with `OPTIONS`:

`-d`	*dir*	search for `ROOT` files in *dir*
`-D`	*dir*	search for `ROOT` file in *dir* and select all its sessions
`-b`		build heap image
`-o`	*option*	override Isabelle option (syntax: *name*=*val* or *name*)
`-v`		be verbose
`-h`		show further build options (help)

Some Available Options

- `browser_info` – output HTML browser info (default: `false`)
- `document=pdf` – output PDF document
- `document_output=dir` – specify alternative directory `dir` for generated output
- `quick_and_dirty` – accept proof by `sorry`
- `names_short` – do not use qualified names in output
- `show_question_marks` – control printing of question marks for schematic variables

Using Sessions for Interactive Development

often invoked: '`$ isabelle jedit -l Some_Session Some_Theory.thy`'

- starts Isabelle/jEdit as usual opening `Some_Theory.thy`
- builds heap-image of `Some_Session` on demand
- all theories of `Some_Session` are immediately available in interactive session

Document Preparation

Sectioning and Structuring

- `chapter`, `section`, `subsection`, ... – different levels of sectioning
- ■ – for itemizations
- ► – for enumerations
- `text` ‹ ... › – plain text and LaTeX code

Isabelle Symbols – Lists

symbol	internal	abbreviation
■	`\<^item>`	\ i t e m
►	`\<^enum>`	\ e n u m

General Structure of Document Antiquotations

$$
\begin{array}{rcl}
\mathit{antiquotation} & ::= & @\{\mathit{name}\ \mathit{options}^{?}\ \mathit{arguments}\} \\
& | & \backslash\!<\!\hat{}\,\mathit{name}\!> \mathit{cartouche} \\
& | & \mathit{cartouche} \\
\mathit{options} & ::= & [\,] \mid [\ \mathit{option}\ (,\mathit{option})^{*}\ ] \\
\mathit{option} & ::= & \mathit{name} \mid \mathit{name} = \mathit{name}
\end{array}
$$

Antiquotations

- `text` – uninterpreted inner syntax
- `theory_text` – uninterpreted outer syntax
- `theory` – session-qualified theory name
- `thm` fact^{*} – theorem statements
- `thm [source]` fact^{*} – names of theorems
- `prop` φ – well-typed proposition φ
- `term` t – well-typed term t

Antiquotations (cont'd)

- `value` t – result of evaluating t
- `term_type` t – well-typed term t together with its type
- `typeof` t – type of well-typed term t
- `const` c – constant c
- `typ` τ – well-formed type τ
- `type` κ – type constructor κ
- `method` m – proof method m
- `datatype` τ – data type specification of τ
- `verbatim` – uninterpreted text in typewriter font
- ...
- complete list: `print_antiquotations`
- advantage of antiquotations: hyperlinked, checked, maintainable

Setting Up a Session Root Directory

- use '`$ isabelle mkroot dir`' to set up directory `dir` (can be `.`) as session root
- results in:
  - `dir/ROOT` – session setup for document preparation
    (note the `document_files` section)
  - `dir/document/root.tex` – LaTeX setup
- for BibTeX (together with `cite` antiquotation) create file `document/root.bib` and add `root.bib` to `document_files` section in `ROOT` file

Type Classes

Type Classes in Isabelle

- mechanism to collect all types that support certain operations (like being ordered or having a size/hash/… function)
- each type class comes with an accompanying sort (of same name)
- sorts are used to track type class membership, that is, "being of sort s" is synonymous with "being an instance of class s"
- each type τ has a collection of sorts $s_1, \ldots, s_n$, written $\tau :: \{s_1, \ldots, s_n\}$
- special case $\tau :: \{s\}$ written $\tau :: s$
- default for each type τ is $\tau :: \texttt{type}$

`Demo12.thy` – Transforming Arbitrary Values to Strings

- `CHR ''c''` – literal for character "c" (strings are lists of characters)
- `class` c – introduces new type class c
- `instantiation` $t :: c$ – starts instantiation of type t into class c
- `instance` – start actual instantiation proof

General Form

```
class c = ... +
  fixes c1 and ... and cn
  assumes "..." and ... and "..."
begin
...
end
```

Demo12.thy – Total Orders

- type class may assume properties of fixed constant(s)
  (the comparison relation of a partial order is reflexive, antisymmetric, and transitive)
- type classes may build on each other (total orders are partial orders that are total)

Advantage of Type Classes

- algorithms can be written in generic way; example
  - sorting algorithm: `sort :: "'a :: linorder list ⇒ 'a list"`
- type classes are integrated into type-checking algorithm; examples
  - typing `sort [3, 5]` will enforce `3 :: 'a :: linorder`
  - typing `sort [3 :: int, 5]` will succeed (no manual proof required)
  - typing `sort [(3 :: int, 5 :: int), (2, 7)]` will succeed depending on whether a `linorder`-instance for pairs has been defined/imported

Limitations of Type Classes

- “one shot” instances (if you chose instance once, it is fixed)
- only single type variable as parameter, hence
- no multi-parameter type classes
  (e.g., consider a vector-field with field elements of type `'a` and vectors of type `'b`)

Locales

- more generic than type-classes
  - several parameters
  - several instances
- less automation
- more information: `isabelle doc locales`

Further Reading


Makarius Wenzel.
The Isabelle System Manual.
Isabelle documentation, 2021.


Makarius Wenzel.
Chapter 4 – Document preparation.
In *The Isabelle/Isar Reference Manual*, pages 70–90. 2021.


Florian Haftmann.
Haskell-style type classes with Isabelle/Isar.
Isabelle documentation, 2021.


Clemens Ballarin.
Tutorial to Locales and Locale Interpretation.
Isabelle documentation, 2021.