

# Gencot Functional Properties Proof Manual

Gunnar Teege

August 28, 2022

# Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Isabelle Basics</b>	<b>3</b>
2.1	Invoking Isabelle . . . . .	3
2.1.1	Installation and Configuration . . . . .	3
2.1.2	Theories and Sessions . . . . .	3
2.1.3	Invocation as Editor . . . . .	4
2.1.4	Invocation for Batch Processing . . . . .	5
2.2	Isabelle Theories . . . . .	5
2.2.1	Theory Structure . . . . .	5
2.2.2	Types and Constants . . . . .	6
2.2.3	Definitions . . . . .	6
2.2.4	Overloading . . . . .	6
2.2.5	Statements . . . . .	6
2.2.6	Locales . . . . .	6
2.3	Isabelle Proofs . . . . .	7
2.3.1	Goals . . . . .	7
2.3.2	Proof Scripts . . . . .	7
2.3.3	Backward Reasoning . . . . .	7
2.3.4	Forward Reasoning . . . . .	8
2.4	Isabelle HOL . . . . .	8
2.4.1	Meta Level Operators . . . . .	8
2.4.2	Logical Operators . . . . .	8
2.4.3	Logic Rules . . . . .	8
2.4.4	Equational Reasoning . . . . .	9
2.4.5	The Simplifier . . . . .	9
<b>3</b>	<b>Abstractions for struct Types</b>	<b>10</b>
<b>4</b>	<b>Abstractions for array Types</b>	<b>11</b>

# Chapter 1

## Introduction

This manual describes how to prove functional properties for a Cogent program which resulted from translating a C program using Gencot. It is not a general manual about working with Isabelle. It only uses a very restricted subset of Isabelle features. It does not presume prior knowledge about Isabelle.

The Cogent compiler generates an Isabelle representation of the compiled program, the “shallow embedding”. It mainly consists of definitions in the language of higher order logic for all functions defined in the Cogent program. The goal of the functional properties proofs described in this manual is to systematically develop more abstract definitions which together form an abstract specification of the Cogent program and can be used to prove program properties which are relevant for the use of the program, such as security properties.

The Cogent compiler also generates a C program and a proof that this program is a refinement of the shallow embedding, i.e., it behaves in the same way. Thus, if combined with the refinement proof, the abstract specification can be applied to the C program and the functional properties proofs are also valid for the C program.

## Chapter 2

# Isabelle Basics

Isabelle is a “proof assistant” for formal mathematical proofs. It supports a notation for propositions and their proofs, it can check whether a proof is correct, and it can even help to find a proof.

### 2.1 Invoking Isabelle

After installation, Isabelle can be invoked interactively as an editor for entering propositions and proofs, or it can be invoked noninteractively to check a proof and generate a PDF document which displays the propositions and proofs.

#### 2.1.1 Installation and Configuration

**\*\* todo \*\***

#### 2.1.2 Theories and Sessions

The propositions and proofs in Isabelle notation are usually collected in “theory files” with names of the form `name.thy`. A theory file must import at least one other theory file to build upon its content. For theories based on higher order logic, as it is the case for the Cogent shallow embedding, the usual starting point to import is the theory *Main*.

Several theory files can be grouped in a “session”. A session is usually stored in a directory in the file system. It consists of a file named `ROOT` which contains a specification of the session, and the theory files which belong to the session.

When Isabelle loads a session it loads and checks all its theory files. Then it can generate a “heap file” for the session which contains the processed

session content. The heap file can be reloaded by Isabelle to avoid the time and effort for processing and checking the theory files.

A session always has a single parent session, with the exception of the Isabelle builtin session **Pure**. Thus, every session depends on a linear sequence of ancestor sessions which begins at **Pure**. The ancestor sessions have separate heap files. A session is always loaded together with all ancestor sessions.

Every session has a name of the form **chap/sess** where **chap** is an arbitrary “chapter name”, it defaults to **Unsorted**. The session name and the name of the parent session are specified in the **ROOT** file in the session directory. When a session is loaded by Isabelle, its directory and the directories of all ancestor sessions must be known by Isabelle.

Every session may be displayed in a “session document”. This is a PDF document generated by translating the content of the session theory files to  $\text{\LaTeX}$ . A frame  $\text{\LaTeX}$ document must be provided which includes all content generated from the theory files. The path of the frame document, whether a session document shall be generated and which theories shall be included is specified in the **ROOT** file.

The command

```
isabelle mkroot [OPTIONS] [Directory]
```

can be used to initialize the given directory (default is the current directory) as session directory. It creates an initial **ROOT** file to be populated with theory files names and other specification for the session, and it creates a simple frame  $\text{\LaTeX}$ document.

### 2.1.3 Invocation as Editor

Isabelle is invoked for editing using the command

```
isabelle jedit [OPTIONS] [Files ...]
```

It starts an interactive editor and opens the specified theory files. If no file is specified it opens the file **Scratch.thy** in the user’s home directory. If that file does not exist, it is created as an empty file.

The editor also loads (but does not open) all transitively imported theory files. If these are Isabelle standard theories it finds them automatically. If they belong to the session in the current directory it also finds them. If they belong to other sessions, the option

```
-d <directory pathname>
```

must be used to make the session directory known to Isabelle. For every used session a separate option must be specified. Also the directories of all

ancestor sessions of the session the opened files belong to must be specified using this option if they are not yet known to Isabelle.

The option

```
-l <session name>
```

can be used to specify a session to load (together with all ancestor sessions) to use imported theories from it. If a heap file exists for that session it is used, otherwise a heap file is created by loading all the session's theories.

### 2.1.4 Invocation for Batch Processing

Isabelle is invoked for batch processing of all theory files in one or more sessions using the command

```
isabelle build [OPTIONS] [Sessions ...]
```

It loads all theory files of the specified sessions and checks the contained proofs. It also loads all required ancestor sessions. If not known to Isabelle, the corresponding session directories must be specified using option `-d` as described in Section 2.1.3. Sessions required for other sessions are loaded from heap files if existent, otherwise the corresponding theories are loaded and a heap file is created.

If option `-b` is specified, heap files are also created for all sessions specified in the command. Option `-c` clears the specified sessions (removes their heap files) before processing them. Option `-n` omits the actual session processing, together with option `-c` it can be used to simply clear the heap files.

The specified sessions are only processed if at least one of their theory files has changed since the last processing or if the session is cleared using option `-c`. If option `-v` is specified all loaded sessions and all processed theories are listed on standard output.

If specified for a session in its `ROOT` file (see Section ??), also the session document is generated when a session is processed.

## 2.2 Isabelle Theories

### 2.2.1 Theory Structure

- Theory name
- Imports
- Outer/Inner syntax
- Sections and text

### **2.2.2 Types and Constants**

- Strongly typed
- Type names
- type synonyms and declarations
- Constant names
- Constant declarations

### **2.2.3 Definitions**

- definitions
- abbreviations

### **2.2.4 Overloading**

- overloading
- adhoc overloading

### **2.2.5 Statements**

- theorem, lemma, ...
- statement names
- named statement collections (lemmas, buckets)

### **2.2.6 Locales**

- locale as parameterized theory
- locale parameters
- locale content
- locale interpretation

## **2.3 Isabelle Proofs**

### **2.3.1 Goals**

- prop syntax
- assumptions
- conclusion
- view as rewrite rule
- subgoals (proof state)

### **2.3.2 Proof Scripts**

- method syntax
- apply
- apply scripts
- done, by
- sorry

### **2.3.3 Backward Reasoning**

- Reasoning step
- conclusion unification
- rule with one assumption
- rule with several assumptions
- rule method
- rule\_tac method
- introduction rules
- intro method



### 2.3.4 Forward Reasoning

- assumption unification
- assumption selection
- rotate method
- drule method
- drule\_tac method
- destruction rules
- erule method
- erule\_tac method
- elimination rules

## 2.4 Isabelle HOL

### 2.4.1 Meta Level Operators

- rewrite rules
- $\equiv$
- $\implies$

### 2.4.2 Logical Operators

- $\wedge, \vee, \neg, \longrightarrow$
- $=, \neq, \longleftrightarrow$
- $\forall, \exists$

### 2.4.3 Logic Rules

- conjI, conjE, disjI1, disjI2, disjE, impI, mp
- contrapos\_\*
- iffI, iffE, iffD1, iffD2
- allI, allE, exI, exE

#### **2.4.4 Equational Reasoning**

- Equations, conditional equations
- Substitution
- subst method
- symmetric attribut

#### **2.4.5 The Simplifier**

- simpset, simp attribute
- simp method
- add:, only:, del:
- simp\_all method
- recursive simplification for conditions
- simp trace
- debugging with subst

## Chapter 3

# Abstractions for struct Types

## Chapter 4

# Abstractions for array Types