Logic from Language in Isabelle

Samuel Balco, Alexander Kurz, Larry Moss, Tom Ridge

September 29, 2014

Abstract

The intention is to provide a beginner's tutorial on implementing logics in Isabelle.

Contents

1 Introduction

...

2 $\mathcal{A}$: the logic of All p are q — Section2.thy

theory *Section2*
imports *Main* ~~/*src*/*HOL*/*Library*/*Order-Relation*

begin

2.1 Syntax and Semantics — Types and Definitions in Isabelle

In this section we are going to formalize the proofs of Section 2 of the Lecture Notes. Therefore, we need to find a datatype of formulas of the kind *All p*

are q and in order to do this we even need to first define a datatype of atomic propositions *p,q,...*. After we have done this, we can define the semantics of such formulas in a model, we can implement the derivability relation and proof soundness and completeness.

The set of atomic propositions is formalized as a type.[1] Roughly speaking types are sets as known from set theory, but there are some differences. Making good use of types is one of the things one has to learn using Isabelle/HOL. For example, formalizing atomic proposition as a type imposes a constraint on the set of propositions:

> Types in Isabelle are non-empty.

typedecl *atProp*

Next we use the **datatype** declaration to define formulas. *All* is called a constructor, which can be considered as a function that takes two arguments of type *atProp* and produces an element of type *formula*. The addition (*All* _ *are* _) is optional, but allows us to write, as in the lecture notes, *All p are q* instead of *All p q*.

datatype *formula* = *All atProp atProp* (*All* - *are* -)

Inductively defined sets are best formalized using the datatype declaration, for example, because Isabelle then provides inbuilt support for induction. The **print-theorems** command is not necessary, but if you move your mouse over it in Isabelle you can see which theorems you get for free by using the datatype declaration. For example

```
formula.eq.simps:
  equal_class.equal All ?atProp1 are ?atProp2 All ?atProp1' are ?atProp2'
  \<equiv> ?atProp1 = ?atProp1' \<and> ?atProp2 = ?atProp2'
```

says that two formulas are equal iff their respective atomic propositions are equal.[2]

The next definition describes models over a carrier as models parametric in an an aribrary (non-empty) type called *'a*. The arrow ⇒ and *set* are type constructors, where *set* is the type of all subsets of *'a* and ⇒ constructs the type of all set-theoretic functions. To summarize, a model over *'a* is a function from atomic propositions to the powerset of *'a*, that is, it is an element of the type (*atProp* ⇒ *'a set*), for which we introduce the abbreviation[3]

type-synonym *'a model* = (*atProp* ⇒*'a set*)

In particular, the notion of model above cannot have empty carrier. Alternatively, we could have defined a model over *'a* as a pair of a possible empty

[1] See [6, p.173] for further explanations on `typedecl`.
[2] See [6, Section 8.5.2] (and in particular p.176) and [3, Section 2.4] for more details.
[3] See [6, p.24] for more on `type_synonym`.

subset of *'a* and a function (*atProp* ⇒ *'a set*) but then we would need an extra condition to make sure that the image of the function lies inside that carrier. In other words, we get the additional complication of specifying the well formed models among the premodels as in

```
type-synonym 'a pre-model = 'a set × (atProp ⇒ 'a set)
definition wf-model :: 'a pre-model ⇒ bool
  where wf-model M ≡ let (M,f) = M in (! p. f p ⊆ M)
```

In the following we will only use the first alternative and not make use of pre-model and wf-model.

The lecture notes continue with the definition of satisfiability

$$\mathcal{M} \models \mathsf{All}\ p\ \mathsf{are}\ q \quad \text{iff} \quad [\![p]\!] \subseteq [\![q]\!]$$

which we formalize as follows.

```
fun satisfies :: 'a model ⇒ formula ⇒ bool (- ⊨ - )
where
  satisfies M (All x are y) = (M x ⊆ M y)
```

The lecture notes continue with the following example.

Example. Let $\mathbf{P} = \{n, p, q\}$. Let $M = \{1, 2, 3, 4, 5\}$. Let $[\![n]\!] = \emptyset$, $[\![p]\!] = \{1, 3, 4\}$, and $[\![q]\!] = \{1, 3\}$. This specifies a model $\mathcal{M}$. The following sentences are true in $\mathcal{M}$: All n are n, All n are p, All n are q, All p are p, All q are p, and All q are q. The other three sentences in $\mathcal{A}$ are false in $\mathcal{M}$.

The interesting question is how to formalise the three "let" statements above. In particular, the "let $M = \ldots$" is formalized differently than the other two. The reason is that the carrier M of the model is formalized as a type parameter. Therefore we can instantiate it by defining a datatype:

```
datatype example-2-1 = e1 | e2 | e3 | e4 | e5
```

The other two "let" statements are translated as assumptions. For the first one, note that (*UNIV :: atProp set*) is the Isabelle way of naming *UNIV* as the largest set in *atProp set*, that is, *UNIV* is *atProp* seen as a set. For the second note that Isabelle infers the type of the carrier of *M* upon reading *M p* = {*e1,e3,e4*}, the elements of which are elements of the type *example-2-1*. We can now formalize and prove the claims made in the exercise.

```
lemma example-2-1:
assumes (UNIV :: atProp set) = {n,p,q} ∧ n≠p ∧ n≠q ∧ p≠q
assumes M n = {}  ∧  M p = {e1,e3,e4}  ∧  M q = {e1,e3}
shows M ⊨ All n are n
  and M ⊨All n are p
  and M ⊨All n are q
  and M ⊨ All p are p
  and M ⊨ All q are p
```

and $M \models$ *All q are q*
and $\neg\ (M \models$ *All p are n*$)$
and $\neg\ (M \models$ *All p are q*$)$
and $\neg\ (M \models$ *All q are n*$)$

The proof of the 9 statements can be done automatically using sledgehammer [2]. Sledgehammer is an Isabelle command that calls automatic theorem provers, either locally installed or over the internet, in order to prove the current goal. For us, sledgehammer is a crucial ingredient of Isabelle. It allows us to conduct proofs close to the level of pen and paper mathematics. Whenever in these notes, or in the theory files, you see a **by**(*metis ...*) this shows a proof found by sledgehammer. Actually, for better readability of this document, we made most of these automatically found proofs invisible as can easily be seen by comparing directly with the theory file.

We emphasize that the proofs **by**(*metis ...*) are not meant to be readable by humans. In our notes, they are usually proofs that are obvious on the mathematical level of abstraction. On the other hand, if you know that such a proof depends on a particular fact it is a good idea to check whether this fact is contained in the list of facts appearing in the dots of a **by**(*metis ...*).

video

The lecture notes continue with extending satisfiability to sets of formulas

$$\mathcal{M} \models \Gamma \text{ iff } \mathcal{M} \models \phi \text{ for every } \phi \in \Gamma$$

and to semantic entailment between sets of formulas and formulas

$$\Gamma \models \phi \text{ iff for all } \mathcal{M}\text{: if } \mathcal{M} \models \Gamma\text{, then also } \mathcal{M} \models \phi.$$

which we formalize as follows.

> Overloading names, although possible to a degree, is not as easy in Isabelle as it is in pen and paper maths. For example, comparing the definitions of *satisfies* and *M-satisfies-G* one would think that, as they have different types, both could well have the same name, or, at least, use the same notation $\models$. But one problem here is that the parsing is done before the type checking and ambiguous parse trees would result (recommended experiment). So this is a point that requires some compromise from the mathematician's perspective.

definition *M-satisfies-G* :: *'a model* ⇒ *formula set* ⇒ *bool* (- ⊨*M* -)
where
M-satisfies-G M G ≡ ∀*f*. *f* ∈ *G* ⟶ (*M* ⊨ *f*)

If we want to define semantic consequence or entailment, we want to say that $G \models g$ iff $\forall M :: 'a\ model. M \models G \rightarrow M \models g$. Due to the free type variable $'a$ we run into a slight technical difficulty. For reasons explained in some detail in [1], every variable on the right of a definition also needs to appear on the left. So we could write, using a dummy term *ty*, ...

```
definition entails-b :: formula set ⇒ formula ⇒ 'a ⇒ bool
where
  entails-b G f (ty::'a) ≡ ∀ (M::'a model). (M ⊨M G) ⟶ (M ⊨ f)
```

... but this definition has the disadvantage that it looks as if the function *entails* depended on values of type $'a$. One way out is to use the "type itself" construction, which provides a type depending on $'a$ with only element (this element is denoted by TYPE($'a$) and we will encounter it later). More on the itself type can be found in [8].

```
definition entails :: formula set ⇒ formula ⇒ 'a itself ⇒ bool (- ⊨G - -)
where
  entails G f (ty::'a itself) ≡ ∀ (M::'a model). (M ⊨M G) ⟶ (M ⊨ f)
```

Further Reading. For general background on Isabelle see Paulson's original [7] (in particular Section 1 and 2). [7] also gives a reason why types are assumed to be non-empty. In [6, p.119] one can find a model checker as a worked out example.

2.2 Proof Theory — Rules and Natural Deduction in Isabelle

The deductive system $\mathcal{A}$ is given as

$$\frac{}{\mathsf{All}\ p\ \mathsf{are}\ p}\ \textsc{Axiom} \qquad \frac{\mathsf{All}\ p\ \mathsf{are}\ n \quad \mathsf{All}\ n\ \mathsf{are}\ q}{\mathsf{All}\ p\ \mathsf{are}\ q}\ \textsc{Barbara}$$

which is formalized as follows. (The 0 in $0 \vdash$ - is an annotation to avoid overloading wrt to - $\vdash$ - in the next definition.)

```
inductive derivable :: formula ⇒ bool (0⊢ -)
where
  A-axiom: 0⊢ (All X are X)
| A-barbara: ⟦0⊢(All X are Y); 0⊢(All Y are Z)⟧ ⟹ 0⊢(All X are Z)
```

Isabelle implements natural deduction. The Isabelle notation for a rule of the shape

$$\frac{\mathsf{A} \quad \mathsf{B}}{\mathsf{C}}\ \mathit{name}$$

is

$$\mathit{name}\text{: } [\![\mathrm{A};\mathrm{B}]\!] \Longrightarrow \mathsf{C}$$

as in the rules *A-axiom* and *A-barbara* above.

Something about "inductive"

Let us look at how to reason in Isabelle using the natural deduction rules. The following example is again from the lecture notes.

Example. Let Γ be

$$\{\mathsf{All}\ l\ \mathsf{are}\ m, \mathsf{All}\ q\ \mathsf{are}\ l, \mathsf{All}\ m\ \mathsf{are}\ p, \mathsf{All}\ n\ \mathsf{are}\ p, \mathsf{All}\ l\ \mathsf{are}\ q\}$$

Let ϕ be $\mathsf{All}\ q\ \mathsf{are}\ p$. Here is a proof tree showing that $\Gamma \vdash \phi$:

$$\dfrac{\dfrac{\mathsf{All}\ q\ \mathsf{are}\ l \qquad \dfrac{\dfrac{\mathsf{All}\ l\ \mathsf{are}\ m \qquad \dfrac{}{\mathsf{All}\ m\ \mathsf{are}\ m}\ \textsc{Axiom}}{\mathsf{All}\ l\ \mathsf{are}\ m}\ \textsc{Barbara} \qquad \mathsf{All}\ m\ \mathsf{are}\ p}{\mathsf{All}\ l\ \mathsf{are}\ p}\ \textsc{Barbara}}{\mathsf{All}\ q\ \mathsf{are}\ p}\ \textsc{Barbara}$$

The proof tree can be implemented in Isabelle as follows.

```
lemma example-2-5:
assumes 0⊢All l are m
and 0⊢All q are l
and 0⊢All m are p
and 0⊢All n are p
and 0⊢All l are q
shows 0⊢ All q are p

apply (rule-tac Y=l in A-barbara)
 apply (rule assms(2))

 apply (rule-tac Y=m in A-barbara)
  apply (rule-tac Y=m in A-barbara)
   apply (rule assms(1))
```

```
  apply (rule A-axiom)
 apply (rule assms(3))
done
```

Comparing the Isabelle proof with the paper proof, we see that in Isabelle we reasoned backwards from the goal *All q are p*. The first **apply** (*rule-tac Y=l in A-barbara*), for example, corresponds to the bottom application of the BARBARA rule. video

here it would be good to have an exercise for the reader

Later in the lecture notes we want to compare derivability from assumptions with semantic entailment, so we define $\Gamma \vdash \phi$ as a variation of the above $\vdash \phi$:

```
inductive derives :: formula set ⇒ formula ⇒ bool (- ⊢ -)
where
  A-assumption: f ∈ hs ⟹ hs ⊢ f
| A-axiom: hs ⊢ (All X are X)
| A-barbara: ⟦hs ⊢(All X are Y); hs ⊢(All Y are Z)⟧ ⟹ hs ⊢(All X are Z)
```

Accordingly, we have the following variation of the above example.

```
lemma example-2-5-b:
fixes l m n p q
defines G-2-5 ≡ {All l are m,All q are l,All m are p,All n are p,All l are q}
shows G-2-5 ⊢ All q are p
apply (rule-tac Y=l in A-barbara)
 apply (rule A-assumption) apply (simp add: G-2-5-def)
  apply (rule-tac Y=m in A-barbara)
    apply (rule-tac Y=m in A-barbara)
     apply (rule A-assumption) apply (simp add: G-2-5-def)

     apply (rule A-axiom)
    apply (rule A-assumption) apply (simp add: G-2-5-def)
done
```

The proof is similar to the first one, but instead of referring to an assumption of the lemma, we use now the rule *A-assumption*. After applying the rule *A-assumption*, Isabelle leaves us with a subgoal, namely *All q are l ∈ G-2-5* which is obviously true and easy to discharge by **apply** (*simp add: G-2-5-def*). [4]

maybe a bit more about auto, simp, etc ... [6, Section 3.1]

[4] A summary of rules is given in [6, Section 5.7]. Chapter 5 also contains a general introduction to the logic of Isabelle.

Next we come to our first mathematical result on the calculus, namely that it is sound. Recall that *TYPE*(*atProp*) is the unique element of the type *atProp itself*, which appears in the definition of $\models G$.

```
lemma prop-2-2-1:
  fixes G g
  assumes G ⊢ g
  shows  G ⊨G g (TYPE(atProp))
using assms
proof (induct rule: derives.induct)
  case (A-assumption)
  show ?case by (metis A-assumption.hyps M-satisfies-G-def entails-def)
next
  case (A-axiom)
  show ?case by (metis entails-def order-refl satisfies.simps)
next
  case (A-barbara)
   show ?case by (metis A-barbara.hyps(2) A-barbara.hyps(4) entails-def satis-
fies.simps subset-trans)
qed
```

To quote from the book: The soundness proofs of all the logical systems in this book are all pretty much the same. They are always inductions, and the crux of the matter is usually a simple fact about sets. (Above, the crux of the matter is that the inclusion relation $\subseteq$ on subsets of a given set is always a transitive relation.) We almost never present the soundness proof in any detail. The Isabelle proof gives another justification for skipping the details of the mathematical soundness proof: After telling Isabelle that we want to use induction and after listing the cases, the details of the proof are done automatically using sledgehammer, that is, the three lines above **by** (*metis ...*) are provided by an automatic theorem prover. For us, this essentially means that no mathematical ideas are needed. [5]

It may also be worth noting that the Isabelle proof is quite a bit shorter than the proof in the book. Usually, such easy proofs do not make it into the literature. Nowadays that can be done without sacrificing rigor by delegating these proofs to a theorem prover. Moreover, checking all the cases using the tool may well be faster then checking the cases (carefully) by hand.

Further Reading. Section 3 and 4 of [7] contain an exposition of Isabelle's meta-logic and an example of an object logic.

[5] Although it is clear that with automatic theorem proving becoming more and more powerful it will happen more and more often that automatic provers will find proofs that would require ideas from a human point of view. On the other hand, for the proofs of this section, there is an astonishingly good match between the details one would hide in a pen and paper proof and the details that can be discharged by an automatic theorem prover.

2.3 Completeness — Proofs in Isabelle/Isar

Isabelle supports two styles of proof, apply-style and Isar-style. We have seen an example of both. The proof for Example 2.5 was given apply-style. This is very convenient for that type of problem, which consists in finding a derivation in a calculus, the rules of which have been directly encoded using Isabelle rules.

But the proof of the soundness result in Proposition 2.2.1 was written in a very different style. As an exercise you might want to try and do the soundness proof apply-style, starting with

```
apply (rule derives.induct)
apply (rule assms)
```

You will notice that you will need to know quite a bit about how Isabelle proofs are implemented internally. Instead, the Isar proof language [9] allows proofs to be written in a style much closer to informal proofs and is much quicker to learn. This section will present more examples and some exercises, for background and details see the introduction [5]. A useful resource is also the Isabelle/Isar reference manual [9], in particular Appendix A.

Coming back to the development of the syntax and semantics of our logic, it is the next definition which is the crucial one as it links up both syntax and semantics: Given a theory G, derivability induces a preorder on atomic propositions, which is then used to define the notion of canonical model below.

definition *less-equal-atprop* :: *atProp* ⇒ *formula set* ⇒ *atProp* ⇒ *bool* (- ≲- -)
where
  less-equal-atprop u G v ≡ *G* ⊢ *All u are v*

We show that $\lesssim$ is a preorder. We would hope that there is a theory about orders and preorders ready for us to use and to find it we write:

find-theorems *name:preorder*

After which, in the output window, we find

```
Order_Relation.preorder_on_empty: preorder_on {} {}
```

Pressing the command key, hovering over the line above and clicking without delay brings up the theory `Order_Relation` and the definition of `preorder_on`, which takes two arguments, a set A and a relation, ie, a subset of $A \times A$. The relation in question is $\{(x, y) \mid x \lesssim G\ y\}$. The set on which this relation lives can be given by using *UNIV*. video

lemma *prop-2-4-1*:
  fixes *G*

```
  shows preorder-on UNIV { (x,y). x ≲G y }
proof (auto simp add: preorder-on-def)
   show refl { (x,y). x ≲G y }
    unfolding refl-on-def
     using A-axiom by (metis UNIV-Times-UNIV case-prodI iso-tuple-UNIV-I
less-equal-atprop-def mem-Collect-eq subset-iff)
  show trans { (x,y). x ≲G y }
    unfolding trans-def
    using A-barbaraqed
```

As a side remark, the latex code of Proposition 2.4.1 and its proof in the book is only 10% shorter, measured in number of characters, than the Isabelle code (only part of which is shown above).

Next we give the definition of the canonical model of a theory G. In words it says that in the canonical model the interpretation of the atomic proposition u is the set of all atomic propositions smaller or equal to u.

```
definition canonical-model G u ≡ { v. v ≲G u }
```

The next lemma confirms that the canonical model of G satisfies G:

```
lemma lemma-2-4-2:
  fixes G assumes M = canonical-model G
  shows M ⊨M G
proof (auto simp add: M-satisfies-G-def)
  fix g  assume g ∈ G
  then obtain p q where (All p are q) = g
  then have p ≲G q using A-assumption
  then have M p ⊆ M q using A-barbara
  thus M ⊨ g
qed
```

Next comes the completeness theorem.

```
lemma thm-2-4-3:
  assumes G ⊨G (All p are q) (TYPE(atProp))
  shows G ⊢ All p are q
proof −
  def M ≡ canonical-model G
  have M ⊨ All p are q using lemma-2-4-2
  then have M p ⊆ M q

  have p ≲G p using A-axiom
  then have p ∈ M p using canonical-model-def

  have p ∈ M q using ‹M p ⊆ M q›
  then have p ≲G q using canonical-model-def
  thus ?thesis
qed

end
```

3 How to write your own Isar proofs

theory *Section2-exercises*
imports *Main ~~/src/HOL/Library/Order-Relation*
Section2

begin

This section is exeperimental. It asks the question whether the examples and explanations from the previous section are enough to get the reader started on its own proofs. Ideally, the reader would be able to make up his own lemmas and proofs, but it is likely that the previous section is not enough of a preparation for this. Therefore, this section gives some exercises with hints and additional explanations, in an attempt to bridge the gap.[6]

Probably the most useful references for writing proofs in Isar are [5] and Section 2.2.3 and Appendix A in [9].

Here are a few items worth noticing for your first Isabelle exercises:[7]

- This document does not show the full Isabelle source code, so you need to refer to Section2.thy to see the code of the previous section.
- It is important to keep the spacing as eg in `M p` and not to write `Mp`.
- To write the symbol $\subseteq$ in Isabelle, one can either find and select it in the "Symbols" windows, or one can start typing `\subseteq` as in latex and use the tab-key to select it. Similarly, for $\models$ type `|=` and tab.
- If you inspect the file Section2.thy in an editor, you will find that what appears as $\subseteq$ in jEdit is written `\<subseteq>` in ascii.
- In case the parser is complaining try to put some extra round brackets.
- Isabelle distinguishes between inner and outer syntax. Inner syntax must appear between quotes in outer syntax. These quotes have been suppressed in the latex document, but are important in the theory file Section2.sty. Single tokens such as `atProp` in `typedecl atProp` do not need to be put between quotes.
- To get an idea what error messages such as "Illegal application of proof command in state mode" are about, have a look at Fig 2.1 in [9, Section 2.2.3].
- ...

[6] Also refer to future sections that deal with more aspects of Isabelle
[7] This list needs to be extended and then probably structured in some good way.

The exercises are (at least at the moment) not ordered according to a didactic concept but follow the order of the lecture notes

Example 2.1 states that there are exactly 9 formulas in $\mathcal{A}$. Here we suggest to prove this claim in Isabelle. As a warm-up we do:

lemma *example-2-1-aa*:
assumes (*UNIV* :: *atProp set*) = {*n*}
shows *card* (*UNIV*::*formula set*) = *1*
proof −
obtain *A* **where** *A* ≡ *UNIV* :: *formula set* **by** *force*
then have *A* = {*All n are n*} **by** (*metis* (*full-types*) *UNIV-I assms empty-not-UNIV formula.exhaust singleton-iff subsetI subset-singletonD*)
then have *card A* = *1* **by** *fastforce*
thus *?thesis* **by** (*metis* ‹*A* ≡ *UNIV*›)
qed

The exercise is now to show the following. (Warning: this exercise may be more difficult than the others). [8]

lemma *example-2-1-ab*:
assumes (*UNIV* :: *atProp set*) = {*n*,*p*,*q*}
and *n*≠*p* **and** *n*≠*q* **and** *p*≠*q*
shows *card* (*UNIV*::*formula set*) = *9*

Example 2.2 from the lecture notes. Hint: Start the proof with
`unfolding entails_def`
`proof rule+`.[9]

lemma *example-2-2*:
shows {*All p are q*, *All n are p*} ⊨*G All n are q TYPE*(*atProp*)

Exercise 2.4.1: Show in Isabelle that the canonical model of G satisfies all semantic consequences of G. Hint: That shoule be easy: just use sledgehammer.

lemma *isacise-2-4-1*:
fixes *G*
assumes *M* = *canonical-model G*
assumes *G* ⊨*G g* (*TYPE*(*atProp*))
shows *M* ⊨ *g*

Exercise 2.4.2: Show in Isabelle that if the canonical model M of G satisfies

[8] Hint: Construct explicitely the set of all formulas using *def X* ≡ {*All n are n*, . . . } , then use sledgehammer to prove that this set has 9 elements and then go on to prove *X* = *A* (where *A* is as in *example-2-1-aa*).

[9] `rule` applies an introduction rule and `+` denotes a tactics that iterates this as long as possible [4].

g then $G \vdash g$. Hint: This should be similar to *lemma-2-4-2* of the previous section.

lemma *isacise-2-4-2*:
  fixes G g M
  assumes M = *canonical-model* G
  assumes $M \models g$
  shows $G \vdash g$

end

References

[1] R. Arthan. HOL Constant Definition Done Right. In *ITP*, pages 531–536, 2014.

[2] J. C. Blanchette. Hammering Away, A Users Guide to Sledgehammer for Isabelle/HOL. http://isabelle.in.tum.de/dist/Isabelle2013-2/doc/sledgehammer.pdf, December 2013.

[3] J. C. Blanchette, L. Panny, A. Popescu, and D. Traytel. Defining (Co)datatypes in Isabelle/HOL. http://www.cl.cam.ac.uk/research/hvg/Isabelle/dist/Isabelle2013-2/doc/datatypes.pdf, December 2013.

[4] P. Lammich. https://lists.cam.ac.uk/pipermail/cl-isabelle-users/2009-September/msg00022.html.

[5] T. Nipkow. A Tutorial Introduction to Structured Isar Proofs. http://isabelle.in.tum.de/website-Isabelle2009-1/dist/Isabelle/doc/isar-overview.pdf.

[6] T. Nipkow, L. C. Paulson, and M. Wenzel. Isabelle/HOL A Proof Assistant for Higher-Order Logic. http://www.cl.cam.ac.uk/research/hvg/Isabelle/dist/Isabelle2013-2/doc/tutorial.pdf, December 2013.

[7] L. C. Paulson. The foundation of a generic theorem prover. *J. Autom. Reasoning*, 5(3), 1989. available at http://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-130.pdf.

[8] M. Wenzel. Type classes and overloading in higher-order logic. In *TPHOLs*, pages 307–322, 1997.

[9] M. Wenzel. The Isabelle/Isar Reference Manual. http://www.cl.cam.ac.uk/research/hvg/Isabelle/dist/Isabelle2013-2/doc/isar-ref.pdf, December 2013.

A On using Isabelle's document processing facilities

"Isabelle/HOL A Proof Assistant for Higher-Order Logic", December 5, 2013, p.58 explains how to set-up and use Isabelle's document processing facilities.

Typing

```
isabelle build -D .
```

produces a latex file

```
./output/session.tex
```

which in turn is called upon by

```
./document/root.tex
```

from which it is produced a pdf file

```
./output/document/root.pdf
```

If compilation produces a latex error this is not advertised to the user directly. Rather there is a short message "FAILED" and then the name of the log file which contains the latex error messages.