# Automated Reasoning: Coursework 1

**Proving and Reasoning in Isabelle/HOL**

Robert Atkey  
bob.atkey@ed.ac.uk  
3rd February 2015

## 1 Introduction

This is the first coursework assignment for the Automated Reasoning course. It is divided into two parts. In the first part, you will get some experience with the rules of natural deduction, building on the exercises that are on the course web page. In the second part, you will be introduced to a particular technique for automated reasoning, Quantifier Elimination (QE), and be asked to prove some of the key lemmas used to prove that a QE procedure for the theory of dense linear orders is correct.

There are two (essentially identical) versions of this document. The source version, which you can load into Isabelle to complete your proofs:

http://www.inf.ed.ac.uk/teaching/courses/ar/coursework/ARCoursework1.thy

and the PDF version, which is easier to read:

http://www.inf.ed.ac.uk/teaching/courses/ar/coursework/ARCoursework1.pdf

You should fill in the source version with your proofs for submission. Submission instructions are at the end of this document.

The deadline for submission is **2pm, Friday 27th February 2015**.

## 2 Marks

You will only earn marks for an unfinished proof if you provide some explanation as to your proof strategy or an explanation as to why you are stuck. You may also earn marks if you can prove the theorem by asserting a sensible lemma with `lemma` and “proving” it with the “cheat” command `sorry`. Be careful to note the restrictions on the proof methods that you are allowed to use for each question.

## 3 Background Reading

This assignment uses the interactive theorem prover Isabelle that has been introduced in the lectures and exercises.

As you will be using Isabelle interactively, you will need to be familiar with the system before you start. Formalized mathematics is not trivial! You will find this assignment much easier if you attend the lectures, attempt the various Isabelle exercises given on the course webpages, and ask questions about using Isabelle before you start. It is recommended that you read Chapter 5 of the Isabelle/HOL tutorial located at:


We also recommend that you use the Isabelle Cheat Sheet from Jeremy Avigad, which can be found at:

http://www.inf.ed.ac.uk/teaching/courses/ar/FormalCheatSheet.pdf
Section 1.5 of the course textbook *Logic in Computer Science* by Huth and Ryan contains some more background information about the disjunctive normal used in Section 5.2 below. However, it should be possible to complete this coursework without consulting the book.

## 4 Part 1: Natural Deduction in Isabelle/HOL (40 marks)

In this section, you will get some practice with natural deduction by proving some theorems from propositional and first-order logic. Each of these theorems could be solved directly with Isabelle’s automatic tactics, but here, you are asked to use only the following basic introduction and elimination rules:

- **conjI**: \[ ?P; ?Q \] \implies ?P \land ?Q
- **conjE**: \[ ?P \land ?Q \] \implies ?P
- **conjunct1**: ?P \land ?Q \implies ?P
- **conjunct2**: ?P \land ?Q \implies ?Q
- **disjI1**: ?P \implies ?P \lor ?Q
- **disjI2**: ?Q \implies ?P \lor ?Q
- **disjE**: \[ ?P \lor ?Q; ?P \implies ?R; ?Q \implies ?R \] \implies ?R
- **impI**: \( ?P = \implies ?Q \) \implies ?P \rightarrow ?Q
- **mp**: \[ ?P \rightarrow ?Q; ?P \] \implies ?Q
- **notI**: \( ?P = \implies \text{False} \) \implies \neg ?P
- **notE**: \[ \neg ?P; ?P \] \implies ?R
- **iffI**: \[ ?P = \implies ?Q; ?Q = \implies ?P \] \implies ?P = ?Q
- **iffD1**: \[ ?Q = ?P; ?Q \] \implies ?P
- **iffD2**: \[ ?P = ?Q; ?Q \] \implies ?P
- **allI**: \( \forall x. ?P x \rightarrow \forall x. ?P x \)
- **allE**: \[ \forall x. ?P x; ?P ?x \rightarrow ?R \rightarrow ?R \]
- **exI**: \[ ?P ?x \rightarrow \exists x. ?P x \]
- **exE**: \[ \exists x. ?P x; \exists x. ?P x \rightarrow ?Q \rightarrow ?Q \]

You may also use the following classical rules:

- **excluded_middle**: \( \neg ?P \lor ?P \)
- **ccontr**: \( \neg ?P \rightarrow \text{False} \rightarrow ?P \)

Note that you can display any of these theorems, and any other named theorem, even while in the middle of a proof. For instance, to display the rule **conjI**, just use the following command:

\[ \text{thm conjI} \]

In each step of the proof, you may use apply with any of the methods **rule**, **erule**, **drule**, **frule**, and their variants **rule_tac**, **erule_tac**, **drule_tac** and **frule_tac**. You will also need to use the method **assumption**, and you may also use the commands **defer** and **prefer** to manipulate the goal-stack during a proof. You are not permitted to use any other proof methods for this part.

Prove the following lemmas by replacing the cheat proof **oops** with a real proof in each case:

- **lemma** \( P \rightarrow P \)
- **lemma** \( P \land Q \rightarrow Q \land P \)
- **lemma** \( P \lor Q \rightarrow Q \lor P \)

Prove the following lemmas by replacing the cheat proof **oops** with a real proof in each case:

- **lemma** \( P \rightarrow P \) (1 mark)
- **lemma** \( P \land Q \rightarrow Q \land P \) (1 mark)
- **lemma** \( P \lor Q \rightarrow Q \lor P \) (1 mark)
lemma \((Q \land R) \land P \rightarrow (P \land R) \land Q\)
oopsort{2 marks}

lemma \(A \implies (B \land C) \land D \implies E \implies F \implies D \land A \land C\)
oopsort{2 marks}

lemma \((A \implies B) \land (C \implies D) \land (A \lor C) \implies B \lor D\)
oopsort{2 marks}

lemma \((Q \lor R) \land P \rightarrow \neg P \rightarrow Q\)
oopsort{3 marks}

lemma \(\neg P \rightarrow Q \rightarrow (\neg P \rightarrow \neg Q) \rightarrow P\)
oopsort{3 marks}

lemma \(\neg (P \land Q) \rightarrow \neg P \lor \neg Q\)
oopsort{4 marks}

lemma \(\forall x y. R x y \rightarrow (\forall x. R x x)\)
oopsort{3 marks}

lemma \(\forall x. P x \rightarrow Q x \rightarrow \neg (\exists x. P x \land \neg Q x)\)
oopsort{4 marks}

lemma \(\exists x. \forall y. P x y \rightarrow (\forall y. \exists x. P x y)\)
oopsort{4 marks}

lemma \(\neg (\exists \text{barber. } man \text{ barber } \land (\forall x. \text{man } x \land \neg \text{shaves } x x \leftrightarrow \text{shaves barber } x))\)
oopsort{5 marks}

lemma \((\forall x::\text{int}. (\exists y. P x y) \rightarrow Q x) \land (\forall y (x::\text{int}) y. R x y \rightarrow R y x) \rightarrow (\forall z. P a z \rightarrow Q z)\)
oopsort{5 marks}

5 Part 2 : Introduction to Quantifier Elimination (60 marks)

Quantifier elimination is a technique for deciding whether or not a first-order formula is true in some theory. Informally, a theory is a set of formulas that captures all the true statements about some class of mathematical structures, e.g., the theory of groups is the collection of all statements that are true about all groups.

(form this paragraph can be skipped) Formally, a first-order theory is a set \(T\) of first-order formulas that is closed under entailment. That is, for any formulas \(P\) and \(Q\) such that \(P \in T\) and \(P \models Q\), then \(Q \in T\). If we have a formula \(P \in T\), then we write \(T \models P\) to indicate that \(P\) is a consequence of the theory \(T\). For any set of formulas \(\Sigma\) such that every formula \(P \in T\) is entailed by \(\Sigma\), then we say that the theory \(T\) is axiomatized by \(\Sigma\). If there is a finite set \(\Sigma\) that axiomatizes \(T\), then we say that \(T\) is finitely axiomatizable. When writing informally, we usually describe a theory by the set of axioms that axiomatizes it.

If a theory \(T\) supports quantifier elimination then it is possible to take any first-order formula \(P\) that uses the same function and predicate symbols as \(T\) and generate an equivalent formula \(Q\) that has no quantifiers and no additional free variables. If the original formula \(P\) had no free variables, then \(Q\) will also have no free variables, and in many cases it is possible to easily decide whether or not \(Q\) is in the theory \(T\). Since \(Q\) is equivalent to the original formula \(P\), we will have decided whether or not \(P\) is true in the theory \(T\).

In this part, we will look at the quantifier elimination procedure for the first-order theory of dense linear orders. First, we will see how to locally introduce axiomatic theories in Isabelle using locales.

5.1 Part 2.1 : Local Axiomatic Reasoning in Isabelle

In Lecture 6, I presented two different methods for extending Isabelle so that it is able to state and prove theorems about interesting mathematical theories. The first method is to extend Isabelle by stating new axioms that make unproven assertions about some theory. For example, we could state
the Peano axioms for arithmetic. The second method is to extend Isabelle by making definitions, building on the underlying logic or on previous definitions.

In general, extension by definition is to be preferred, because it guarantees that we do not introduce an inconsistency into Isabelle, which would render all the proofs we carry out in Isabelle worthless. However, sometimes we want to reason locally from some axioms in order to prove statements about classes of mathematical objects. An example of this is proving statements about all groups, where this is possible, rather than repeatedly proving the same statement for each group of interest.

Isabelle provides a facility for locally assuming axioms using a feature called locales. We can declare a new locale for the theory of groups as follows:

```isabelle
locale group =
  fixes mult :: 'a ⇒ 'a ⇒ 'a
  and unit :: 'a
  and inverse :: 'a ⇒ 'a
  assumes left-unit: mult unit x = x
  and associativity: mult (mult x y) z = mult x (mult y z)
  and left-inverse: mult (inverse x) x = unit
```

A locale fixes a collection of function symbols (and predicate symbols if we include functions whose type ends in bool), and a collection of axioms.

Declaring a new locale does not introduce its axioms into Isabelle’s collection of known theorems, as it does if we declare axioms using the axiomatization keyword (see the exercise accompanying Lecture 6). Instead, we write (in <locale-name>) after lemma (or definition or theorem) to indicate that we using the theory of groups:

```isabelle
lemma (in group) associativity-backwards: mult x (mult y z) = mult (mult x y) z
apply (subst associativity)
apply (rule refl)
done
```

If the declaration in group was removed, then Isabelle would treat the symbol mult as indeterminate and complain that it didn’t know about the theorem associativity.

For this coursework, we will be interested in a particular axiomatic theory called dense linear orders, which supports quantifier elimination. A dense linear order is some set with:

1. A less than relation \( x \sqsubseteq y \), that is transitive (if \( x \sqsubseteq y \) and \( y \sqsubseteq z \) then \( x \sqsubseteq z \)) and irreflexive (it is never the case that \( x \sqsubseteq x \)).

2. It is linear: all the elements of the set are arranged along a line. Formally, for any pair \( x, y \), either they are equal or \( x \sqsubseteq y \) or \( y \sqsubseteq x \).

3. It is dense: between every pair \( x, y \) such that \( x \sqsubseteq y \), there lies another element \( z \).

Examples of dense linear orders include the real numbers \( \mathbb{R} \) and the rationals \( \mathbb{Q} \), with the usual less than ordering. An non-example is the integers \( \mathbb{Z} \), because it is not dense (there is no integer between 0 and 1, for example).

We can describe the theory of dense linear orders by using the following locale declaration in Isabelle, which translates our informal axioms into Isabelle notation:

```isabelle
locale dense-linear-order =
  fixes lt :: 'a ⇒ 'a ⇒ bool (infixl ⊑ 50)
  assumes transitivity: [x ⊑ y; y ⊑ z] ⊢ x ⊑ z
  and irreflexivity: ∼ (x ⊑ x)
  and linearity: x = y ∨ x ⊑ y ∨ y ⊑ x
  and dense: x ⊑ y ⊢ (∃ z. x ⊑ z ∧ z ⊑ y)
```

Isabelle also allows the declaration of new locales by extending existing ones with new axioms. For the first quantifier elimination procedure below, we will be interested in dense linear orders that are unbounded. For every element \( x \), there is some element below \( x \) and some element above \( x \):
locale unbounded-dense-linear-order = dense-linear-order +
  assumes no-top : ∃ y. x ⊏ y
  and no-bottom : ∃ y. y ⊏ x

5.2 Part 2.2: Quantifier Elimination for Unbounded Dense Linear Orders

Suppose we are given a formula \( P \) to eliminate quantifiers from. If we work bottom-up through the syntax tree of the formula, we can see that to eliminate quantifiers from the whole formula, it suffices to be able to eliminate quantifiers from formulas of the form \( ∃x. Q \), where \( Q \) contains no quantifiers. If we can do this, then we can also eliminate the quantifier in \( ∀x. Q \) by transforming it into \( ¬∃x. ¬Q \).

We can make the problem of quantifier elimination in \( ∃x. Q \) easier if we assume that it is of the form:

\[
∃x. (a_{1,1} ∧ ... ∧ a_{m_1,1}) ∨ ... ∨ (a_{1,n} ∧ ... ∧ a_{m,n})
\]

(1)

where each \( a_{i,j} \) is an atom of the form \( y = z \) or \( y ⊏ z \). If the formula is of this form, then it is equivalent to:

\[
(∃x. a_{1,1} ∧ ... ∧ a_{m_1,1}) ∨ ... ∨ (∃x. a_{1,n} ∧ ... ∧ a_{m,n})
\]

(2)

Now we only have to eliminate quantifiers from formulas of the form \( ∃x. a_1 ∧ ... ∧ a_n \).

But first, we have to be able to convert to the form in (1). This special form, where the formula is disjunction of conjunctions of atoms, is called Disjunctive Normal Form (DNF). We can convert any quantifier-free formula into DNF by working bottom-up:

- Atoms \( y = z \) and \( y ⊏ z \) are already in DNF.
- If \( P \) and \( Q \) are in DNF, then \( P ∨ Q \) is in DNF.
- If \( P \) and \( Q \) are in DNF, then:

\[
P ∧ Q ≡ \bigg( \bigvee_i C_i \bigg) ∧ \bigg( \bigvee_j D_j \bigg) ≡ \bigg( \bigvee_i \big( C_i ∨ \bigvee_j D_j \big) \bigg) ≡ \bigvee_{i,j} (C_i ∧ D_j)
\]

- If \( P \) is in DNF, then

\[
¬P ≡ ¬\bigg( \bigvee_i \bigwedge_{j_i} a_{i,j} \bigg) ≡ \bigwedge_i ∼\bigg( \bigwedge_{j_i} a_{i,j} \bigg) ≡ \bigwedge_i \bigvee_{j_i} ∼a_{i,j}
\]

which is not in DNF (it is conjunctive normal form), so use previous three points to convert to DNF.

The last point assumed that it was possible to negate atoms in the theory of dense linear orders. We can do this by using the following two consequences of the theory:

1. \((¬ x ⊏ y) \iff (y ⊏ x ∨ x = y)\)
2. \((¬ x = y) \iff (x ⊏ y ∨ y ⊏ x)\)

Below, you will be asked to prove these equivalences in Isabelle.

After conversion to DNF, we have reduced the problem of quantifier elimination to problems of the form shown in (2), where we only need to eliminate the quantifier in formulas of the form \( ∃x. a_1 ∧ ... ∧ a_n \).

We separate the atoms \( a_i \) into those that contain the variable \( x \) and those that don’t:

\[
(b_1 ∧ ... ∧ b_{n_1}) ∧ (∃x. c_1 ∧ ... ∧ c_{n_2})
\]

Now we examine the atoms \( c_i \):

- If \( x = x \) occurs in the \( c_i \), then just discard it because it is trivially true.
• If $x \sqsubseteq x$ occurs in the $c_i$, then the whole formula is equivalent to False, by the irreflexivity axiom of dense linear orders.

• If $x = y$ or $y = x$ occurs in the $c_i$, then we can eliminate $x$ by replacing it by $y$:
  \[(\exists x. (x = y) \land c_1 \land \ldots \land c_{n_2}) \equiv (c_1[y/x] \land \ldots \land c_{n_2}[y/x])\]

If none of these points applies, then we are left with a formula that only contains atoms of the form $s \sqsubseteq x$ or $x \sqsubseteq t$. So the whole formula is equivalent to one of the form:

\[\exists x. \left(\bigwedge_i s_i \sqsubseteq x \right) \land \left(\bigwedge_j x \sqsubseteq t_j\right)\]  \(\text{(3)}\)

where none of the $s_i$ or $t_j$ are $x$.

We can now make use of the following property of dense linear orders. If we have two finite sets of elements $L$ and $U$ such that every element in $L$ is less than every element in $U$, then, by denseness there exists an $x$ that is less than all the elements of $U$ and greater than all the elements of $L$. The reverse also holds, by linearity. In Isabelle notation:

\[
\begin{align*}
\text{finite } L; \text{finite } U & \implies (\forall y. y \in L \implies (\forall z. z \in U \implies y \sqsubseteq z)) \\
& \equiv (\exists x. (\forall c. c \in U \implies x \sqsubseteq c) \land (\forall d. d \in L \implies d \sqsubseteq x))
\end{align*}
\]

\(\text{(4)}\)

Below, you are asked to prove this statement in Isabelle.

Given the equivalence (4), we are justified in eliminating the quantifier from (3) to the equivalent formula:

\[\bigwedge_{i,j} s_i \sqsubseteq t_j\]

And we have completely eliminated the quantifier $\exists x$. Putting everything together, we can now eliminate quantifiers from arbitrary formulas containing the atomic propositions of the theory of dense linear orders, and therefore decide their validity in that theory.

The remainder of this coursework assignment asks you to prove in Isabelle the necessary results in the theory of dense linear orders that shows that the quantifier elimination procedure described here is correct. It is not neccessary to actually implement the procedure, though you might like to try it if you are interested.

5.2.1 Negation of Atoms (8 marks)

Prove the following two lemmas that show that it is possible to convert the negation of an atom in the theory of dense linear orders into a positive statement. Note that these are true in the theory of dense linear orders, without making any assumptions about boundedness or unboundedness.

You can use any of the proof methods listed in the Formal Cheat Sheet, including the simp method to prove these lemmas.

**lemma (in dense-linear-order) negate-lessthan :** $(\neg x \sqsubseteq y) \iff (y \sqsubseteq x \lor x = y)$  \(\text{(4 marks)}\)

**lemma (in dense-linear-order) negate-equality :** $(\neg x = y) \iff (x \sqsubseteq y \lor y \sqsubseteq x)$  \(\text{(4 marks)}\)
5.2.2 Existence of minima and maxima of finite sets (15 marks)

The proof of (4) relies on the fact that every finite set of elements in a dense linear order has a minimum and a maximum element. The following two lemma statements state these facts in Isabelle notation. You have to fill in the proofs of these lemmas. Note that the proof of the second lemma will be very similar to that for the first. Once you have done the first proof, you will be able to copy and paste the proof of the first lemma and tweak it slightly to prove the second one.

In order to prove these lemmas, you will need to do induction on finite sets. The `finite.induct` rule from Isabelle’s standard library of theorems will let you prove some predicate \( ?P \) by induction on the size of the finite set \( ?x \):

```
finite.induct : \[ finite ?x; \{ }; \\wedge A a. [finite A; ?P A] \implies ?P (insert a A)] \implies ?P ?x
```

You can use any of the proof methods listed in the Formal Cheat Sheet, including the `simp` method to prove these lemmas. The `simp` method has a number of built-in rules that can automatically prove trivial statements about whether or not values are members of sets.

**Lemma (in dense-linear-order) min-exists :**

\[
finite S \iff S \neq \{} \longrightarrow (\exists x. x \in S \land (\forall y. y \in S \longrightarrow x = y \lor x \sqsubseteq y))
\]

`oops`

**Lemma (in dense-linear-order) max-exists :**

\[
finite S \iff S \neq \{} \longrightarrow (\exists x. x \in S \land (\forall y. y \in S \longrightarrow x = y \lor y \sqsubseteq x))
\]

`oops`

5.2.3 The Quantifier Elimination lemma for unbounded DLOs (25 marks)

It now remains to prove the equivalence (4) that shows that the quantifier elimination procedure for unbounded DLOs preserves equivalence. You should fill in the proof of the following lemma. You will need to make use of the lemmas `min_exists` and `max_exists` from above. **Hint:** you will need to examine several different cases, depending on whether the sets \( U \) and \( L \) are empty or not. You can use the `excluded_middle` rule, or the `case_tac` method to split the proof into two cases.

You can use any of the proof methods listed in the Formal Cheat Sheet, including the `simp` method to prove this lemma.

**Lemma (in unbounded-dense-linear-order)**

```
qelim-lemma : 
[ finite L; finite U ] \implies 
(\forall y. y \in L \longrightarrow (\forall z. z \in U \longrightarrow y \sqsubseteq z)) = (\exists x. (\forall c. c \in U \longrightarrow x \sqsubseteq c) \land (\forall d. d \in L \longrightarrow d \sqsubseteq x))
```

`oops`

5.3 Part 2.3 : Bounded Dense Linear Orders (12 marks)

A bounded dense linear order is a linear that has a bottom and a top element. The theory of bounded dense linear orders is captured by the following locale declaration:

```
locale bounded-dense-linear-order = dense-linear-order +
  fixes top :: 'a and bottom :: 'a
  assumes bottom-is-bottom : (bottom = x) \lor (bottom \sqsubseteq x)
  and top-is-top : (top = x) \lor (x \sqsubseteq top)
```

**Question:** Describe how to adapt the quantifier elimination procedure described above to handle the theory of bounded dense linear order instead of unbounded ones. State and prove the necessary lemma(s) analogous to `qelim_lemma` above to show that your procedure is correct. Write an English description of your changes into a comment in the theory file you submit.

You can use any of the proof methods listed in the Formal Cheat Sheet, including the `simp` method to prove your lemmas.
6 Demonstrator Hours

The course lecturer, Robert Atkey, will be available to give advice in room 5.05 of Appleton Tower each Thursday and Friday between 2pm and 4pm. You may also send questions directly to bob.atkey@ed.ac.uk or post questions to the course mailing list at ar-students@inf.ed.ac.uk.

7 Submission

By 2pm on Friday 27th February 2015, you must submit your theory file by typing the following commands at the terminal:

```
submit 1 ar ARCoursework1.thy
```

Late coursework will be penalised in accordance with the Informatics standard policy. Please consult your course guide for specific information about this. Also note that, while we encourage students to discuss the practical among themselves, we take plagiarism seriously and any such case will be treated appropriately. Please consult your student guide for more information about this matter.