Universität Innsbruck
Institut für Informatik
Prof. Dr. Clemens Ballarin

SS 2008
Exercise Sheet 2
Due 17 April 2008

Automatic Deduction — Introduction to Isabelle

LVA 703522

1 Type Inference

▷ Is the term $\lambda x.\, x\, x$ type correct? Justify your answer.

No. The only applicable typing rule is that for abstraction:

$$\frac{\Gamma[x \mapsto \tau_1] \vdash x\, x :: \tau_2}{\Gamma \vdash \lambda x.\, x\, x :: \tau_1 \Rightarrow \tau_2}.$$

Next, the only applicable typing rule is that for application:

$$\frac{\Gamma[x \mapsto \tau_1] \vdash x :: \sigma \Rightarrow \tau_2 \qquad \Gamma[x \mapsto \tau_1] \vdash x :: \sigma}{\Gamma[x \mapsto \tau_1] \vdash x\, x :: \tau_2}.$$

Next, the only applicable typing rule (on both sides) is that for variables. This rule is only applicable on both sides however if $\Gamma[x \mapsto \tau_1](x) = \sigma \Rightarrow \tau_2$ *and* $\Gamma[x \mapsto \tau_1](x) = \sigma$, i.e. $\sigma = \sigma \Rightarrow \tau_2$ for some types σ and τ_2, which is clearly not possible.

2 Natural Deduction

We will use the calculus of natural deduction to prove some lemmas of propositional and predicate logic in Isabelle.

2.1 Propositional Logic

- Only use these rules in the proofs:

```
notI: (P ⟹ False) ⟹ ¬ P
notE: ⟦¬ P; P⟧ ⟹ R
conjI: ⟦P; Q⟧ ⟹ P ∧ Q
conjE: ⟦P ∧ Q; ⟦P; Q⟧ ⟹ R⟧ ⟹ R
disjI1: P ⟹ P ∨ Q
disjI2: Q ⟹ P ∨ Q
disjE: ⟦P ∨ Q; P ⟹ R; Q ⟹ R⟧ ⟹ R
impI: (P ⟹ Q) ⟹ P ⟶ Q
```

```
impE: ⟦P ⟶ Q; P; Q ⟹ R⟧ ⟹ R
mp: ⟦P ⟶ Q; P⟧ ⟹ Q
iffI: ⟦P ⟹ Q; Q ⟹ P⟧ ⟹ P = Q
iffE: ⟦P = Q; ⟦P ⟶ Q; Q ⟶ P⟧ ⟹ R⟧ ⟹ R
```

- Only use the methods `(rule` r`)`, `(erule` r`)` and `assumption`, where r is one of the rules given above.

▷ Prove the following lemmas in Isabelle.

```
lemma "((A ∨ B) ∨ C) ⟶ A ∨ (B ∨ C)"
  apply (rule impI)
  apply (erule disjE)
   apply (erule disjE)
    apply (rule disjI1)
    apply assumption
   apply (rule disjI2)
   apply (rule disjI1)
   apply assumption
  apply (rule disjI2)
  apply (rule disjI2)
  apply assumption
  done

lemma "(A ∨ A) = (A ∧ A)"
  apply (rule iffI)
   apply (erule disjE)
    apply (rule conjI)
     apply assumption
    apply assumption
   apply (rule conjI)
    apply assumption
   apply assumption
  apply (erule conjE)
  apply (rule disjI1)
  apply assumption
  done

lemma "(D ⟶ A) ⟶ (A ⟶ (B ∧ C)) ⟶ (B ⟶ ¬ C) ⟶ ¬ D"
  apply (rule impI)+
  apply (rule notI)
  apply (erule impE)
   apply assumption
  apply (erule impE)
   apply assumption
  apply (erule conjE)
  apply (erule impE)
   apply assumption
  apply (erule notE)
```

```
  apply assumption
  done

lemma "(A ⟶ ¬ B) = (B ⟶ ¬ A)"
  apply (rule iffI)
   apply (rule impI)
   apply (rule notI)
   apply (erule impE)
    apply assumption
   apply (erule notE)
   apply assumption
  apply (rule impI)
  apply (rule notI)
  apply (erule impE)
   apply assumption
  apply (erule notE)
  apply assumption
  done
```

2.2 Pierce's law

Prove Pierce's law `((A ⟶ B) ⟶ A) ⟶ A`.

▷ First give a paper proof using case distinction and/or proof by contradiction.

A proof by case distinction works as follows. Suppose `A`. Then the proposition holds, because this is the consequent of the the outermost implication. Otherwise, if we suppose `B` the proposition evaluates to true, and likewise if we suppose `¬ B`.

▷ Now give a proof in Isabelle. In addition to the rules and methods from Exercise 2.1, you may use `(case_tac` P`)` (where P is a Boolean expression, e.g. a variable) for case distinctions, `back` to select a different unifier when applying a method, and the theorem `classical: (¬ P ⟹ P) ⟹ P`.

```
lemma Pierce: "((A ⟶ B) ⟶ A) ⟶ A"
  apply (case_tac "A")
   apply (rule impI)
   apply assumption
  — case ¬ A
  apply (case_tac "B")
   apply (rule impI)
   apply (erule impE)
    apply (rule impI)
    apply assumption
   apply assumption
  — case ¬A; ¬B
  apply (rule impI)
```

```
  apply (erule impE)
   apply (rule impI)
   apply (erule notE)
   apply assumption
  apply assumption
  done
```

2.3 Predicate Logic

We are again talking about proofs in the calculus of Natural Deduction. In addition to the theorems given in the exercise "Propositional Logic" (Exercises 2), you may now also use

```
exI: P x ⟹ ∃x. P x
exE: ⟦∃x. P x; ⋀x. P x ⟹ Q⟧ ⟹ Q
allI: (⋀x. P x) ⟹ ∀x. P x
allE: ⟦∀x. P x; P x ⟹ R⟧ ⟹ R
```

▷ Give a proof of the following propositions or an argument why the formula is not valid:

```
lemma "(∃x. ∀y. P x y) ⟶ (∀y. ∃x. P x y)"
  apply (rule impI)
  apply (rule allI)
  apply (erule exE)
  apply (rule exI)
  apply (erule allE)
  apply assumption
  done

lemma "(∀x. P x ⟶ Q) = ((∃x. P x) ⟶ Q)"
  apply (rule iffI)
   apply (rule impI)
   apply (erule exE)
   apply (erule allE)
   apply (erule impE)
    apply assumption
   apply assumption
  apply (rule allI)
  apply (rule impI)
  apply (erule impE)
   apply (rule exI)
   apply assumption
  apply assumption
  done

lemma "((∃ x. P x) ∧ (∃ x. Q x)) = (∃ x. (P x ∧ Q x))"
```

```
  refute
  oops
```

A possible counterexample is: `P = even`, `Q = odd`, interpreted over the natural numbers.

```
lemma "((∃ x. P x) ∨ (∃ x. Q x)) = (∃ x. (P x ∨ Q x))"
  apply (rule iffI)
   apply (erule disjE)
    apply (erule exE)
    apply (rule exI)
    apply (rule disjI1)
    apply assumption
   apply (erule exE)
   apply (rule exI)
   apply (rule disjI2)
   apply assumption
  apply (erule exE)
  apply (erule disjE)
   apply (rule disjI1)
   apply (rule exI)
   apply assumption
  apply (rule disjI2)
  apply (rule exI)
  apply assumption
  done
```

The following lemma also requires `classical`: `(¬ P ⟹ P) ⟹ P` (or an equivalent theorem) in order to be proved.

```
lemma "(¬ (∀ x. P x)) = (∃ x. ¬ P x)"
  apply (rule iffI)
   apply (rule classical)
   apply (erule notE)
   apply (rule allI)
   apply (rule classical)
   apply (erule notE)
   apply (rule exI)
   apply assumption
  apply (erule exE)
  apply (rule notI)
  apply (erule allE)
  apply (erule notE)
  apply assumption
  done
```

2.4 A Riddle: Rich Grandfather

▷ First prove the following formula, which is valid in classical predicate logic, informally with pen and paper. Use case distinctions and/or proof by contradiction.

If every poor man has a rich father,
then there is a rich man who has a rich grandfather.

theorem
```
"∀x. ¬ rich x ⟶ rich (father x) ⟹
∃x. rich (father (father x)) ∧ rich x"
```

Proof
(1) We first show: `∃x. rich x`.
Proof by contradiction.
Assume `¬ (∃x. rich x)`.
Then `∀x. ¬ rich x`.
We consider an arbitrary `y` with `¬ rich y`.
Then `rich (father y)`.
(2) Now we show the theorem.
Proof by cases.
Case 1: `rich (father (father x))`.
The rich man who has a rich grandfather is `x`. We are done.
Case 2: `¬ rich (father (father x))`.
Then `rich (father (father (father x)))`.
Also `rich (father x)`,
because otherwise `rich (father (father x))`.
The rich man who has a rich grandfather is `father x`.
qed

▷ Now prove the formula in Isabelle using a sequence of rule applications (i.e. only using the methods `rule`, `erule` and `assumption`). In addition to the theorems that were allowed in the exercise "Predicate Logic", you may now also use `classical: (¬ P ⟹ P) ⟹ P`.

Since we are not allowed to use lemmas, (1) will show up as two copies of the same proof script in the proof.

theorem
```
"∀x. ¬ rich x ⟶ rich (father x) ⟹
∃x. rich (father (father x)) ∧ rich x"
apply (rule classical)
apply (rule exI)
apply (rule conjI)

— Show rich (father (father x2))
  apply (rule classical)
— Assume ¬ rich (father (father x2)), case 2
  apply (rule allE) apply assumption
  — use rule rather then erule in order not to delete the assumption, it is needed a second time
```

```
  apply (erule impE) apply assumption
— Now we have rich (father (father (father x2)))
  apply (erule notE)
— Show ∃x. rich (father (father x)) ∧ rich x
  apply (rule exI)
  apply (rule conjI) apply assumption
— Show rich (father x2)
  apply (rule classical)
  apply (erule allE)
  apply (erule notE)
  apply (erule impE) apply assumption
  apply assumption

— Show rich x2
apply (rule classical)
— Assume ¬ rich x2, case 1
apply (rule allE) apply assumption
apply (erule impE) apply assumption
— Now we have rich (father x2)
apply (erule notE)
— Show ∃x. rich (father (father x)) ∧ rich x
apply (rule exI)
apply (rule conjI) apply assumption
— Show rich x41
apply (rule classical)
apply (erule allE)
apply (erule notE)
apply (erule impE) apply assumption
apply assumption
done
```