universität innsbruck

Interactive Theorem Proving | SS 2019 | LVA 703640

EXAM 1 | June 27, 2019

This exam consists of **five** exercises on topics as indicated. Each item is worth 10 points. There are 100 points in total, excluding the bonus exercises (marked by *). The questions may refer to the Coq code as on the separate sheet. You need at least 50 points (combined with the points for the presentation, if this applies) to pass.

1 Inhabitation and provability

Let ϕ be the formula $(p \to q) \to (\neg q \to \neg p)$ and ψ be $(\neg p \to \neg q) \to (q \to p)$; see the Coq code.

(a) Explain what the Curry–Howard correspondence is, give the typed λ-term for the Coq proof of ϕ, and give the corresponding proof of ϕ in intuitionistic propositional logic, marking applied inference rules as introduction- or elimination-rule as appropriate. [10]

(b) Show that the formula ψ is not provable in intuitionistic propositional logic. Explain what will happen in Coq when applying the tactic `tauto` to it. Will it succeed, fail, or not terminate, or ...? Why? [10]

2 Typability and reduction of λ-terms

Let for arbitrary natural n the numeral c_n be $\lambda fx.f^n(x)$. For example, $c_0 = \lambda fx.x$ and $c_2 = \lambda fx.f(fx)$.

(a) Check for each of c_0, c_1 and c_2c_2 whether it is typable in simply typed λ-calculus, and if so, give its principal type. [10]

(b) Reduce c_2c_2 to β-normal form. You don't need to give all β-steps, but at least two of them. If α-conversion is needed, give a(t least one) step where it was used. [10]

(c)* Suppose $M \to_\beta N$ and N is typable in simply typed λ-calculus (Curry), is M then typable as well? If so, explain why. If not, give a counterexample. [10]

3 Polymorphic λ-calculus $\lambda 2$ (System F)

See the Coq code for a so-called impredicative encoding Nat of the natural numbers in polymorphic λ-calculus. To illustrate that this is a reasonable representation,[1] a predecessor function pred is defined on natural numbers (terms of type Nat).

(a) Explain in what way polymorphic λ-calculus is a dependently typed λ-calculus (e.g. what kind of dependency does it allow; you may but need not use the notion of PTS to explain this), and give an example where this dependency is used (involves a reduction step) when computing the predecessor of some number. [10]

(b) If we would replace `Prop` in the definition of the type Nat by `Type`, then Coq fails when processing the remaining code. Explain why it fails and where (on what code) it fails. [10]

(c)* Give the sorts, axioms (pairs) and rules (triples) used for representing the calculus of constructions as a PTS, and show that the type Nat (as in the Coq code) is derivable in it. [10]

4 λP and PTSs

(a) Explain in what way λP is a dependently typed λ-calculus (e.g. what kind of dependency does it allow; you may but need not use the notion of PTS to explain this), and give an example of a term M that is in λP but not in $\lambda 2$; show that M indeed is in λP and argue why it is not in $\lambda 2$. [10]

(b) Simply typed λ-calculus can be seen as a PTS. Give the sorts, axioms, and rules needed for this, and explain how that there are only 3 inference rules in simply typed λ-calculus can be reconciled with that PTSs have (many) more inference rules. [10]

5 Inductive types

Let a positive natural number (Pos) be inductively defined to be either 1, or 2 times a Pos, or 1 plus 2 times a Pos; see the Coq code.

(a) Explain how the number 6 is represented as a Pos, that this representation is unique, and give the induction and recursion principles corresponding to Pos. (You may express this in the way most convenient to you.) [10]

(b) Consider Pos2nat as given by the Coq script. Argue that Pos2nat defines a *total function*, and how/why Coq accepts it as such. [10]

(c)* How would you define the natural numbers including 0 based on Pos (using Pos as a black box)? Discuss the (dis)advantages of each of the representations of natural numbers used above: as certain λ-terms ($\lambda f x.f^n(x)$ resp. $\lambda x f.f^n(x)$) and inductively (nat resp. based on Pos). [10]

$$\text{(axiom)} \quad \langle\rangle \vdash s_1 : s_2 \qquad \text{if } (s_1, s_2) \in \mathcal{A}$$

$$\text{(start)} \quad \frac{\Gamma \vdash A : s}{\Gamma, x{:}A \vdash x : A} \qquad \text{if } x \notin \mathsf{dom}(\Gamma)$$

$$\text{(weakening)} \quad \frac{\Gamma \vdash M : A \quad \Gamma \vdash B : s}{\Gamma, x{:}B \vdash M : A} \qquad \text{if } x \notin \mathsf{dom}(\Gamma)$$

$$\text{(product)} \quad \frac{\Gamma \vdash A : s_1 \quad \Gamma, x{:}A \vdash B : s_2}{\Gamma \vdash (\Pi x{:}A.\, B) : s_3} \qquad \text{if } (s_1, s_2, s_3) \in \mathcal{R}$$

$$\text{(application)} \quad \frac{\Gamma \vdash M : (\Pi x{:}A.\, B) \quad \Gamma \vdash N : A}{\Gamma \vdash MN : B[x := N]}$$

$$\text{(abstraction)} \quad \frac{\Gamma, x{:}A \vdash M : B \quad \Gamma \vdash (\Pi x{:}A.\, B) : s}{\Gamma \vdash \lambda x{:}A.M : (\Pi x{:}A.\, B)}$$

$$\text{(conversion)} \quad \frac{\Gamma \vdash M : A \quad \Gamma \vdash B : s}{\Gamma \vdash M : B} \qquad \text{if } A =_\beta B$$

[1] Predecessor cannot be defined on the Church numerals in the simply typed λ-calculus.

```
(* exercise 1 *)

Hypothesis p q:Prop.

Lemma onea : (p -> q) -> (~q -> ~p).
Proof.
unfold not.
intros x y z.
apply y.
apply x.
assumption.
Defined.

Print onea. (* 1a: which lambda-term is printed here? *)

Lemma oneb : forall p q:Prop, (~p -> ~q) -> (q -> p).
Proof.
(* 1b what would be the result of the tauto-tactic here? *)
Admitted.

(* exercise 3 *)

Definition Nat := forall A:Prop, A -> (A -> A) -> A.
(* 3b where and how does Coq fail below, if we would change Prop to Type here in Nat *)

Definition Unary := Nat -> Nat.
Definition Binary := Nat -> Nat -> Nat.
Definition Pair := Binary -> Nat.

Definition succ : Unary :=
  fun n A x f => n A (f x) f.
Definition pair : Nat -> Nat -> Pair :=
  fun n m z => z n m.
Definition fst : Binary :=
  fun n m => n.
Definition snd : Binary :=
  fun n m => m.
Definition zero : Nat :=
  fun _ x _ => x.
Definition next : Pair -> Pair :=
  fun p z => z (succ (p fst)) (p fst).

Definition pred : Unary :=
  fun n => n Pair (pair zero zero) next snd.

Definition three : Nat :=
  fun _ x f => f (f (f x)).

(* Exercise 5 *)

Inductive Pos : Set :=
  | xI : Pos -> Pos (* representing 2x+1 *)
  | xO : Pos -> Pos (* representing 2x *)
  | xH : Pos. (* representing 1 *)

Check Pos_ind. (* 5a, induction principle *)
Check Pos_rec. (* 5a, recursion principle *)

Fixpoint Pos2nat (p:Pos) : nat := match p with
| xI p' => S (Pos2nat p' + Pos2nat p')
| xO p' => Pos2nat p' + Pos2nat p'
| xH => 1
end. (* 5b, why does Coq accept this as a definition of a total function? *)
```

Summary The exam covers the material presented during the lectures. The available slides and course notes Logical Verification (Femke van Raamsdonk) serve as a basis for the exam. The other available material (only) has a supporting role, providing more in depth coverage of topics only touched upon in the material mentioned above.

Course Notes All the theory in the Course Notes was covered and should be known, including when and how to apply the presented algorithms. Only *passive* knowledge of the proof-assistant Coq and its syntax (for statements, tactics etc.) is required: You should be able to understand statements and their proofs (using language as covered in the Course Notes) .

Slides Per week:

1. Know what ITP is, its distinction with CA, Model Checkers, ATP, and examples of proof-assistants. Need for foundations; paradoxes (Russell, Curry). ITPs with foundations in set theory (Mizar), type theory (Isabelle, Coq).

2. Church's simple theory of types. Curry–Howard (CH) correspondence between simply typed λ-calculus and (intuitionistic) implication propositional logic, extended to an isomorphism: also relating inhabitants to proofs and β-reduction to proof normalisation (reducing introduction followed by elimination of the same connective). Various desirable properties such as decidability of type checking, based on meta-theoretic properties of β-reduction such as confluence, termination, and subject-reduction. Proving termination by proving the (stronger) strong computability property (extra material).

3. Curry vs. Church typing, with forgetful map from right to left and unification algorithm from left to right giving the principal (i.e. most general) type. The inhabitation problem and its algorithmic solution (and complexity): syntactically (via normal forms; see Course Notes) and semantically (via CH and Kripke Semantics of IPL; see following week). Connection and some differences between classical and intuitionistic propositional logic.

4. Kripke semantics of modal logics in general (e.g. temporal logics as in LICS), and of intuitionistic propositional logic in particular (see additional material for this week, for precise statements). The disjunction property and its proof using Kripke semantics.

5. Lambda P (P for predicate), as generalization of simply typed lambda calculus, with proof system and its properties (e.g. type inhabitation becomes undecidable), and examples naturally expressed in it. (minimal) predicate logic via CH. Representation of Lambda P in Coq.

6. Some further derivations in Lambda P.

The order of a logic. Lambda 2 (2 for 2nd-order), as (another) generalization of simply typed lambda calculus, with proof system and its properties, and examples naturally expressed in it. 2nd-order propositional logic via CH; connection to QBF making classical prop2 decidable (but intuitionistic prop2 is not).

Towards a uniform presentation for inference systems for lambda calculi, parametrised by 'abstraction power'.

7. Impredicativity and impredicative definitions of logical connectives and various data types.

   Data types by inductive definitions, with construction (cases) as introduction and pattern matching as elimination. Their representation and manipulation in Coq.

   Program extraction. From proof of property to program for the corresponding specification, with Coq examples.

8. Inductively defined data types, their induction/recursion principles and representation in Coq, with examples.

   Well-founded induction as structural induction via accessibility, and structural induction as well-founded induction via substructure relation.

9. Uniform presentation of dependently typed lambda calculi as PTSs, with simply typed, lambda P, lambda 2, and the calculus of constructions (the system underlying Coq, minus inductive types), by choosing different parameters: sorts, axioms, and rules. The meta-theoretic propertie of PTSs.

10. Discussion of various choices one can make to represent concepts in an ITP (e.g. the representations using impredicative encoding, and using inductive types as mentioned above, or the various representations of natural numbers (unary, binary, Church-numerals), etc.). Deep vs. shallow embedding (e.g. representing classical propositional logic as Coq propositions, or as a data type with operations on them).

    Discussion of a Coq-theory representing Russell's paradox exploiting the absence of a hierarchy among Types brought about by the (unsafe) type-in-type option. (Note: the representation of sets employed, was made to exploit this; in standard usage of Coq other representations are used, e.g. Ensembles.)

To Know

- paradoxes: Curry's and Russell's;
- lambda-calculus, and representation of basic data-types (nats,booleans) in it (Church numerals, projections); what it means to represent;
- simply typed λ-calculus and its inference rules;

- intuitionistic first-order logic and its inference rules;
- difference between intuitionistic and classical logic;
- BHK (Brouwer–Heyting–Kolmogorov) interpretation;
- Kripke semantics of (propositional) intuitionistic logic;
- introduction vs. elimination;
- inductive types and their induction and recursion principles;
- CH (Curry–Howard) correspondence;
- lambda P
- lambda 2
- PTSs with instances Barendregt's λ-cube and Calculus of Constructions (CoC);
- the role the sort-, axiom-, and rule-parameters play in PTSs, and their concrete instantiations for obtaining simply typed lambda-calculus, lambda P, lambda 2, CoC.
- dependent types, and the rendering of dependencies in PTSs;
- the type-inhabitation, type-checking, type-inference problems;
- basic meta-theoretic properties of β-reduction: termination, confluence, subject-reduction.
- (im)predicativity;
- shallow/deep embedding;
- correctness by construction (program extraction) vs. correctness by enriching (Hoare logic);
- basic Coq syntax, for doing logical reasoning and reasoning about inductive data types.

To be able to do

- β-reduction (with α-conversion when needed);
- proof normalisation;
- classify an inference rule as being an introduction/elimination rule; recognise β-redexes;
- do the Curry–Howard transformation for types/β-reduction/... into propositions/proof normalisation/... (only the propositional case)

- checking (non-)inhabitation for simple types via normal form algorithm and Kripke models;
- simple type inference; finding principal type via Hindley–Milner based on unification/constraint solving;
- type checking; using the inference rules to establish that a term does have the type given, in simply typed lambda-calculus, lambda P, lambda 2, and PTSs (in the latter case: given the inference system).
- translate Coq statements/proofs to the above notions.