A Semantical Proof of Consistency for Minimal Propositional Logic in Coq

Felipe Sasdelli[1], Maycon Amaro[1], Rodrigo Ribeiro[1]

[1]Departamento de Computação – Universidade Federal de Ouro Preto (UFOP)
35400-000 – Ouro Preto – MG – Brazil

{felipe.sasdelli,maycon.amaro}@aluno.ufop.edu.br, rodrigo.ribeiro@ufop.edu.br

Abstract. *Consistency is a key property of any logical system. However, proofs of consistency usually rely on heavy proof theory notions like admissibility of cut. A more semantics based approach to consistency proofs is to explore the correspondence between a logics and its relation with the evaluation in a λ-calculus, known as Curry-Howard isomorphism. In this work, we describe a Coq formalization of consistency of minimal propositional logics using this semantics approach and compare it with the traditional approaches used in proof theory.*

1. Introduction

A crucial property of a logical system is consistency, which states that it does not entails a contradiction. Basically, consistency implies that not all formulas are provable. While having a simple motivation, consistency proofs rely on the well-known admissibility of cut property, which has a complex inductive proof. Gentzen, in his seminal work, gives the first consistency proof of logic by introducing an auxiliar formalism, the sequent calculus, in which consistency is trivial. Next, Gentzen showed that natural deduction system is equivalent to his sequent calculus extended with an additional rule: the cut rule. The final (and hardest) piece of Gentzen's proof is to show that the cut rule is redundant, i.e., it is admissible. As a consequence, we know something stronger: all propositions provable in the natural deduction system are also provable in the sequent calculus without cut. Since we know that the sequent calculus is consistent, we hence also know that the natural deduction calculus is [Negri et al. 2001].

However, proving the admissibility of cut is not easy, even for simple logics. Proofs of admissibility need nested inductions and we need to be really careful to ensure a decreasing measure on each use of the inductive hypothesis. Such proofs have a heavy syntactic flavor since they recursively manipulate proof tree structures to eliminate cuts. A more semantic based approach relies on interpreting logics as its underlying λ-calculus and prove consistency by using its computation machinery. In this work, we report the Coq formalization of these two approaches and advocate the use of the latter since it result on easy to follow proofs. We organize this work as follows: Section 2 present basic definitions about the logic considered and Section 3 describe the semantics of our logic objects and its consistency proof. Section 4 presents a brief comparision between two consistency proofs and concludes.

The complete formalization was verified using Coq version 8.10.2 and its available on-line [Sasdelli et al. 2020]. For space reasons, we rely on reader's intuition to explain Coq code fragments. Good introductions to Coq are avaliable elsewhere [Chlipala 2013].

2. Basic Definitions

First, we consider formulas of a minimal fragment of propositional logics which is formed only by the constant *falsum* ($\bot$) and logic implication ($\supset$). Following common practice, we denote contexts by a list of formulas. The following Coq snipetts declare these concepts.

$\texttt{Inductive}\ \alpha : \texttt{Set} :=$
$\mid$ *Falsum* $: \alpha$
$\mid$ *Implies* $: \alpha \to \alpha \to \alpha.$
$\texttt{Definition}\ \Gamma :=$ *list* $\alpha.$

While types for formulas (α) and contexs (Γ) have an immediate interpretation, the previous types miss an important part of propositional logic: variables. We represent variables by an inductive judgement which states the membership of a formula in a context.

We let notation $\alpha \in \Gamma$ denote an inductive predicate that states that a formula α is an element of context Γ. The rules for variable judgement and its Coq implementation are presented below.

$\texttt{Inductive}$ *var* $: \Gamma \to \alpha \to \texttt{Type} :=$
$\mid$ *Here* : $\forall$ *G p*, *var* (*p* :: *G*) *p*
$\mid$ *There* : $\forall$ *G p p'*, *var G p* $\to$ *var* (*p'* :: *G*) *p*.

$$\frac{}{\alpha \in (\alpha :: \Gamma)}\ \{Here\}$$

$$\frac{\alpha \in \Gamma}{\alpha \in (\beta :: \Gamma)}\ \{There\}$$

The first constructor of type *var* specifies that a formula α is in the context $\alpha :: \Gamma$ and the constructor *There* specifies that if a formula α is in Γ then we have $\alpha \in (\beta :: \Gamma)$, for any formula β.

Using the previous definitions, we can implement natural deduction rules for our minimal logic, as presented below.

$\texttt{Inductive}$ *nd* $: \Gamma \to \alpha \to \texttt{Type} :=$
$\mid$ *Id* : $\forall$ *G p*,
  var G p $\to$
  nd G p
$\mid$ *ExFalsum* : $\forall$ *G p*,
  nd G Falsum $\to$
  nd G p
$\mid$ *Implies_I* : $\forall$ *G p p'*,
  nd (*p'* :: *G*) *p* $\to$
  nd G (*Implies p' p*)
$\mid$ *Implies_E* : $\forall$ *G p p'*,
  nd G (*Implies p' p*) $\to$
  nd G p' $\to$
  nd G p.

$$\frac{x \in \Gamma}{\Gamma \vdash x}\ \{Id\}$$

$$\frac{\Gamma \vdash \bot}{\Gamma \vdash \alpha}\ \{Ex\}$$

$$\frac{\Gamma \cup \{\alpha\} \vdash \beta}{\Gamma \vdash \alpha \supset \beta}\ \{\supset\!-I\}$$

$$\frac{\Gamma \vdash \alpha \supset \beta \quad \Gamma \vdash \alpha}{\Gamma \vdash \beta}\ \{\supset\!-E\}$$

The first rule (*Id*) stabilishes that any formula in the context is provable and rule *ExFalsum* defines the principle *ex-falsum quod libet* which allow us to prove any formula if we have a deduction of *Falsum*. Rule *Implies_I* specifies that from a deduction of a formula *p* from a context *p'* :: *G*, *nd* (*p'* :: *G*) *p*, we can prove the implication *Implies p' p*. The last rule, *Implies_E*, represents the well-known *modus-ponens*, which allows us to deduce a formula *p* from deductions of *Implies p' p* and *p'*.

Next section uses the relation between logics and λ-calculus and its evaluation to prove the consistency of minimal logic.

3. Semantics and Consistency

We prove the consistency of logics by exploring its correspondence with the simply typed λ-calculus. We do this by implementing in Coq a well-known idea [Augustsson and Carlsson 1999] for implementing denotational semantics for λ-term in type theory based proof assistants.

We define the denotation of a formula by recursion on its structure. The idea is to associate the empty type (*False*) with the formula *Falsum* and a function type with formula *Implies p1 p2*, as presented next.

```
Program Fixpoint sem_form (p : α) : Type :=
  match p with
  | Falsum ⇒ False | Implies p1 p2 ⇒ sem_form p1 → sem_form p2
  end.
```

Using *sem_form* function, we can define context semantics as tuples of formula semantics as follows:

```
Program Fixpoint sem_ctx (G : Γ) : Type :=
  match G with
  | ∅ ⇒ unit | (t :: G') ⇒ sem_form t × sem_ctx G'
  end.
```

Function *sem_ctx* recurses over the structure of the input context building right-nested tuple ending with the Coq *unit* type, which is a type with a unique element. Since context are mapped intro tuples, variables must be mapped into projections on such tuples. This would allow us to retrieve the value associated with a variable in a context.

```
Program Fixpoint sem_var {G p}(v : var G p) : sem_ctx G → sem_form p :=
    match v with
    | Here ⇒ fun env ⇒ fst env | There v' ⇒ fun env ⇒ sem_var v' (snd env)
    end.
```

Function *sem_var* receives a variable (value of type *var G p*) and a semantics of a context (a value of type *sem_ctx G*) and returns the value of the formula represented by such variable. Whenever the variable is built using constructor *Here*, we just return the first component of the input context semantics and when we have the constructor *There* we just call *sem_var* recursively.

Our next step is to define the semantics of natural deduction proofs. The semantics of proofs is implemented by function *sem_nat_ded* which maps proofs (values of type *nat_ded G p*) and context semantics (values of type *sem_ctx G*) to the value of input proof conclusion (type *sem_form p*). The first case specifies that the semantics of an identity rule proof (constructor *Id*) is just retrieving the value of the underlying variable in the context semantics by calling function *sem_var*. Second case deals with *ExFalsum* rule: we recurse over the proof object *Hf* which will produces a Coq object of type *False*, which is empty and so we can finish the definition with an empty pattern match. Semantics of implication introduction (*Implies_I*) simply recurses on the subderivation *Hp* using an extended context (*v'* , *env*). Finally, we define the semantics of implication elimination as simply function application of the results of the recursive call on its two subderivations.

```
Program Fixpoint sem_nat_ded {G p}(H : nat_ded G p)
  : sem_ctx G → sem_form p :=
  match H with
  | Id v ⇒ fun env ⇒ sem_var v env
  | ExFalsum Hf ⇒ fun env ⇒
      match sem_nat_ded Hf env with
      end
  | Implies_I Hp ⇒ fun env v' ⇒ sem_nat_ded Hp (v' , env)
  | Implies_E Hp Ha ⇒ fun env ⇒ (sem_nat_ded Hp env) (sem_nat_ded Ha env)
  end.
```

Using all thoses previously defined pieces we can prove the consistency of our little natural deduction system merely by showing that it should not be the case that we have a proof of *Falsum* using the empty set of assumptions. We can proof such fact by exhbiting a term of type *nat_ded* $\emptyset$ *Falsum* $\rightarrow$ *False*[1], which is trivially done by using function *sem_nat_ded*.

```
Theorem consistency : nat_ded ∅ Falsum → False := fun p ⇒ sem_nat_ded p tt.
```

4. Conclusion

In this work we briefly describe a Coq formalization of a semantics based consistency proof for minimal propositional logic. The complete proof is only 85 lines long and only use of some basic dependently typed programming features of Coq. We also formalize the consistency of this simple logic in Coq using Gentzen's admissibility of cut approach which resulted in around 270 lines of code and uses some extra proof tactics libraries. As future work, we intend to extend the current formalization to full propositional logic and also other formalisms like Hilbert systems and analytic tableaux [Smullyan 1995].

References

Augustsson, L. and Carlsson, M. (1999). An exercise in dependent types: A well-typed interpreter. In *In Workshop on Dependent Types in Programming, Gothenburg*.

Chlipala, A. (2013). *Certified Programming with Dependent Types: A Pragmatic Introduction to the Coq Proof Assistant*. The MIT Press.

Negri, S., von Plato, J., and Ranta, A. (2001). *Structural Proof Theory*. Cambridge University Press.

Sasdelli, F., Amaro, M., and Ribeiro, R. (2020). A semantical proof of consistency for minimal propositional logic in coq. https://github.com/rodrigogribeiro/consistency-coq.

Smullyan, R. (1995). *First-order Logic*. Dover books on advanced mathematics. Dover.

[1]Here we use the fact that $\neg\alpha$ is equivalent to $\alpha \supset \bot$.

A. Consistency Proof Based on Admissibility

A.1. Overview

Due to limited space, we include a brief description of our formalization of consistency using Gentzen's cut-elimination method in this appendix. First, we describe types for describing the syntax of formulas, natural deduction proofs and sequent calculus proofs. Next, we provide proof sketchs of weakening for sequent calculus, equivalence of sequent-calculus with

A.2. Basic Definitions

Unlike in our semantics based proof, we need to represent variables in formula syntax. We chose to represent variable identifiers as natural numbers.

```
Definition var := nat.

Inductive form : Type :=
| Falsum  : form
| Var     : var → form
| Implies : form → form → form.
```

Constructors `Falsum` e `Implies` represent the false constant and logical implication, respectively. Using the defined syntax, we can represent natural deduction and sequent calculus rules as inductive types as presented next.

```
Inductive nd : ctx → form → Prop :=
| Id_Nd G a
  : a el G →
    nd G a
| ExFalsum G a
  : nd G Falsum →
    nd G a
| Implies_I G a b
  : nd (a :: G) b →
    nd G (Implies a b)
| Implies_E G a b
  : nd G (Implies a b) →
    nd G a → nd G b.
```

$$\frac{x \in \Gamma}{\Gamma \vdash x}\,\{Id\}$$

$$\frac{\Gamma \vdash \bot}{\Gamma \vdash \alpha}\,\{Ex\}$$

$$\frac{\Gamma \cup \{\alpha\} \vdash \beta}{\Gamma \vdash \alpha \supset \beta}\,\{\supset\!-I\}$$

$$\frac{\Gamma \vdash \alpha \supset \beta \quad \Gamma \vdash \alpha}{\Gamma \vdash \beta}\,\{\supset\!-E\}$$

The only change on how we represent the natural deduction proof system is in the rule for variables. We use the Coq library boolean list membership predicate (`member`), which fits better for proof automation. In order to simplify the task of writing code that uses this predicate, we defined notation `a el G` which means `member a G`. The other constructors of type `nd` are identitical to the ones presented in Section 3.

Next, we the sequent calculus formulation for our minimal logic. The only difference with the natural deduction is in one rule for implication. The sequent calculus rule counter-part for implication elimination is called implication left rule, which states that we can conclude any formula γ in a context Γ if we have that: 1) $\alpha \supset \beta \in \Gamma$; 2)

$\Gamma \Rightarrow \alpha$ and 3) $\Gamma \cup \{\beta\} \Rightarrow \gamma$. The rules for the sequent-calculus an its correspondent Coq implementation are presented next.

```
Inductive sc : ctx → form → Prop :=
| Id G a
  : (Var a) el G →
    sc G (Var a)
| Falsum_L G a
  : Falsum el G →
    sc G a
| Implies_R G a b
  : sc (a :: G) b →
    sc G (Implies a b)
| Implies_L G a b c
  : (Implies a b) el G →
    sc G a →
    sc (b :: G) c →
    sc G c.
```

$$\frac{x \in \Gamma}{\Gamma \Rightarrow x}\,\{Id\}$$

$$\frac{\Gamma \Rightarrow \bot}{\Gamma \Rightarrow \alpha}\,\{Ex\}$$

$$\frac{\Gamma \cup \{\alpha\} \Rightarrow \beta}{\Gamma \Rightarrow \alpha \supset \beta}\,\{\supset\!-I\}$$

$$\frac{\begin{array}{c}\alpha \supset \beta \in \Gamma \\ \Gamma \Rightarrow \alpha \\ \Gamma \cup \{\beta\} \vdash \gamma\end{array}}{\Gamma \Rightarrow \gamma}\,\{\supset\!-E\}$$

Using the previous definitions we can prove consistency of our minimal logic by implementing Gentzen's argument using Coq. In the next section, we outline the theorems and lemmas proved.

B. Proving Consistency

In order to prove the admissibility of cut for sequent-calculus, we need to prove *weakening*, which states that the inclusion of new hypothesis does not change provability.

Lemma 1 (Weakening)**.** If $\Gamma \Rightarrow \alpha$ and $\Gamma \subseteq \Gamma'$ then $\Gamma' \Rightarrow \alpha$.

Proof. By structural induction on the derivation of $\Gamma \Rightarrow \alpha$. □

Using weakening property we can prove a generalized version of admissibility of cut. We first prove this auxiliar lemma in order to get a stronger induction hypothesis. From the next lemma, admissibility is just a corollary.

Lemma 2 (Genealized admissibility)**.** If $\Gamma \Rightarrow \alpha$ and $\Gamma' \Rightarrow \beta$ then $\Gamma \cup (\Gamma' - \{\alpha\}) \Rightarrow \beta$.

Proof. By structural induction on α using Lemma 1 and nested inductions on $\Gamma \Rightarrow \alpha$ and $\Gamma' \Rightarrow \beta$, when needed. □

Corolary 1 (Admissibility of cut)**.** If $\Gamma \Rightarrow \alpha$ and $\Gamma \cup \{\alpha\} \Rightarrow \beta$ then $\Gamma \Rightarrow \beta$.

Proof. Immediate consequence of Lemmas 1 and 2. □

Consistency of sequent calculus trivially by inspection on the structure of derivations.

Theorem 1 (Consistency of sequent calculus)**.** There is no proof of $\emptyset \Rightarrow \bot$.

Proof. Immediate from the sequent calculus rules (there is no rule to introduce $\bot$). □

The Coq code for all these proofs can be found in file SequentCalculus.v online [Sasdelli et al. 2020].

The next step in the mechanization of the consistency of our minimal logic is to stabilish the equivalence between sequent calculus and natural deduction systems. The equivalence proofs between these two formalism are based on a routine induction on derivations using admissibility of cut. We omit its description for brevity. The complete proofs of these equivalence results can be found in file NatDed.v in the source code repository [Sasdelli et al. 2020].

Finally, we can prove the consistency of natural deduction by combining the proofs of consistency of the sequent calculus and the equivalence between these formalisms.

Theorem 2 (Consistency for Natural Deduction)**.** There is no proof of $\emptyset \vdash \bot$.

Proof. Suppose that $\emptyset \vdash \bot$. By the equivalence between natural deduction and sequent calculus, we have $\emptyset \Rightarrow \bot$, which contradicts Theorem 1. $\square$