Abstract

I prove the equivalence of tableau calculus and sequent calculus.

0.1 Introduction

It is essential to prove or disprove that a formula is a theorem in many areas of applied logic. This is done through automatic reasoning, theorem provers. One such versatile and successful type of theorem provers are tableau-based theorem provers. These are theorem prover that utilise the semantic tableau, a procedure which can determine if a formula is satisfiable, and hence if the formula is a theorem. We aim to build a tool that given a set of tableau rules, among other parameters, it will automatically generate a corresponding tableau-based theorem provers which is proven to be correct. We aim to do this for classical propositional logic first in the hopes that this can be extend for more complex logical systems in the future.

In this project we have the following,

1. Explored the formalisation of the equivalence of sequent calculi and tableau calculi in Coq.
2. Created an theorem a tableau-based theorem prover for classical propositional logic using Coq.

In the first part of the project, the formalisation of sequent calculi and tableau calculi was explored and how it could be encoded into Coq. Sequent calculi, like tableau calculi, can be utilised to determine whether a formula is satisfiable. Gentzen first introduced the "calculus of sequents" in "Untersuchungen uber das logische Schliessen" (Investigations into logical deduction - gerhard gentzen)). It is now a common system to use when reasoning about satisfiability of formula in classical propositional logic. It follows that there is a strong relationship between the notion of a closed tableau in the tableau calculus and a sequent being derivable in the calculus of sequents. By exploring the equivalence of sequent calculus and tableau calculus, a transformation from the semantics of sequent calculus and tableau calculus can be made in Coq. With this, userss of of our tool, the automatic synthesiser of tableau-based theorem provers, will not need prerequisite knowledge about the tableau calculus to use the tool. Instead, they can use the semantics of the common sequent calculus. (Not too sure if this is correct. Need more citations)

(TWB Section. Two Para) In the second part of the project, we attempt to make a generalised synthesiser for logical systems. We aim to make a tool kit in Coq which allows users to input logical rules and then extract a corresponding theorem provers which is proven to be correct with respect to the logical system the user proves the logical rules corresponds to.

This project is aimed to extend the work done in the Tableau Work Bench (ADD CITATION - Pietro). The Tableau Work Bench, implemented in O'Caml, exists as a "user-friendly framework for building automated tableau-based theorem provers". It allows users to give an input set of rules for their custom logical system in propositional logic to generate a corresponding tableau-based theorem prover. However, the generated tableau-based theorem provers generated are not guaranteed correctness.

Apart from the Tableau Work Bench, to implement a theorem prover for a custom logic system, assuming there is no pre-existing one already, one would have to implement and program the theorem prover from scratch.

Furthermore, if one wants to prove that a logical set of rules corresponds to a particular logical system they would have to prove this separately from their corresponding implementation of a theorem prover. This is a major issue on knowing if the implementation of the theorem prover correctly corresponds to the logical system the user wants it to describe.

We aim to developed a tool which will still allow users to quickly prototype and implement calculi they wish to test like the Tableau Work Bench while also providing an option to have the implementation proven to be correct. To simplify, in this project we only look at classical propositional logic.

0.2 Background

0.2.1 Tableau Calculus

A tableau calculus provides a decision procedure to determine if a formula is satisfiable through the decomposition of sets of formulae. Further more, the tableau method can be used to determine if a formula is valid in a specific logical system.

More specifically, a tableau calculus consists of a finite set of rules which describe the logical system, L, the tableau method can determine if formulae are valid with respect to. Underlying the tableau method is a tree structure, where branches represent a rule application.
Definition 0.2.1. Tableau Rule

The rules of a tableau are expressed as sets, multi-sets or lists depending on the logics being expressed. We will express the tableau rules as a set as we are working within classical propositional logic. A rule is composed of a numerator and a denominator. A numerator $\mathcal{N}$ is a set of formulae in the logical system L. A denominator is either a set of branches, $\mathcal{D}_i$, which are each sets of formulae in L or the symbol $\bot$ signifying a closed tableau, indicated the termination of a branch. These rules are typically written as the following,

$$(\rho)\frac{\mathcal{N}}{\mathcal{D}_1|\cdots|\mathcal{D}_n} \qquad\qquad (\rho')\frac{\mathcal{N}}{\bot}$$

To apply a rule to a formula set Γ, the variables in the numerator $\mathcal{N}$ must be unified to match Γ. Then the denominator must be instantiated following the unification of the numerator. Each branch of the denominator acts as subgoals.

A tableau for a formula set Γ is a tree of nodes where Γ is the root and all children of a node are applications of a rule on that node.
Definition 0.2.2. Invertible Rule A rule ρ is invertible if and only if whenever there exists a closed tableau for an instance of its numerator, there exists a closed tableau for each branch in its denominator.

0.2.2 Sequent Calculus

A sequent calculus is a very useful tool in proof theory. Proving the validity of formulas in a sequent calculus is similar to the tableau method, however the node in a sequent calculus proof are a sequent of formulae instead of a set of formulae.
Definition 0.2.3. Sequent A sequent is an expression of the following form,

$$\Gamma \texttt{==>} \Delta$$

Where Γ and Δ are a set, multi-set or list of formulae depending on the calculi being defined. ==> acts as an auxiliary symbol. We will express a sequent with lists.

Γ is denoted as the antecedent and Δ is denoted as the succedent. Both of these expressions could be empty.

Given the sequent,

$$A_1, \ldots, A_n \texttt{==>} B_1, \ldots, B_n$$

This is equivalent to the following formula,

$$(A_1 \& \ldots \& A_n) \supset (B_1 \vee \ldots \vee B_n)$$

(Define more, seq rules, deduction tree, cut?)

0.2.3 Classical Propositional Logic

(Define tableau + sequent defs)

0.3 Proof Theory

The system of sequent calculus has been encoded into Coq based on the rules given by Floris van Doorn. First the notion of a sequent being a tuple of lists is defined, the left side and right side of a sequent. Given this, we encode the notion of sequent being derivable as a direct translation of the sequent rules.

Furthermore, the system of tableau calculus was encoded in a similar manner. A tableau is represented as a list. Then the notion of a closed tableau is established through a direct translation of the tableau rules.

To show that the system of tableau calculus is equivalent to the system of sequent calculus we aim to prove the following,

$$\text{closed } X \iff X = \Gamma \cup \neg\Delta \,\&\, \text{derivable } \Gamma ==> \Delta \tag{1}$$

This is proven to an extent in Coq. Currently the proof is reliant on the exchange lemma for the tableau calculus begin admitted.

The proof is also reliant on a slight modification of rules. However, equivalence of the rules with respect to the ones being used by Floris van Doorn can be achieved through the admissibility of the weakening lemma.

The admissibility of the structure rules, apart from the cut lemma, has been proven with Floris van Doorn's rules for sequent calculus by relying on a proof of the exchange lemma being admitted.

0.4 Tableau Rules

To implement a theorem prover in Coq, the notion of tableau rules was defined. The definition of a formula was taken from the work Floris van Doorn did. First we define the following,

```
Inductive Results :=
  | Open
  | Closed
  | Failure
.
Definition PropFSet := list PropF.
Definition Numerator := PropFSet.
Definition Denominator := sum (list PropFSet) Results.
Definition Rule := prod Numerator Denominator.
```

Now given the notion of a rule, we need a method to instantiate these rules to work on an arbitrary set of formulae, PropFSet, to apply the rule in a tableau proof.

The method to apply a rule is based on Jack Daniel Kelly's subthesis. We first define a partitioning procedure to generate all maps which match a schema to a set of formulas. This allows us to generate possible instantiations of a rule.

0.4.1 Partition Function

The partition function is defined as the following,

```
Definition partitions schema    := map filterpi (partitions_help schema    nil).
```

Where partitions_help is the following,

```
Definition partitions_help (schema : PropFSet) : PropFSet -> partitionTuple -> list partiti
  refine (Fix (length_wf PropF) (fun _ => PropFSet -> partitionTuple -> list partitionTupl
   (fun schema partitions_help_rec =>
   (match schema as schema' return (schema = schema' -> PropFSet -> partitionTuple -> list
    | nil => fun _ _ acc  => acc ::nil
    | s::ss => fun H    acc  => let    := partition_help s    acc  in
        flat_map (fun    => partitions_help_rec (applyPartition ss   ) _      )
    end) eq_refl) schema).
    rewrite H. assert (length (applyPartition ss   ) <= length ss) by apply unchanging.
    simpl. omega.
    Defined.

Definition partitions schema    := (partitions_help schema    nil).
```

The algorithm is the following,

1. For each formulas in schema we check if there is a formula in the set Γ which matches it.
2. The mapping from propositional variables in the schema to the formula in the set Γ is recorded.
3. If this can be done for each of the formula in the schema such that propositional variables are not mapped twice, (ie, $p \mapsto a$ and $p \mapsto b$), the list of maps are recorded.
4. This is repeated for all possible combinations of mapping.

This gives us all possible ways to instantiate the rules such that the rule can be applied to a tableau node.

0.4.2 Rule Application

To apply a rule we (currently) take the first such partition that allows us to apply the rule using the partition function (if one exists at all). With this we can instantiate the rule, take the set difference of the numerator of the rule and the tableau node we want to apply the rule to. Append that set to each part of the denominator of the rule, unless it is a result. In that case we are already done.

(Different types of rule application, to SetPropF, DerTree etc)

(Wellfounded Relations)

(Strategy language)

(Defining a terminating function for tree search)

(Backtracking and further properties)