Abstract

I prove the equivalence of tableau calculus and sequent calculus.

0.1 Introduction

0.2 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,

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

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.

0.3 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.

0.3.1 Partition Function

The partition function is defined as the following,

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

Where partisions_help is the following,

```
Definition partitions_help (schema : PropFSet) : PropFSet -> partitionTuple -> l
  refine (Fix (length_wf PropF) (fun _ => PropFSet -> partitionTuple -> list par
   (fun schema partitions_help_rec =>
   (match schema as schema' return (schema = schema' -> PropFSet -> partitionTup
    | 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 unch
    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.

0.3.2 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.