Assignment 2 Solutions

COMP 2411, Session 1, 2004

Q1: For example:

$$\dfrac{\dfrac{\dfrac{\varphi \wedge \neg\psi}{\varphi}\,\wedge E \quad \dfrac{[\varphi \to \psi]^1}{\psi}\,\to E \quad \dfrac{\varphi \wedge \neg\psi}{\neg\psi}\,\wedge E}{\dfrac{\bot}{\neg(\varphi \to \psi)}\,\neg I_1}\,\neg E}$$

Q2: Note that modus ponens is one of the rules of inference of Natural deduction minus RAA (it is $\to$ elimination). Hence to prove the claim of the exercise, it suffices to give a proof in Natural deduction minus RAA of the three axioms considered.

$$\dfrac{\dfrac{[\psi]^1 \quad [\varphi]^2}{\psi \to \varphi}\,\to I_1}{\varphi \to \psi \to \varphi}\,\to I_2$$

$$\dfrac{\dfrac{\dfrac{[\varphi]^1 \quad [\varphi \to \psi]^2}{\psi}\,\to E \quad \dfrac{[\varphi]^1 \quad [\varphi \to \psi \to \chi]^3}{\psi \to \chi}\,\to E}{\varphi \to \chi}\,\to I_1}{(\varphi \to \psi) \to \varphi \to \chi}\,\to I_2}{(\varphi \to \psi \to \chi) \to (\varphi \to \psi) \to \varphi \to \chi}\,\to I_3$$

$$\dfrac{\dfrac{\dfrac{[\neg\varphi]^2[\varphi]^1}{\bot}\,\neg E}{\psi}\,\bot}{\varphi \to \psi}\,\to I_1}{\neg\varphi \to \varphi \to \psi}\,\to I_2$$

Q3: For example:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[\psi]^1 \quad [\varphi]^2}{\psi \to \varphi}\,\to I_1}{(\varphi \to \psi) \vee (\psi \to \varphi)}\,\vee I \quad [\neg((\varphi \to \psi) \vee (\psi \to \varphi))]^3}{\bot}\,\neg E}{\psi}\,\bot}{\varphi \to \psi}\,\to I_2}{(\varphi \to \psi) \vee (\psi \to \varphi)}\,\vee I \quad \dfrac{[\neg((\varphi \to \psi) \vee (\psi \to \varphi))]^3}{\bot}\,\neg E}{(\varphi \to \psi) \vee (\psi \to \varphi)}\,\mathrm{RAA}_3$$

Q4: For example

```
%
%  standardize(+Formula, -StandardFormula)
%    StandardFormula is a formula built from negation only applied to atoms,
%    and disjunction and conjunction applied to (possibly empty) sets of formulas;
%    Formula is a formula defined from negation, binary disjunction,
%    conjunction, implication and logical equivalence, and unary disjunction
%    and conjunction as allowed in StandardFormula;
%    Formula is to be viewed as an abbreviation of Standardformula
%
standardize(A eqv B, Formula) :- !,
        standardize(and_set [or_set [neg A, B], or_set [neg B, A]], Formula).
standardize(A imp B, Formula) :- !,
        standardize(or_set [neg A, B], Formula).
standardize(A or B, or_set Formulas) :- !,
        standardize(A, AFormula),
        standardize(B, BFormula),
        sort([AFormula, BFormula], Formulas).
standardize(A and B, and_set Formulas) :- !,
        standardize(A, AFormula),
        standardize(B, BFormula),
        sort([AFormula, BFormula], Formulas).
standardize(or_set Forms, or_set Formulas) :- !,
        maplist(standardize, Forms, StandardForms),
        sort(StandardForms, Formulas).
standardize(and_set Forms, and_set Formulas) :- !,
        maplist(standardize, Forms, StandardForms),
        sort(StandardForms, Formulas).
standardize(neg neg A, Formula) :- !,
        standardize(A, Formula).
standardize(neg(A eqv B), Formula) :- !,
        standardize(or_set [and_set [A, neg B], and_set [B, neg A]], Formula).
standardize(neg(A imp B), Formula) :- !,
        standardize(and_set [A, neg B], Formula).
standardize(neg(A or B), and_set Formulas) :- !,
        standardize(neg A, AFormula),
        standardize(neg B, BFormula),
        sort([AFormula, BFormula], Formulas).
standardize(neg(A and B), or_set Formulas) :- !,
        standardize(neg A, AFormula),
        standardize(neg B, BFormula),
        sort([AFormula, BFormula], Formulas).
standardize(neg or_set Forms, and_set Formulas) :- !,
        maplist(negate, Forms, NegatedForms),
        maplist(standardize, NegatedForms, NegatedFormulas),
        sort(NegatedFormulas, Formulas).
standardize(neg and_set Forms, or_set Formulas) :- !,
```

```
        maplist(negate, Forms, NegatedForms),
        maplist(standardize, NegatedForms, NegatedFormulas),
        sort(NegatedFormulas, Formulas).
standardize(Formula, Formula).

%
%  negate(+Formula, - NegatedFormula)
%
negate(Formula, neg Formula).
```

Q5: For example:

```
%
%  create_tableau(+Formula, -Tableau)
%    Tableau is a pair of the form
%    (ListOfFormulas, ListOfSubtableaux),
%    where ListOfFormulas is initialized to [Formula];
%    the tableau is constructed by instantiating the logical
%    variable for ListOfSubtableaux
%
create_tableau(Formula, Tableau) :-
        Tableau = ([Formula], _),
        extend_tableau(Tableau).

%
%  extend_tableau(+(ListOfFormulas, ListOfSubtableaux)
%    1. check for a pair of contradicatory formulas (branch is closed)
%    2. check if Formulas contains only literals (branch is open)
%    3. if possible, find a conjunction in some member of ListOfFormulas;
%       either the conjunction is over the empty set and the branch is open,
%       or the conjuncts are added to ListOfFormulas to make up the root
%       of ListOfSubtableaux
%    4. otherwise find a disjunction in some member of ListOfFormulas;
%       either the disjunction is over the empty set and the branch is closed,
%       or each disjunct is added to ListOfFormulas to make up the root
%       of as many members of ListOfSubtableaux
%
extend_tableau((Formulas, [closed])) :-
        check_closed(Formulas), !.
extend_tableau((Formulas, [open])) :-
        checklist(literal, Formulas), !.
extend_tableau((Formulas, [SubTableau])) :-
        select(and_set Conjuncts, Formulas, RemainingFormulas), !,
        (   Conjuncts = [],
            SubTableau = open, !
        ;   append(Conjuncts, RemainingFormulas, NewFormulas),
            SubTableau = (NewFormulas, _),
```

```
            extend_tableau(SubTableau)
        ).
extend_tableau((Formulas, SubTableaux)) :-
        select(or_set Disjuncts, Formulas, RemainingFormulas),
        (   Disjuncts = [],
            SubTableaux = [closed], !
        ;   maplist(hook(RemainingFormulas), Disjuncts, NewFormulas),
            maplist(make_tableau, NewFormulas, SubTableaux),
            checklist(extend_tableau, SubTableaux)
        ).

%
%  make_tableau(+ListOfFormulas, -Tableau)
%    Tableau is a tableau whose root is labeled with ListOfFormulas
%    and with uninstantiated subtableaux.
%
make_tableau(Formulas, (Formulas, _)).


%
%  hook(+ListOfFormulas, +Formula, -NewListOfFormulas)
%
hook(Formulas, Formula, NewFormulas) :-
        append([Formula], Formulas, NewFormulas).

%
% literal(+Formula)
%
literal(Formula)  :-
        atom(Formula).
literal(neg Formula) :-
        atom(Formula).

%
%  check_closed(+Formulas)
%
check_closed(Formulas) :-
        member(Formula, Formulas),
        member(neg Formula, Formulas).
```