BFO

By bittner

September 7, 2008

Contents

theory *EMR*

imports *FOL*

begin
typedecl *Rg*

arities *Rg* :: *term*

consts

OR :: *Rg* => *Rg* => *o*
POR :: *Rg* => *Rg* => *o*
PR :: *Rg* => *Rg* => *o*
PPR :: *Rg* => *Rg* => *o*
Sum :: *Rg* => *Rg* => *Rg* => *o*
Diff :: *Rg* => *Rg* => *Rg* => *o*
Prod :: *Rg* => *Rg* => *Rg* => *o*
UniR :: *Rg* => *o*

axioms

PR-refl: (*ALL a*. *PR*(*a*,*a*))
PR-antisym: (*ALL a b*. (*PR*(*a*,*b*) & *PR*(*b*,*a*) --> *a*=*b*))
PR-trans: (*ALL a b c*. (*PR*(*a*,*b*) & *PR*(*b*,*c*) --> *PR*(*a*,*c*)))
PR-diff: (*ALL a b*. (~*PR*(*a*,*b*)) --> (*EX c*. *Diff*(*a*,*b*,*c*)))
PR-sum: (*ALL a b*. (*EX c*. *Sum*(*a*,*b*,*c*)))
PR-prod: (*ALL a b*. (*OR*(*a*,*b*) --> (*EX c*. *Prod*(*a*,*b*,*c*))))

defs

OR-def: *OR*(*a*,*b*) == (*EX c*. (*PR*(*c*,*a*) & *PR*(*c*,*b*)))
POR-def: *POR*(*a*,*b*) == *OR*(*a*,*b*) & ~*PR*(*a*,*b*) & ~*PR*(*b*,*a*)

PPR-def: *PPR*(*a*,*b*) == *PR*(*a*,*b*) & ~*PR*(*b*,*a*)
Sum-def: *Sum*(*a*,*b*,*c*) == (*ALL d*. *OR*(*d*,*c*) <−> (*OR*(*d*,*a*) | *OR*(*d*,*b*)))
Diff-def: *Diff*(*a*,*b*,*c*) == (*ALL d*. *OR*(*d*,*c*) <−> (*EX e*. (*PR*(*e*,*a*) & ~*OR*(*e*,*b*) & *OR*(*e*,*d*))))
Prod-def: *Prod*(*a*,*b*,*c*) == (*ALL d*. *OR*(*d*,*c*) <−> (*OR*(*d*,*a*) & *OR*(*d*,*b*)))
UniR-def: *UniR*(*a*) == (*ALL b*. *PR*(*b*,*a*))

lemma *PR-refl-rule*: *PR*(*a*,*a*)
apply(*insert PR-refl*)
apply(*auto*)
done

lemma *PR-trans-rule*: [|*PR*(*a*,*b*); *PR*(*b*,*c*)|] ==> *PR*(*a*,*c*)
apply(*insert PR-trans*)
apply(*erule allE*,*erule allE*,*erule allE*)
apply(*rule mp*)
apply(*assumption*)
apply(*rule conjI*)
apply(*assumption*)
apply(*assumption*)
done

lemma *PR-antisym-rule*: [|*PR*(*a*,*b*); *PR*(*b*,*a*)|] ==> *a*=*b*
apply(*insert PR-antisym*)
apply(*erule allE*,*erule allE*)
apply(*rule mp*)
apply(*assumption*)
apply(*rule conjI*)
apply(*assumption*)
apply(*assumption*)
done

lemma *PR-diff-rule*: ~*PR*(*a*,*b*)==>(*EX c*. *Diff*(*a*,*b*,*c*))
apply(*insert PR-diff*)
apply(*auto*)
done

lemma *PPR-imp-PR*: *PPR*(*a*,*b*) ==> *PR*(*a*,*b*)
apply(*unfold PPR-def*)
apply(*auto*)
done

theorem *PPR-asym*: *PPR*(*a*,*b*) ==> ~*PPR*(*b*,*a*)
apply(*unfold PPR-def*)

apply(*auto*)
done

theorem *PPR-trans*: [|*PPR*(*a*,*b*);*PPR*(*b*,*c*)|]==>*PPR*(*a*,*c*)
apply(*unfold PPR-def*)
apply(*clarify*)
apply(*rule conjI*)
apply(*rule PR-trans-rule*)
apply(*assumption*)
apply(*assumption*)
apply(*drule PR-trans-rule*)
apply(*assumption*)
apply(*rule notI*)
apply(*drule PR-trans-rule*)
apply(*assumption*)
apply(*erule notE*)
apply(*assumption*)
done

theorem *PR-imp-PPR-or-Id*: *PR*(*a*,*b*) ==> (*PPR*(*a*,*b*) | (*a*=*b*))
apply(*safe*)
apply(*unfold PPR-def*)
apply(*safe*)
apply(*drule PR-antisym-rule*)
apply(*auto*)
done

theorem *PPR-or-Id-imp-PR*: (*PPR*(*a*,*b*) | (*a*=*b*)) ==> *PR*(*a*,*b*)
apply(*insert PPR-imp-PR*)
apply(*insert PR-refl*)
apply(*auto*)
done

theorem *PR-and-PPR-imp-PPR*: [|*PR*(*a*,*b*);*PPR*(*b*,*c*)|] ==> *PPR*(*a*,*c*)
apply(*drule PR-imp-PPR-or-Id*)
apply(*safe*)
apply(*drule PPR-trans* [*of a b c*])
apply(*auto*)
done

theorem *PPR-and-PR-imp-PPR*: [|*PPR*(*a*,*b*);*PR*(*b*,*c*)|] ==> *PPR*(*a*,*c*)
apply(*drule PR-imp-PPR-or-Id* [*of b c*])
apply(*safe*)
apply(*drule PPR-trans* [*of a b c*])
apply(*auto*)
done

theorem *OR-refl*: *OR*(*a*,*a*)

apply(*unfold OR-def*)
apply(*insert PR-refl*)
apply(*auto*)
done

theorem *OR-sym*: $OR(a,b) ==> OR(b,a)$
apply(*unfold OR-def*)
apply(*auto*)
done

theorem *POR-sym*: $POR(a,b) ==> POR(b,a)$
apply(*unfold POR-def*)
apply(*insert OR-sym*)
apply(*auto*)
done

theorem *POR-irrefl*: $\sim POR(a,a)$
apply(*unfold POR-def*)
apply(*insert PR-refl*)
apply(*auto*)
done

theorem *PR-imp-OR*: $PR(a,b) ==> OR(a,b)$
apply(*unfold OR-def*)
apply(*insert PR-refl*)
apply(*auto*)
done

theorem *PR-and-OR*: $[|PR(a,b);OR(a,c)|] ==> OR(b,c)$
apply(*unfold OR-def*)
apply(*insert PR-trans-rule*)
apply(*auto*)
done

theorem *PR-and-notOR-imp-notOR*: $[|PR(a,b);\sim OR(b,c)|] ==> \sim OR(a,c)$
apply(*insert PR-and-OR*)
apply(*auto*)
done

theorem *notOR-sym*: $\sim OR(a,b) ==> \sim OR(b,a)$
apply(*insert OR-sym*)
apply(*auto*)
done

theorem *PR-ssuppl*: ~*PR*(*a*,*b*) ==> (*EX c*. (*PR*(*c*,*a*) & ~*OR*(*c*,*b*)))
apply(*drule PR-diff-rule*)
apply(*unfold Diff-def*)
apply(*insert OR-refl*)
apply(*auto*)
done

lemma *PR-ssuppl-transpos*: ~(*EX c*. (*PR*(*c*,*a*) & ~*OR*(*c*,*b*))) ==> *PR*(*a*,*b*)
apply(*insert PR-ssuppl*)
apply(*auto*)
done

theorem *OR-imp-OR-imp-PR*: (*ALL c*. (*OR*(*c*,*a*) --> *OR*(*c*,*b*))) ==> *PR*(*a*,*b*)
apply(*rule PR-ssuppl-transpos*)
apply(*insert PR-imp-OR*)
apply(*auto*)
done

theorem *OR-ident* : (*ALL a b*. (*ALL c*. (*OR*(*c*,*a*) <-> *OR*(*c*,*b*))) <-> *a*=*b*)
apply(*rule allI*,*rule allI*)
apply(*unfold iff-def*)
apply(*rule conjI*)
apply(*rule impI*)
apply(*rule PR-antisym-rule*)
apply(*auto*)
apply(*rule OR-imp-OR-imp-PR*)
apply(*auto*)
apply(*rule-tac a*=*b* **and** *b*=*a* **in** *OR-imp-OR-imp-PR*)
apply(*auto*)
done

theorem *Sum-unique*: [|*Sum*(*a*,*b*,*c*);*Sum*(*a*,*b*,*d*)|] ==> *c* = *d*
apply(*unfold Sum-def*)
apply(*rule PR-antisym-rule*)
apply(*insert OR-imp-OR-imp-PR*)
apply(*auto*)
done

theorem *Sum-imp-PR-and-PR*: *Sum*(*a*,*b*,*c*) ==> *PR*(*a*,*c*) & *PR*(*b*,*c*)
apply(*safe*)
apply(*rule-tac a*=*a* **and** *b*=*c* **in** *OR-imp-OR-imp-PR*)
apply(*unfold Sum-def*)
apply(*force*)
apply(*rule-tac a*=*b* **and** *b*=*c* **in** *OR-imp-OR-imp-PR*)
apply(*auto*)
done

theorem *Sum-sym*: *Sum*(a,b,c) ==> *Sum*(b,a,c)
apply(*unfold Sum-def*)
apply(*auto*)
done

theorem *Sum-refl*: *Sum*(a,a,a)
apply(*unfold Sum-def*)
apply(*auto*)
done

theorem *Sum-imp-id*: *Sum*(a,a,b) ==> a=b
apply(*unfold Sum-def*)
apply(*rule PR-antisym-rule*)
apply(*rule OR-imp-OR-imp-PR*)
apply(*auto*)
apply(*rule OR-imp-OR-imp-PR*)
apply(*auto*)
done

theorem *PR-imp-Sum*: *PR*(a,b) ==> *Sum*(a,b,b)
apply(*unfold Sum-def*)
apply(*safe*)
apply(*drule-tac* a=d **and** b=a **in** *OR-sym*)
apply(*drule-tac* a=a **and** b=b **and** c=d **in** *PR-and-OR*)
apply(*assumption*)
apply(*rule OR-sym*)
apply(*assumption*)
done

theorem *Diff-unique*: [|*Diff*(a,b,c);*Diff*(a,b,d)|] ==> c = d
apply(*unfold Diff-def*)
apply(*rule PR-antisym-rule*)
apply(*rule-tac* a=c **and** b=d **in** *OR-imp-OR-imp-PR*)
apply(*clarify*)
apply(*erule-tac* x=*ca* **in** *allE*)
apply(*clarify*)
apply(*frule-tac* x=*ca* **in** *spec*)
apply(*insert OR-refl*)
apply(*erule iffE*)
apply(*auto*)
apply(*rule-tac* a=d **and** b=c **in** *OR-imp-OR-imp-PR*)
apply(*clarify*)
apply(*rotate-tac 1*)
apply(*erule-tac* x=*ca* **in** *allE*)
apply(*clarify*)
apply(*frule-tac* x=c **in** *spec*)
apply(*erule iffE*)
apply(*auto*)
done

theorem *Prod-unique*: [|*Prod*(*a*,*b*,*c*);*Prod*(*a*,*b*,*d*)|] ==> *c* = *d*
apply(*unfold Prod-def*)
apply(*rule PR-antisym-rule*)
apply(*insert OR-imp-OR-imp-PR*)
apply(*auto*)
done

theorem *Diff-imp-notPR*: *Diff*(*a*,*b*,*c*) ==> ~*PR*(*a*,*b*)
apply(*unfold Diff-def*)
apply(*safe*)
apply(*erule-tac x*=*c* **in** *allE*)
apply(*insert OR-refl*)
apply(*auto*)
apply(*drule-tac a*=*e* **and** *b*=*b* **in** *notOR-sym*)
apply(*drule-tac a*=*a* **and** *b*=*b* **and** *c*=*e* **in** *PR-and-notOR-imp-notOR*)
apply(*assumption*)
apply(*drule-tac a*=*e* **and** *b*=*a* **in** *PR-imp-OR*)
apply(*drule-tac a*=*e* **and** *b*=*a* **in** *OR-sym*)
apply(*clarify*)
done

theorem *notOR-imp-Diff*: ~*OR*(*a*,*b*) ==> *Diff*(*a*,*b*,*a*)
apply(*unfold Diff-def*)
apply(*safe*)
apply(*rule-tac x*=*a* **in** *exI*)
apply(*auto*)
apply(*insert PR-refl*)
apply(*force*)
apply(*insert OR-sym*)
apply(*force*)
apply(*drule-tac a*=*e* **and** *b*=*a* **and** *c*=*d* **in** *PR-and-OR*)
apply(*auto*)
done

theorem *Diff-imp-PR*: *Diff*(*a*,*b*,*c*) ==> *PR*(*c*,*a*)
apply(*unfold Diff-def*)
apply(*rule-tac a*=*c* **and** *b*=*a* **in** *OR-imp-OR-imp-PR*)
apply(*rule allI*)
apply(*clarify*)
apply(*erule-tac x*=*ca* **in** *allE*)
apply(*safe*)
apply(*drule-tac a*=*e* **and** *b*=*a* **and** *c*=*ca* **in** *PR-and-OR*)
apply(*assumption*)
apply(*rule OR-sym*)
apply(*assumption*)
done

theorem *PR-and-notPR-imp-notPR*: [|*PR*(*a*,*b*);~*PR*(*c*,*b*)|] ==> ~*PR*(*c*,*a*)
apply(*safe*)
apply(*drule PR-trans-rule* [*of c a b*])
apply(*auto*)
done

theorem *PR-and-Diff-impl-Diff-PR*: [|*PR*(*a*,*b*);*Diff*(*c*,*b*,*d*)|] ==> (*EX e*. (*Diff*(*c*,*a*,*e*) & *PR*(*d*,*e*)))
apply(*frule Diff-imp-notPR* [*of c b d*])
apply(*frule Diff-imp-PR* [*of c b d*])
apply(*frule PR-and-notPR-imp-notPR* [*of a b c*])
apply(*assumption*)
apply(*frule PR-diff-rule* [*of c a*])
apply(*safe*)
apply(*rule-tac x=ca* **in** *exI*)
apply(*safe*)
apply(*insert excluded-middle* [*of OR*(*c*,*a*)])
apply(*safe*)
apply(*drule-tac a=c* **and** *b=a* **in** *notOR-imp-Diff*)
apply(*rotate-tac 6*)
apply(*drule Diff-unique*)
apply(*assumption*)
apply(*force*)
apply(*rule-tac a=d* **and** *b=ca* **in** *OR-imp-OR-imp-PR*)
apply(*safe*)
apply(*drule-tac a=caa* **and** *b=d* **in** *OR-sym*)
apply(*frule-tac a=d* **and** *b=c* **and** *c=caa* **in** *PR-and-OR*)
apply(*assumption*)
apply(*unfold Diff-def*)
apply(*rotate-tac 5*)
apply(*erule-tac x=caa* **in** *allE*)
apply(*erule-tac x=caa* **in** *allE*)
apply(*drule-tac a=d* **and** *b=caa* **in** *OR-sym*)
apply(*rotate-tac 7*)
apply(*erule iffE*)
apply(*drule-tac P=OR*(*caa*, *d*) **and** *Q*=(*EX* (*e*::*Rg*). (*PR*(*e*, *c*) & ((~ *OR*(*e*, *b*)) & *OR*(*e*, *caa*)))) **in** *mp*)
apply(*assumption*)
apply(*erule exE*)
apply(*erule conjE*)
apply(*erule conjE*)
apply(*drule-tac a=e* **and** *b=b* **in** *notOR-sym*)
apply(*frule-tac a=a* **and** *b=b* **and** *c=e* **in** *PR-and-notOR-imp-notOR*)
apply(*assumption*)
thm *conjI*
apply(*drule-tac a=a* **and** *b=e* **in** *notOR-sym*)
apply(*drule-tac P=PR*(*e*, *c*) **and** *Q*=~ *OR*(*e*, *a*) **in** *conjI*)

apply(*assumption*)
apply(*drule-tac P*=(*PR*(*e*, *c*) & (~ *OR*(*e*, *a*))) **and** *Q*=*OR*(*e*, *caa*) **in** *conjI*)
apply(*assumption*)
apply(*drule-tac P*=% *e*. ((*PR*(*e*, *c*) & (~ *OR*(*e*, *a*))) & *OR*(*e*, *caa*)) **in** *exI*)
apply(*erule iffE*)
apply(*auto*)
done

theorem *UniR-unique*: [|*UniR*(*a*);*UniR*(*b*)|] ==> *a*=*b*
apply(*unfold UniR-def*)
apply(*rule-tac a*=*a* **and** *b*=*b* **in** *PR-antisym-rule*)
apply(*auto*)
done

end

theory *QSizeR*

imports *EMR*

begin

consts

SSR :: *Rg* => *Rg* => *o*
Sym :: *Rg* => *Rg* => *Rg* => *o*
LER :: *Rg* => *Rg* => *o*

axioms

SSR-refl: *ALL a*. *SSR*(*a*,*a*)
SSR-sym: *SSR*(*a*,*b*) ==> *SSR*(*b*,*a*)
SSR-trans: [|*SSR*(*a*,*b*);*SSR*(*b*,*c*)|] ==> *SSR*(*a*,*c*)
PR-and-SSR-imp-PR: [|*PR*(*a*,*b*);*SSR*(*a*,*b*)|] ==> *PR*(*b*,*a*)
SSR-plus: [|*Plus*(*a*,*c*,*d1*);*Plus*(*b*,*c*,*d2*);*Sym*(*c*,*a*,*b*)|] ==> (*SSR*(*a*,*b*) <-> *SSR*(*d1*,*d2*))
LER-total: *ALL a b*. (*LER*(*a*,*b*) | *LER*(*b*,*a*))
LER-and-LER-imp-SSR: [|*LER*(*a*,*b*);*LER*(*b*,*a*)|] ==> *SSR*(*a*,*b*)

lemma *SSR-refl-rule*: *SSR*(*a*,*a*)
apply(*insert SSR-refl*)
apply(*auto*)
done

defs

Sym-def: *Sym*(*c*,*a*,*b*) == (*ALL d*. (*PR*(*d*,*c*) --> (*PR*(*d*,*a*) <-> *PR*(*d*,*b*))))
LER-def: *LER*(*a*,*b*) == (*EX c*. (*SSR*(*c*,*a*) & *PR*(*c*,*b*)))

consts

RSSR :: *Rg* => *Rg* => *o*
NEGR :: *Rg* => *Rg* => *o*
SSCR :: *Rg* => *Rg* => *o*
LRSSR :: *Rg* => *Rg* => *o*
LNRSSR :: *Rg* => *Rg* => *o*

defs

NEGR-def: *NEGR*(*a*,*b*) == (*EX c1 c2*. (*SSR*(*c1*,*a*) & *PR*(*c1*,*b*) & *Diff*(*b*,*c1*,*c2*) & *RSSR*(*b*,*c2*)))
SSCR-def: *SSCR*(*a*,*b*) == (~*NEGR*(*a*,*b*) & ~*NEGR*(*b*,*a*))
LRSSR-def: *LRSSR*(*a*,*b*) == (*LER*(*a*,*b*) | *RSSR*(*a*,*b*))
LNRSSR-def: *LNRSSR*(*a*,*b*) == (*LRSSR*(*a*,*b*) & ~*RSSR*(*a*,*b*))

axioms

RSSR-refl: *ALL a*. *RSSR*(*a*,*a*)
RSSR-sym: *RSSR*(*a*,*b*) ==> *RSSR*(*b*,*a*)
RSSR-between: [|*RSSR*(*a*,*b*);*LER*(*a*,*c*);*LER*(*c*,*b*)|] ==> (*RSSR*(*c*,*a*) & *RSSR*(*c*,*b*))

RSSR-and-NEGR-imp-NEGR: [|*RSSR*(*a*,*b*);*NEGR*(*b*,*c*)|] ==> *NEGR*(*a*,*c*)
NEGR-and-RSSR-imp-NEGR: [|*NEGR*(*a*,*b*);*RSSR*(*b*,*c*)|] ==> *NEGR*(*a*,*c*)

NEGR-and-LER-imp-NEGR: [|*NEGR*(*a*,*b*);*LER*(*b*,*c*)|] ==> *NEGR*(*a*,*c*)

RSSR-sum: [|*Sum*(*a*,*c*,*d1*);*Sum*(*b*,*c*,*d2*);*Sym*(*c*,*a*,*b*);*RSSR*(*a*,*b*)|] ==> *RSSR*(*d1*,*d2*)
RSSR-sum2: [|*Sum*(*a*,*b*,*c*);*NEGR*(*a*,*c*)|] ==> ~*LRSSR*(*b*,*a*)

theorem *Id-imp-SSR*: *a*=*b* ==> *SSR*(*a*,*b*)
apply(*insert SSR-refl*)
apply(*auto*)
done

theorem *PR-and-PR-imp-SSR*: [|*PR*(*a*,*b*);*PR*(*b*,*a*)|] ==> *SSR*(*a*,*b*)
apply(*insert PR-antisym*,*insert Id-imp-SSR*)
apply(*auto*)
done

theorem *PR-and-SSR-imp-Id*: [|*PR*(*a*,*b*);*SSR*(*a*,*b*)|] ==> *a* = *b*

apply(frule-tac a=a **and** b=b **in** PR-and-SSR-imp-PR)
apply(assumption)
apply(rule PR-antisym-rule)
apply(safe)
done

theorem PR-imp-LER: PR(a,b) ==> LER(a,b)
apply(unfold LER-def)
apply(rule-tac x=a **in** exI)
apply(insert SSR-refl)
apply(auto)
done

theorem PPR-imp-notSSR: PPR(a,b) ==> ~SSR(a,b)
apply(rule ccontr)
apply(auto)
apply(frule-tac a=a **and** b=b **in** PPR-imp-PR)
apply(drule-tac a=a **and** b=b **in** PR-and-SSR-imp-PR)
apply(assumption)
apply(unfold PPR-def)
apply(auto)
done

theorem LER-refl: LER(a,a)
apply(unfold LER-def)
apply(insert SSR-refl,insert PR-refl)
apply(auto)
done

theorem LER-and-SSR-imp-LER: [|LER(a,b);SSR(b,c)|] ==> LER(a,c)
apply(insert LER-total)
apply(erule-tac x=a **in** allE)
apply(erule-tac x=c **in** allE)
apply(clarify)
apply(unfold LER-def)
apply(frule-tac exE)
apply(assumption)
apply(erule exE)
apply(erule exE)
apply(fold LER-def)
apply(clarify)
apply(drule-tac a=b **and** b=c **in** SSR-sym)
apply(frule-tac a=caa **and** b=c **and** c=b **in** SSR-trans)
apply(assumption)
apply(frule-tac P=SSR(caa,b) **and** Q=PR(caa,a) **in** conjI)
apply(assumption)
apply(drule-tac P=% caa. (SSR(caa, b) & PR(caa, a)) **in** exI)

apply(*fold LER-def*)
apply(*drule-tac a=a* **and** *b=b* **in** *LER-and-LER-imp-SSR*)
apply(*assumption*)
apply(*drule-tac a=c* **and** *b=b* **in** *SSR-sym*)
apply(*drule-tac a=a* **and** *b=b* **and** *c=c* **in** *SSR-trans*)
apply(*assumption*)
apply(*unfold LER-def*)
apply(*rule-tac x=c* **in** *exI*)
apply(*drule-tac a=a* **and** *b=c* **in** *SSR-sym*)
apply(*insert PR-refl*)
apply(*auto*)
done

theorem *SSR-and-LER-imp-LER*: [|*SSR*(*c*,*a*);*LER*(*a*,*b*)|] ==> *LER*(*c*,*b*)
apply(*unfold LER-def*)
apply(*clarify*)
apply(*drule-tac a=ca* **and** *b=a* **in** *SSR-sym*)
apply(*drule-tac a=c* **and** *b=a* **and** *c=ca* **in** *SSR-trans*)
apply(*assumption*)
apply(*drule-tac a=c* **and** *b=ca* **in** *SSR-sym*)
apply(*rule-tac x=ca* **in** *exI*)
apply(*auto*)
done

theorem *SSR-imp-LER*: *SSR*(*a*,*b*) ==> *LER*(*a*,*b*)
apply(*unfold LER-def*)
apply(*rule-tac x=b* **in** *exI*)
apply(*drule SSR-sym*)
apply(*insert PR-refl*)
apply(*auto*)
done

theorem *SSR-imp-LER-and-LER*: *SSR*(*a*,*b*) ==> (*LER*(*a*,*b*) & *LER*(*b*,*a*))
apply(*frule SSR-imp-LER*)
apply(*drule SSR-sym*)
apply(*drule SSR-imp-LER*)
apply(*auto*)
done

theorem *LER-trans*: [|*LER*(*a*,*b*);*LER*(*b*,*c*)|] ==> *LER*(*a*,*c*)
apply(*insert LER-total*)
apply(*erule-tac x=a* **in** *allE*)
apply(*erule-tac x=c* **in** *allE*)
apply(*clarify*)
apply(*unfold LER-def*)
thm *exE*

apply(*frule-tac P=% c. (SSR(c, a) & PR(c, b))* **and** *R=(EX (ca::Rg). (SSR(ca, a) & PR(ca, c)))* **in** *exE*)
prefer *2*
apply(*assumption*)
apply(*frule-tac P=% ca. (SSR(ca, b) & PR(ca, c))* **and** *R=(EX (ca::Rg). (SSR(ca, a) & PR(ca, c)))* **in** *exE*)
prefer *2*
apply(*assumption*)
apply(*frule-tac P=% ca. (SSR(ca, c) & PR(ca, a))* **and** *R=(EX (ca::Rg). (SSR(ca, a) & PR(ca, c)))* **in** *exE*)
prefer *2*
apply(*assumption*)
apply(*fold LER-def*)
apply(*clarify*)
apply(*insert LER-total*)
apply(*erule-tac x=xb* **in** *allE*)
apply(*erule-tac x=b* **in** *allE*)
apply(*safe*)
apply(*unfold LER-def*)
apply(*erule-tac P=% c. (SSR(c, xb) & PR(c, b))* **in** *exE*)
apply(*fold LER-def*)
apply(*clarify*)
apply(*unfold LER-def*)
apply(*drule-tac a=xc* **and** *b=xb* **and** *c=c* **in** *SSR-trans*)
apply(*assumption*)
apply(*drule-tac P=SSR(xc, c)* **and** *Q=PR(xc, b)* **in** *conjI*)
apply(*assumption*)
thm *exI*
apply(*drule-tac P=% xc. (SSR(xc, c) & PR(xc, b))* **in** *exI*)
apply(*fold LER-def*)
apply(*drule-tac a=b* **and** *b=c* **in** *LER-and-LER-imp-SSR*)
apply(*assumption*)
apply(*drule-tac a=b* **and** *b=c* **in** *SSR-sym*)
apply(*drule-tac a=xb* **and** *b=c* **and** *c=b* **in** *SSR-trans*)
apply(*assumption*)
apply(*drule-tac P=SSR(xb, b)* **and** *Q=PR(xb, a)* **in** *conjI*)
apply(*assumption*)
apply(*drule-tac P=% xb. (SSR(xb, b) & PR(xb, a))* **in** *exI*)
apply(*fold LER-def*)
apply(*drule-tac a=a* **and** *b=b* **in** *LER-and-LER-imp-SSR*)
apply(*assumption*)
apply(*drule-tac a=c* **and** *b=b* **in** *SSR-sym*)
apply(*drule-tac a=a* **and** *b=b* **and** *c=c* **in** *SSR-trans*)
apply(*assumption*)
apply(*unfold LER-def*)
apply(*rule-tac x=c* **in** *exI*)
apply(*drule-tac a=a* **and** *b=c* **in** *SSR-sym*)
apply(*insert PR-refl*)
apply(*force*)

apply(*erule-tac P=% c.* (*SSR*(*c*, *b*) & *PR*(*c*, *xb*)) **in** *exE*)
apply(*fold LER-def*)
apply(*clarify*)
apply(*drule-tac a=xc* **and** *b=xb* **and** *c=a* **in** *PR-trans-rule*)
apply(*assumption*)
apply(*drule-tac P=SSR*(*xc*, *b*) **and** *Q=PR*(*xc*, *a*) **in** *conjI*)
apply(*assumption*)
apply(*drule-tac P=% xc.* (*SSR*(*xc*, *b*) & *PR*(*xc*, *a*)) **in** *exI*)
apply(*fold LER-def*)
apply(*drule-tac a=a* **and** *b=b* **in** *LER-and-LER-imp-SSR*)
apply(*assumption*)
apply(*drule-tac a=xa* **and** *b=b* **in** *SSR-sym*)
apply(*drule-tac a=a* **and** *b=b* **and** *c=xa* **in** *SSR-trans*)
apply(*assumption*)
apply(*unfold LER-def*)
apply(*rule-tac x=xa* **in** *exI*)
apply(*rule conjI*)
apply(*rule-tac b=xa* **and** *a=a* **in** *SSR-sym*)
apply(*auto*)
done

theorem *SSR-and-RSSR-imp-RSSR*: [|*SSR*(*a*,*b*);*RSSR*(*b*,*c*)|] ==> *RSSR*(*a*,*c*)
apply(*insert LER-total*)
apply(*erule-tac x=b* **in** *allE*)
apply(*erule-tac x=c* **in** *allE*)
apply(*safe*)
thm *SSR-and-LER-imp-LER*
apply(*frule-tac c=a* **and** *a=b* **and** *b=c* **in** *SSR-and-LER-imp-LER*)
apply(*assumption*)
apply(*drule-tac a=a* **and** *b=b* **in** *SSR-sym*)
apply(*drule-tac a=b* **and** *b=a* **in** *SSR-imp-LER*)
thm *RSSR-between*
apply(*drule-tac a=b* **and** *b=c* **and** *c=a* **in** *RSSR-between*)
apply(*assumption,assumption*)
apply(*clarify*)

apply(*frule-tac a=a* **and** *b=b* **in** *SSR-sym*)
apply(*frule-tac a=c* **and** *b=b* **and** *c=a* **in** *LER-and-SSR-imp-LER*)
apply(*assumption*)
apply(*drule-tac a=b* **and** *b=c* **in** *RSSR-sym*)
apply(*drule-tac a=a* **and** *b=b* **in** *SSR-imp-LER*)
apply(*drule-tac a=c* **and** *b=b* **and** *c=a* **in** *RSSR-between*)
apply(*assumption,assumption*)
apply(*clarify*)

done

theorem *RSSR-and-SSR-imp-RSSR*: [|*RSSR*(a,b);*SSR*(b,c)|] ==> *RSSR*(a,c)
apply(*drule-tac a=a* **and** *b=b* **in** *RSSR-sym*)
apply(*drule-tac a=b* **and** *b=c* **in** *SSR-sym*)
apply(*rule-tac a=c* **and** *b=a* **in** *RSSR-sym*)
apply(*rule-tac a=c* **and** *b=b* **and** *c=a* **in** *SSR-and-RSSR-imp-RSSR*)
apply(*auto*)
done

theorem *SSR-imp-RSSR*: *SSR*(a,b) ==> *RSSR*(a,b)
apply(*insert RSSR-refl*)
apply(*insert RSSR-and-SSR-imp-RSSR*)
apply(*auto*)
done

theorem *NEGR-imp-LER*: *NEGR*(a,b) ==> *LER*(a,b)
apply(*unfold NEGR-def*)
apply(*erule exE*,*erule exE*)
apply(*unfold LER-def*)
apply(*rule-tac x=c1* **in** *exI*)
apply(*auto*)
done

theorem *NEGR-imp-LER-and-notSSR*: *NEGR*(a,b) ==> (*LER*(a,b) & ~*SSR*(a,b))
apply(*unfold NEGR-def*)
apply(*unfold LER-def*)
apply(*safe*)
apply(*rule-tac x=c1* **in** *exI*)
apply(*safe*)
apply(*drule-tac a=c1* **and** *b=a* **and** *c=b* **in** *SSR-trans*)
apply(*assumption*)
apply(*drule-tac a=b* **and** *b=c1* **and** *c=c2* **in** *Diff-imp-notPR*)
apply(*drule-tac a=c1* **and** *b=b* **in** *PR-and-SSR-imp-PR*)
apply(*auto*)
done

theorem *NEGR-irrefl*: *ALL a*. (~*NEGR*(a,a))
apply(*rule allI*)
apply(*rule notI*)
apply(*drule NEGR-imp-LER-and-notSSR*)
apply(*insert SSR-refl*)
apply(*auto*)
done

theorem *NEGR-assym*: *NEGR*(a,b) ==> ~*NEGR*(b,a)
apply(*drule NEGR-imp-LER-and-notSSR*)

apply(*safe*)
apply(*drule-tac a=b* **and** *b=a* **in** *NEGR-imp-LER-and-notSSR*)
apply(*drule LER-and-LER-imp-SSR*)
apply(*auto*)
done

theorem *LER-and-NEGR-imp-NEGR*: [|*LER*(*a*,*b*);*NEGR*(*b*,*c*)|] ==> *NEGR*(*a*,*c*)
apply(*unfold NEGR-def*)
apply(*clarify*)
apply(*drule-tac a=c1* **and** *b=b* **in** *SSR-sym*)
apply(*drule-tac a=a* **and** *b=b* **and** *c=c1* **in** *LER-and-SSR-imp-LER*)
apply(*assumption*)
apply(*unfold LER-def*)
apply(*clarify*)
apply(*frule-tac a=ca* **and** *b=c1* **and** *c=c* **and** *d=c2* **in** *PR-and-Diff-impl-Diff-PR*)
apply(*assumption*)
apply(*clarify*)
apply(*frule-tac a=c* **and** *b=ca* **and** *c=e* **in** *Diff-imp-PR*)
apply(*drule-tac a=c* **and** *b=c2* **in** *RSSR-sym*)
apply(*frule-tac a=c2* **and** *b=e* **in** *PR-imp-LER*)
apply(*frule-tac a=e* **and** *b=c* **in** *PR-imp-LER*)
apply(*drule-tac a=c2* **and** *b=c* **and** *c=e* **in** *RSSR-between*)
apply(*assumption*,*assumption*)
apply(*rule-tac x=ca* **in** *exI*)
apply(*rule-tac x=e* **in** *exI*)
apply(*safe*)
apply(*drule-tac a=ca* **and** *b=c1* **and** *c=c* **in** *PR-trans-rule*)
apply(*safe*)
apply(*rule RSSR-sym*)
apply(*assumption*)
done

theorem *SSR-and-NEGR-imp-NEGR*: [|*SSR*(*a*,*b*);*NEGR*(*b*,*c*)|] ==> *NEGR*(*a*,*c*)
apply(*drule SSR-imp-LER*)
apply(*drule LER-and-NEGR-imp-NEGR*)
apply(*auto*)
done

theorem *NEGR-and-SSR-imp-NEGR*: [|*NEGR*(*a*,*b*);*SSR*(*b*,*c*)|] ==> *NEGR*(*a*,*c*)
apply(*drule SSR-imp-LER* [*of b c*])
apply(*drule NEGR-and-LER-imp-NEGR*)
apply(*auto*)
done

theorem *NEGR-trans*: [|*NEGR*(*a*,*b*);*NEGR*(*b*,*c*)|] ==> *NEGR*(*a*,*c*)
apply(*drule NEGR-imp-LER*)
apply(*drule LER-and-NEGR-imp-NEGR*)
apply(*auto*)

done

theorem *PR-and-NEGR-imp-NEGR*: [|*PR*(*a*,*b*);*NEGR*(*b*,*c*)|] ==> *NEGR*(*a*,*c*)
apply(*drule PR-imp-LER*)
apply(*drule LER-and-NEGR-imp-NEGR*)
apply(*auto*)
done

theorem *NEGR-and-PR-imp-NEGR*: [|*NEGR*(*a*,*b*);*PR*(*b*,*c*)|] ==> *NEGR*(*a*,*c*)
apply(*drule PR-imp-LER* [*of b c*])
apply(*drule NEGR-and-LER-imp-NEGR*)
apply(*auto*)
done

theorem *RSSR-imp-notNEGR*: *RSSR*(*a*,*b*) ==> ~*NEGR*(*a*,*b*)
apply(*rule ccontr*)
apply(*auto*)
apply(*insert PR-sum*)
apply(*erule-tac x*=*a* **in** *allE*)
apply(*erule-tac x*=*b* **in** *allE*)
apply(*clarify*)
apply(*frule-tac a*=*a* **and** *b*=*b* **and** *c*=*c* **in** *Sum-imp-PR-and-PR*)
apply(*clarify*)
apply(*drule-tac a*=*a* **and** *b*=*b* **and** *c*=*c* **in** *NEGR-and-PR-imp-NEGR*)
apply(*assumption*)
apply(*drule-tac a*=*a* **and** *b*=*b* **and** *c*=*c* **in** *RSSR-sum2*)
apply(*assumption*)
apply(*unfold LRSSR-def*)
apply(*drule-tac a*=*a* **and** *b*=*b* **in** *RSSR-sym*)
apply(*clarify*)
done

theorem *SSCR-refl*: *ALL a*. *SSCR*(*a*,*a*)
apply(*unfold SSCR-def*)
apply(*rule allI*)
apply(*insert NEGR-irrefl*)
apply(*auto*)
done

theorem *SSCR-sym*: *SSCR*(*a*,*b*) ==> *SSCR*(*b*,*a*)
apply(*unfold SSCR-def*)
apply(*auto*)
done

theorem *LRSSR-refl*: *LRSSR*(*a*,*a*)
apply(*unfold LRSSR-def*)
apply(*insert LER-refl* [*of a*])
apply(*insert RSSR-refl*)
apply(*auto*)
done

theorem *LRSSR-and-LRSSR-imp-RSSR*: [|*LRSSR*(*a*,*b*);*LRSSR*(*b*,*a*)|] ==> *RSSR*(*a*,*b*)
apply(*unfold LRSSR-def*)
apply(*safe*)
apply(*drule-tac a*=*a* **and** *b*=*b* **in** *LER-and-LER-imp-SSR*)
apply(*assumption*)
apply(*drule-tac a*=*a* **and** *b*=*b* **in** *SSR-imp-RSSR*)
apply(*safe*)
apply(*rule RSSR-sym*)
apply(*assumption*)
done

theorem *RSSR-imp-LRSSR-and-LRSSR*: *RSSR*(*a*,*b*) ==> (*LRSSR*(*a*,*b*) & *LRSSR*(*b*,*a*))
apply(*safe*)
apply(*unfold LRSSR-def*)
apply(*rule disjI2*)
apply(*clarify*)
apply(*rule disjI2*)
apply(*rule RSSR-sym*)
apply(*assumption*)
done

theorem *RSSR-iff-LRSSR-and-LRSSR*: *RSSR*(*a*,*b*) <-> (*LRSSR*(*a*,*b*) & *LRSSR*(*b*,*a*))
apply(*insert LRSSR-and-LRSSR-imp-RSSR* [*of a b*])
apply(*insert RSSR-imp-LRSSR-and-LRSSR* [*of a b*])
apply(*auto*)
done

theorem *LRSSR-and-NEGR-imp-NEGR*: [|*LRSSR*(*a*,*b*);*NEGR*(*b*,*c*)|] ==> *NEGR*(*a*,*c*)
apply(*unfold LRSSR-def*)
apply(*safe*)
apply(*rule LER-and-NEGR-imp-NEGR* [*of a b c*])
apply(*safe*)
apply(*rule RSSR-and-NEGR-imp-NEGR* [*of a b c*])
apply(*safe*)
done

theorem *NEGR-and-LRSSR-imp-NEGR*: [|*NEGR*(*a*,*b*);*LRSSR*(*b*,*c*)|] ==> *NEGR*(*a*,*c*)
apply(*unfold LRSSR-def*)
apply(*safe*)
apply(*rule NEGR-and-LER-imp-NEGR* [*of a b c*])

apply(*safe*)
apply(*rule NEGR-and-RSSR-imp-NEGR* [*of a b c*])
apply(*safe*)
done

theorem *LRSSR-total*: *LRSSR*(*a*,*b*) | *LRSSR*(*b*,*a*)
apply(*unfold LRSSR-def*)
apply(*safe*)
apply(*insert LER-total*)
apply(*erule-tac x*=*a* **in** *allE*)
apply(*erule-tac x*=*b* **in** *allE*)
apply(*safe*)
done

theorem *LNRSSR-asym*: *LNRSSR*(*a*,*b*) ==> ~*LNRSSR*(*b*,*a*)
apply(*rule ccontr*)
apply(*auto*)
apply(*unfold LNRSSR-def*)
apply(*unfold LRSSR-def*)
apply(*safe*)
apply(*drule LER-and-LER-imp-SSR* [*of a b*])
apply(*clarify*)
apply(*drule SSR-imp-RSSR* [*of a b*])
apply(*clarify*)
done

theorem *LNRSSR-trans*: [|*LNRSSR*(*a*,*b*);*LNRSSR*(*b*,*c*)|] ==> *LNRSSR*(*a*,*c*)
apply(*unfold LNRSSR-def*)
apply(*safe*)
apply(*unfold LRSSR-def*)
apply(*safe*)
apply(*rule LER-trans* [*of a b c*])
apply(*safe*)
apply(*drule-tac a*=*a* **and** *b*=*c* **and** *c*=*b* **in** *RSSR-between*)
apply(*auto*)
done

end
theory *RBG*

imports *EMR QSizeR*

begin

consts

SpR :: *Rg* => *o*
MxSpR :: *Rg* => *Rg* => *Rg* => *o*
CoPPR :: *Rg* => *Rg* => *o*

CGR :: *Rg* => *Rg* => *o*
CNGSpR :: *Rg* => *Rg* => *o*
ECR :: *Rg* => *Rg* => *o*
DCR :: *Rg* => *Rg* => *o*

defs

MxSpR-def: *MxSpR*(*a*,*b*,*c*) == *SpR*(*a*) & *SpR*(*b*) & *SpR*(*c*) & *PR*(*a*,*c*) & *PR*(*b*,*c*)
& ~*OR*(*a*,*b*) & (*ALL e*. (*SpR*(*e*) & *PR*(*a*,*e*) --> (*a* = *e* | *OR*(*e*,*b*) | ~*PR*(*e*,*c*))))
CoPPR-def: *CoPPR*(*a*,*b*) == *SpR*(*a*) & *SpR*(*b*) & *PPR*(*a*,*b*) & (*ALL c d*. (*MxSpR*(*c*,*a*,*b*)
& *MxSpR*(*d*,*a*,*b*) --> *SSR*(*c*,*d*)))
CGR-def: *CGR*(*a*,*b*) == (*EX c*. (*SpR*(*c*) & *OR*(*c*,*a*) & *OR*(*c*,*b*) & (*ALL d*.
(*CoPPR*(*d*,*c*) --> (*OR*(*d*,*a*) & *OR*(*d*,*b*))))))
CNGSpR-def: *CNGSpR*(*a*,*b*) == *SpR*(*a*) & *SpR*(*b*) & (*EX ca cb*. (*CoPPR*(*ca*,*cb*)
& *MxSpR*(*a*,*ca*,*cb*) & *MxSpR*(*b*,*ca*,*cb*)))
ECR-def: *ECR*(*a*,*b*) == *CGR*(*a*,*b*) & ~*OR*(*a*,*b*)
DCR-def: *DCR*(*a*,*b*) == ~*CGR*(*a*,*b*)

axioms

SP-nested: [|*Sp*(*a*);*Sp*(*b*);*Sp*(*c*);*MxSpR*(*u*,*a*,*c*);*MxSpR*(*v*,*a*,*c*);(*ALL ua va*. (((*MxSpR*(*ua*,*a*,*b*)
& *MxSpR*(*va*,*a*,*b*)) | (*MxSpR*(*ua*,*b*,*c*) & *MxSpR*(*va*,*b*,*c*))) --> *SSR*(*ua*,*va*)))|]
==> *SSR*(*u*,*v*)
SP-PPR-exists: *EX b*. (*SpR*(*b*) & *PPR*(*b*,*a*))

SSR-imp-CoPPR-and-MxSpR: [|*SpR*(*a*);*SpR*(*b*);*SSR*(*a*,*b*)|] ==> (*EX ca cb*. (*CoPPR*(*ca*,*cb*)
& *MxSpR*(*a*,*ca*,*cb*) & *MxSpR*(*b*,*ca*,*cb*)))
CGR-imp-CGR-imp-PR: (*ALL c*. (*CGR*(*c*,*a*) --> *CGR*(*c*,*b*))) ==> *PR*(*a*,*b*)

SpR-CoPPR-NEGR-exists: *SpR*(*a*) ==> (*EX b*. *CoPPR*(*b*,*a*) & *NEGR*(*b*,*a*))
SpR-and-LER-and-notSSR-imp-CoPPR-and-SSR-exists: [|*SpR*(*a*);*LER*(*b*,*a*);~*SSR*(*b*,*a*)|]
==> (*EX c*. (*CoPPR*(*c*,*a*) & *SSR*(*c*,*b*)))

SpR-and-SpR-and-PP-imp-MaxSpR: [|*SpR*(*a*);*SpR*(*b*);*PPR*(*a*,*b*)|] ==> (*EX c*. (*MxSpR*(*c*,*a*,*b*)))

lemma *SP-PR-exists*: *EX b*. (*SpR*(*b*) & *PR*(*b*,*a*))
apply(*insert SP-PPR-exists* [*of a*])
apply(*clarify*)
apply(*drule-tac b*=*a* **and** *a*=*b* **in** *PPR-imp-PR*)
apply(*rule-tac x*=*b* **in** *exI*)
apply(*safe*)

done

lemma *CoPPR-imp-PPR*: *CoPPR*(*a*,*b*) ==> *PPR*(*a*,*b*)
apply(*unfold CoPPR-def*)
apply(*auto*)
done

lemma *CoPPR-imp-SpR-and-SpR*: *CoPPR*(*a*,*b*) ==> (*SpR*(*a*) & *SpR*(*b*))
apply(*unfold CoPPR-def*)
apply(*auto*)
done

theorem *CoPPR-asym*: *CoPPR*(*a*,*b*) ==> ~*CoPPR*(*b*,*a*)
apply(*unfold CoPPR-def*)
apply(*insert PPR-asym*)
apply(*auto*)
done

theorem *CoPPR-trans*: [|*CoPPR*(*a*,*b*);*CoPPR*(*b*,*c*)|] ==> *CoPPR*(*a*,*c*)
apply(*unfold CoPPR-def*)
apply(*safe*)
apply(*rule-tac a*=*a* **and** *b*=*b* **and** *c*=*c* **in** *PPR-trans*)
apply(*assumption*)+
apply(*rule-tac a*=*a* **and** *b*=*b* **and** *c*=*c* **and** *u*=*ca* **and** *v*=*d* **in** *SP-nested*)
apply(*auto*)
done

theorem *PPR-exists*: (*EX b*. *PPR*(*b*,*a*))
apply(*insert SP-PPR-exists*)
apply(*auto*)
done

theorem *Sp-CoPPR-exists*: *SpR*(*a*) ==> (*EX b*. *CoPPR*(*b*,*a*))
apply(*insert SP-PPR-exists* [*of a*])
apply(*clarify*)
apply(*insert SpR-and-LER-and-notSSR-imp-CoPPR-and-SSR-exists* [*of a*])
apply(*frule-tac a*=*b* **and** *b*=*a* **in** *PPR-imp-notSSR*)
apply(*drule-tac a*=*b* **and** *b*=*a* **in** *PPR-imp-PR*)
apply(*frule-tac a*=*b* **and** *b*=*a* **in** *PR-imp-LER*)
apply(*auto*)
done

theorem *PR-and-NEGR-exists*: *EX b*. (*PR*(*b*,*a*) & *NEGR*(*b*,*a*))
apply(*insert SP-PPR-exists* [*of a*])
apply(*clarify*)
apply(*drule-tac a*=*b* **in** *SpR-CoPPR-NEGR-exists*)
apply(*clarify*)
apply(*drule-tac a*=*b* **and** *b*=*a* **in** *PPR-imp-PR*)

```
apply(drule-tac a=ba and b=b and c=a in NEGR-and-PR-imp-NEGR)
apply(assumption)
apply(drule-tac a=ba and b=b in CoPPR-imp-PPR)
apply(drule-tac a=ba and b=b in PPR-imp-PR)
apply(drule-tac a=ba and b=b and c=a in PR-trans-rule)
apply(assumption)
apply(rule-tac x=ba in exI)
apply(safe)
done

theorem CGR-refl: CGR(a,a)
apply(insert SP-PR-exists [of a])
apply(clarify)
apply(unfold CGR-def)
apply(rule-tac x=b in exI)
apply(safe)
apply(rule-tac a=b and b=a in PR-imp-OR,assumption)+
apply(unfold CoPPR-def)
apply(clarify)
apply(drule-tac a=d and b=b and c=a in PPR-and-PR-imp-PPR)
apply(assumption)
apply(drule-tac a=d and b=a in PPR-imp-PR)
apply(drule-tac a=d and b=a in PR-imp-OR)
apply(assumption)
apply(clarify)
apply(drule-tac a=d and b=b and c=a in PPR-and-PR-imp-PPR)
apply(assumption)
apply(drule-tac a=d and b=a in PPR-imp-PR)
apply(drule-tac a=d and b=a in PR-imp-OR)
apply(assumption)
done

theorem CGR-sym: CGR(a,b) ==> CGR(b,a)
apply(unfold CGR-def)
apply(clarify)
apply(rule-tac x=c in exI)
apply(auto)
done


theorem PR-imp-CGR-imp-CGR: PR(a,b) ==> (ALL c. (CGR(c,a) --> CGR(c,b)))
apply(unfold CGR-def)
apply(clarify)
apply(rule-tac x=ca in exI)
apply(safe)
apply(drule-tac a=ca and b=a in OR-sym)
apply(drule-tac a=a and b=b and c=ca in PR-and-OR)
apply(assumption)
apply(simp add: OR-sym)
```

```
apply(erule-tac x=d in allE)
apply(force)
apply(erule-tac x=d in allE)
apply(safe)
apply(drule-tac a=d and b=a in OR-sym)
apply(drule-tac a=a and b=b and c=d in PR-and-OR)
apply(assumption)
apply(simp add:OR-sym)
done

theorem OR-imp-CGR: OR(a,b) ==> CGR(a,b)
apply(unfold OR-def)
apply(clarify)
apply(drule-tac a=c and b=a in PR-imp-CGR-imp-CGR)
apply(drule-tac a=c and b=b in  PR-imp-CGR-imp-CGR)
apply(erule-tac x=c in allE)
apply(insert CGR-refl)
apply(auto)
apply(insert CGR-sym)
apply(auto)
done


theorem  PR-iff-CGR-imp-CGR: PR(a,b) <-> (ALL c. (CGR(c,a) --> CGR(c,b)))
apply(rule iffI)
apply(drule-tac a=a and b=b in PR-imp-CGR-imp-CGR)
apply(assumption)
apply(drule-tac a=a and b=b in  CGR-imp-CGR-imp-PR)
apply(assumption)
done

theorem  Id-iff-CGR-iff-CGR: a=b <-> (ALL c. (CGR(c,a) <-> CGR(c,b)))
apply(rule iffI)
apply(simp)
apply(rule-tac a=a and b=b in PR-antisym-rule)
apply(simp add: PR-iff-CGR-imp-CGR)
apply(simp add: PR-iff-CGR-imp-CGR)
done

theorem PR-imp-CGR: PR(a,b) ==> CGR(a,b)
apply(insert PR-imp-CGR-imp-CGR)
apply(insert CGR-refl)
apply(auto)
done



theorem PR-and-CGR-imp-CGR: [|PR(a,b);CGR(a,c)|] ==> CGR(b,c)
apply(drule PR-imp-CGR-imp-CGR)
```

apply(*insert CGR-sym*)
apply(*auto*)
done

theorem *PR-and-notCGR-imp-notCGR*: [|*PR*(*a*,*b*);~ *CGR*(*b*,*c*)|] ==> ~ *CGR*(*a*,*c*)
apply(*insert PR-and-CGR-imp-CGR*)
apply(*auto*)
done

theorem *Sp-sum*: *OR*(*w*,*a*) <−> (*EX b*. (*SpR*(*b*) & *PR*(*b*,*a*) & *OR*(*w*,*b*)))
apply(*safe*)
apply(*unfold OR-def*)
apply(*clarify*)
apply(*insert SP-PR-exists*)
apply(*drule allI*)
apply(*erule-tac x*=*c* **in** *allE*)
apply(*clarify*)
apply(*rule-tac x*=*b* **in** *exI*)
apply(*safe*)
apply(*rule-tac a*=*b* **and** *b*=*c* **and** *c*=*a* **in** *PR-trans-rule*)
apply(*auto*)
apply(*drule-tac a*=*b* **and** *b*=*c* **and** *c*=*w* **in** *PR-trans-rule*)
apply(*assumption*)
apply(*rule-tac x*=*b* **in** *exI*)
apply(*safe*)
apply(*rule-tac a*=*b* **in** *PR-refl-rule*)
apply(*drule-tac a*=*c* **and** *b*=*b* **and** *c*=*a* **in** *PR-trans-rule*)
apply(*assumption*)
apply(*rule-tac x*=*c* **in** *exI*)
apply(*auto*)
done

theorem *Sp-imp-CNGSpR*: *SpR*(*a*) ==> *CNGSpR*(*a*,*a*)
apply(*unfold CNGSpR-def*)
apply(*clarify*)
apply(*insert SSR-refl-rule* [*of a*])
apply(*insert SSR-imp-CoPPR-and-MxSpR* [*of a a*])
apply(*auto*)
done

theorem *CNGSpR-imp-SSR*: *CNGSpR*(*a*,*b*) ==> *SSR*(*a*,*b*)
apply(*unfold CNGSpR-def*)
apply(*clarify*)
apply(*unfold CoPPR-def*)
apply(*clarify*)
apply(*erule-tac x*=*a* **in** *allE*)
apply(*erule-tac x*=*b* **in** *allE*)

apply(*auto*)
done

theorem *Sp-and-SSR-imp-CNGSpR*: [|*SpR*(*a*);*SpR*(*b*);*SSR*(*a*,*b*)|] ==> *CNGSpR*(*a*,*b*)
apply(*unfold CNGSpR-def*)
apply(*frule-tac a=a* **and** *b=b* **in** *SSR-imp-CoPPR-and-MxSpR*)
apply(*auto*)
done

theorem *Sp-imp-SSR-iff-CONSpR*: [|*SpR*(*a*);*SpR*(*b*)|] ==> (*SSR*(*a*,*b*) <-> *CNGSpR*(*a*,*b*))
apply(*insert Sp-and-SSR-imp-CNGSpR*)
apply(*insert CNGSpR-imp-SSR*)
apply(*auto*)
done

theorem *CNGSpR-imp-Sp-and-Sp*: *CNGSpR*(*a*,*b*) ==> (*SpR*(*a*) & *SpR*(*b*))
apply(*unfold CNGSpR-def*)
apply(*auto*)
done

theorem *CNGSpR-sym*: *CNGSpR*(*a*,*b*) ==> *CNGSpR*(*b*,*a*)
apply(*unfold CNGSpR-def*)
apply(*auto*)
done

theorem *CNGSpR-trans*: [|*CNGSpR*(*a*,*b*);*CNGSpR*(*b*,*c*)|] ==> *CNGSpR*(*a*,*c*)
apply(*frule-tac a=a* **and** *b=b* **in** *CNGSpR-imp-SSR*)
apply(*frule-tac a=b* **and** *b=c* **in** *CNGSpR-imp-SSR*)
apply(*drule-tac a=a* **and** *b=b* **in** *CNGSpR-imp-Sp-and-Sp*)
apply(*drule-tac a=b* **and** *b=c* **in** *CNGSpR-imp-Sp-and-Sp*)
apply(*drule-tac a=a* **and** *b=b* **and** *c=c* **in** *SSR-trans*)
apply(*safe*)
apply(*drule-tac a=a* **and** *b=c* **in** *Sp-and-SSR-imp-CNGSpR*)
apply(*auto*)
done

theorem *ECR-sym*: *ECR*(*a*,*b*) ==> *ECR*(*b*,*a*)
apply(*unfold ECR-def*)
apply(*insert OR-sym*)
apply(*insert CGR-sym*)
apply(*auto*)
done

theorem *ECR-irrefl*: ~*ECR*(*a*,*a*)
apply(*unfold ECR-def*)
apply(*insert CGR-refl*)
apply(*insert OR-refl*)
apply(*auto*)

done

theorem *DCR-sym*: *DCR*(a,b) ==> *DCR*(b,a)
apply(*unfold DCR-def*)
apply(*insert CGR-sym*)
apply(*auto*)
done

theorem *DCR-irrefl*: ~*DCR*(a,a)
apply(*unfold DCR-def*)
apply(*insert CGR-refl*)
apply(*auto*)
done

end
theory *QDiaSizeR*

imports *RBG*

begin

consts

SSRdia :: *Rg* => *Rg* => *o*
LERdia :: *Rg* => *Rg* => *o*
MinBSpR :: *Rg* => *Rg* => *o*

BR :: *Rg* => *Rg* => *Rg* => *o*

defs

MinBSpR-def: *MinBSpR*(a,b) == *SpR*(a) & *PR*(b,a) & (*ALL c*. (*SpR*(c) & *PR*(b,c) --> *LER*(a,c)))
SSRdia-def: *SSRdia*(a,b) == (*EX ca cb*. (*MinBSpR*(*ca*,a) & *MinBSpR*(*cb*,b) & *SSR*(*ca*,*cb*)))
LERdia-def: *LERdia*(a,b) == (*EX ca cb*. (*MinBSpR*(*ca*,a) & *MinBSpR*(*cb*,b) & *LER*(*ca*,*cb*)))

BR-def: *BR*(a,b,c) == *SpR*(a) & *SpR*(b) & *SpR*(c) & (*EX sab sbc sac bab bbc bac*. (*Sum*(a,b,*sab*) & *Sum*(b,c,*sbc*) & *Sum*(a,c,*sac*) & *MinBSpR*(*bab*,*sab*) & *MinBSpR*(*bbc*,*sbc*) & *MinBSpR*(*bac*,*sac*) & *PR*(*bab*,*bac*) & *PR*(*bbc*,*bac*)))

axioms

MinBSpR-exists: (*EX b*. *MinBSpR*(b,a))
PR-and-MinBSpR-and-MinBSpR-imp-PR: [|*PR*(a,b);*MinBSpR*(*aa*,a);*MinBSpR*(*bb*,b)|]

==> PR(aa,bb)

BR-trans: [|BR(a,b,w);BR(b,c,w)|] ==> BR(a,b,c)
BR-connect: [|BR(a,b,w);BR(a,c,w)|] ==> (BR(a,b,c) | BR(a,c,b))

theorem MinBSpR-imp-PR: MinBSpR(a,b) ==> PR(b,a)
apply(unfold MinBSpR-def)
apply(auto)
done

theorem MinBSpR-and-MinBSpR-imp-SSR: [|MinBSpR(a,c);MinBSpR(b,c)|] ==> SSR(a,b)
apply(unfold MinBSpR-def)
apply(rule-tac a=a **and** b=b **in** LER-and-LER-imp-SSR)
apply(auto)
done

theorem MinBSpR-unique: [|MinBSpR(a,c);MinBSpR(b,c)|] ==> a=b
apply(rule PR-and-SSR-imp-Id)
apply(rule PR-and-MinBSpR-and-MinBSpR-imp-PR [of c c])
apply(rule PR-refl-rule)
apply(assumption)+
apply(rule MinBSpR-and-MinBSpR-imp-SSR)
apply(auto)
done

theorem SpR-iff-MinBSpR: SpR(a) <-> MinBSpR(a,a)
apply(rule iffI)
apply(unfold MinBSpR-def)
apply(safe)
apply(rule PR-refl-rule [of a])
apply(drule-tac a=a **and** b=c **in** PR-imp-LER)
apply(assumption)
done

theorem EX-SpR-CGR-and-CGR: (EX c. (SpR(c) & CGR(c,a) & CGR(c,b)))
apply(insert PR-sum)
apply(erule-tac x=a **in** allE)
apply(erule-tac x=b **in** allE)
apply(clarify)
apply(insert MinBSpR-exists)
apply(drule allI)
apply(erule-tac x=c **in** allE)

apply(*clarify*)
apply(*rule-tac x=ba* **in** *exI*)
apply(*safe*)
apply(*unfold MinBSpR-def*)
apply(*clarify*)
apply(*fold MinBSpR-def*)
apply(*drule-tac a=a* **and** *b=b* **and** *c=c* **in** *Sum-imp-PR-and-PR*)
apply(*clarify*)
apply(*drule-tac a=ba* **and** *b=c* **in** *MinBSpR-imp-PR*)
apply(*drule-tac a=a* **and** *b=c* **and** *c=ba* **in** *PR-trans-rule*)
apply(*assumption*)
apply(*drule-tac a=a* **and** *b=ba* **in** *PR-imp-CGR*)
apply(*rule CGR-sym*)
apply(*clarify*)
apply(*drule-tac a=a* **and** *b=b* **and** *c=c* **in** *Sum-imp-PR-and-PR*)
apply(*clarify*)
apply(*drule-tac a=ba* **and** *b=c* **in** *MinBSpR-imp-PR*)
apply(*drule-tac a=b* **and** *b=c* **and** *c=ba* **in** *PR-trans-rule*)
apply(*assumption*)
apply(*drule-tac a=b* **and** *b=ba* **in** *PR-imp-CGR*)
apply(*rule CGR-sym*)
apply(*clarify*)
done

theorem *BR-imp-PR*: *BR*(*a*,*b*,*a*) ==> *PR*(*b*,*a*)
apply(*unfold BR-def*)
apply(*clarify*)
apply(*frule-tac a=a* **and** *b=sac* **in** *Sum-imp-id*)
apply(*clarify*)
apply(*frule-tac a=a* **and** *b=b* **in** *Sum-sym*)
apply(*frule-tac a=b* **and** *b=a* **and** *c=sbc* **and** *d=sab* **in** *Sum-unique*)
apply(*clarify*)
apply(*frule-tac a=bab* **and** *c=sab* **and** *b=bbc* **in** *MinBSpR-unique*)
apply(*clarify*)
apply(*frule-tac a=b* **and** *b=a* **and** *c=sab* **in** *Sum-imp-PR-and-PR*)
apply(*clarify*)
apply(*frule-tac a=bbc* **and** *b=sab* **in** *MinBSpR-imp-PR*)
apply(*frule-tac a=b* **and** *b=sab* **and** *c=bbc* **in** *PR-trans-rule*)
apply(*assumption*)
apply(*frule-tac a=b* **and** *b=bbc* **and** *c=bac* **in** *PR-trans-rule*)
apply(*clarify*)
apply(*simp add*: *SpR-iff-MinBSpR*)
apply(*frule-tac a=a* **and** *b=bac* **and** *c=a* **in** *MinBSpR-unique*)
apply(*safe*)
done

theorem *BR-refl*: [|*SpR*(*a*);*SpR*(*b*)|] ==> *BR*(*a*,*a*,*b*)

```
apply(unfold BR-def)
apply(safe)
apply(rule-tac x=a in exI)
apply(insert Sum-refl [of a])
apply(insert PR-sum)
apply(erule-tac x=a in allE)
apply(erule-tac x=b in allE)
apply(clarify)
apply(rule-tac x=c in exI)
apply(rule-tac x=c in exI)
apply(insert MinBSpR-exists [of a])
apply(insert MinBSpR-exists)
apply(drule allI)
apply(erule-tac x=c in allE)
apply(clarify)
apply(rule-tac x=ba in exI)
apply(rule-tac x=baa in exI)
apply(rule-tac x=baa in exI)
apply(safe)
apply(drule-tac a=a and b=b and c=c in Sum-imp-PR-and-PR)
apply(clarify)
apply(drule-tac a=a and b=c and aa=ba and bb=baa in PR-and-MinBSpR-and-MinBSpR-imp-PR)

apply(safe)
apply(insert PR-refl)
apply(auto)
done

theorem BR-sym: BR(a,b,c) ==> BR(c,b,a)
apply(unfold BR-def)
apply(clarify)
apply(rule-tac x=sbc in exI)
apply(rule-tac x=sab in exI)
apply(rule-tac x=sac in exI)
apply(rule-tac x=bbc in exI)
apply(rule-tac x=bab in exI)
apply(rule-tac x=bac in exI)
apply(safe)
apply(insert Sum-sym)
apply(auto)
done


theorem SSRdia-refl: SSRdia(a,a)
apply(rule ccontr)
```

apply(*unfold SSRdia-def*)
apply(*auto*)
apply(*insert MinBSpR-exists* [*of a*])
apply(*clarify*)
apply(*erule-tac x*=*b* **in** *allE*)
apply(*clarify*)
apply(*erule-tac x*=*b* **in** *allE*)
apply(*safe*)
apply(*insert SSR-refl*)
apply(*erule-tac x*=*b* **in** *allE*)
apply(*auto*)
done

theorem *SSRdia-sym*: *SSRdia*(*a*,*b*) ==> *SSRdia*(*b*,*a*)
apply(*unfold SSRdia-def*)
apply(*clarify*)
apply(*rule-tac x*=*cb* **in** *exI*)
apply(*rule-tac x*=*ca* **in** *exI*)
apply(*safe*)
apply(*rule-tac a*=*ca* **and** *b*=*cb* **in** *SSR-sym*)
apply(*assumption*)
done

theorem *SSRdia-trans*: [|*SSRdia*(*a*,*b*);*SSRdia*(*b*,*c*)|] ==> *SSRdia*(*a*,*c*)
apply(*unfold SSRdia-def*)
apply(*clarify*)
apply(*drule-tac a*=*caa* **and** *b*=*cb* **and** *c*=*b* **in** *MinBSpR-unique*)
apply(*assumption*)
apply(*rule-tac x*=*ca* **in** *exI*)
apply(*rule-tac x*=*cba* **in** *exI*)
apply(*safe*)
apply(*drule-tac a*=*ca* **and** *b*=*cb* **and** *c*=*cba* **in** *SSR-trans*)
apply(*auto*)
done

theorem *LERdia-refl*: *LERdia*(*a*,*a*)
apply(*rule ccontr*)
apply(*unfold LERdia-def*)
apply(*auto*)
apply(*insert MinBSpR-exists* [*of a*])
apply(*clarify*)
apply(*erule-tac x*=*b* **in** *allE*)
apply(*clarify*)
apply(*erule-tac x*=*b* **in** *allE*)
apply(*safe*)
apply(*insert LER-refl*)
apply(*drule allI*)
apply(*erule-tac x*=*b* **in** *allE*)

apply(*auto*)
done

theorem *LERdia-trans*: [|*LERdia*(*a*,*b*);*LERdia*(*b*,*c*)|] ==> *LERdia*(*a*,*c*)
apply(*unfold LERdia-def*)
apply(*clarify*)
apply(*drule-tac a*=*caa* **and** *b*=*cb* **and** *c*=*b* **in** *MinBSpR-unique*)
apply(*assumption*)
apply(*rule-tac x*=*ca* **in** *exI*)
apply(*rule-tac x*=*cba* **in** *exI*)
apply(*safe*)
apply(*drule-tac a*=*ca* **and** *b*=*cb* **and** *c*=*cba* **in** *LER-trans*)
apply(*auto*)
done

theorem *LERdia-and-LERdia-imp-SSRdia*: [|*LERdia*(*a*,*b*);*LERdia*(*b*,*a*)|] ==> *SS-Rdia*(*a*,*b*)
apply(*unfold LERdia-def*,*unfold SSRdia-def*)
apply(*clarify*)
apply(*drule-tac a*=*ca* **and** *b*=*cba* **and** *c*=*a* **in** *MinBSpR-unique*)
apply(*assumption*)
apply(*drule-tac a*=*cb* **and** *b*=*caa* **and** *c*=*b* **in** *MinBSpR-unique*)
apply(*assumption*)
apply(*rule-tac x*=*ca* **in** *exI*)
apply(*rule-tac x*=*cb* **in** *exI*)
apply(*safe*)
apply(*drule-tac a*=*cba* **and** *b*=*caa* **in** *LER-and-LER-imp-SSR*)
apply(*auto*)
done

theorem *SSRdia-imp-LERdia-and-LERdia*: *SSRdia*(*a*,*b*) ==> (*LERdia*(*a*,*b*) & *LERdia*(*b*,*a*))
apply(*unfold SSRdia-def*,*unfold LERdia-def*)
apply(*safe*)
apply(*rule-tac x*=*ca* **in** *exI*)
apply(*rule-tac x*=*cb* **in** *exI*)
apply(*drule-tac a*=*ca* **and** *b*=*cb* **in** *SSR-imp-LER*)
apply(*safe*)
apply(*rule-tac x*=*cb* **in** *exI*)
apply(*rule-tac x*=*ca* **in** *exI*)
apply(*drule-tac a*=*ca* **and** *b*=*cb* **in** *SSR-sym*)
apply(*drule-tac a*=*cb* **and** *b*=*ca* **in** *SSR-imp-LER*)
apply(*safe*)
done

theorem *SSRdia-iff-LERdia-and-LERdia*: *SSRdia*(*a*,*b*) <-> (*LERdia*(*a*,*b*) & *LER-dia*(*b*,*a*))
apply(*rule iffI*)
apply(*drule SSRdia-imp-LERdia-and-LERdia*)

apply(*auto*)
apply(*rule LERdia-and-LERdia-imp-SSRdia*)
apply(*auto*)
done

theorem *LERdia-or-LERdia*: (*LERdia*(*a*,*b*) | *LERdia*(*b*,*a*))
apply(*insert LER-total*)
apply(*insert MinBSpR-exists* [*of a*])
apply(*insert MinBSpR-exists* [*of b*])
apply(*clarify*)
apply(*erule-tac x*=*ba* **in** *allE*)
apply(*erule-tac x*=*baa* **in** *allE*)
apply(*safe*)
apply(*unfold LERdia-def*)
apply(*auto*)
done

theorem *Id-imp-SSRdia*: *a*=*b* ==> *SSRdia*(*a*,*b*)
apply(*insert SSRdia-refl* [*of a*])
apply(*simp*)
done

theorem *PR-and-PR-imp-SSRdia*: [|*PR*(*a*,*b*);*PR*(*b*,*a*)|] ==> *SSRdia*(*a*,*b*)
apply(*rule Id-imp-SSRdia* [*of a b*])
apply(*rule PR-antisym-rule* [*of a b*])
apply(*safe*)
done

theorem *PR-imp-LERdia*: *PR*(*a*,*b*) ==> *LERdia*(*a*,*b*)
apply(*unfold LERdia-def*)
apply(*insert MinBSpR-exists* [*of a*])
apply(*insert MinBSpR-exists* [*of b*])
apply(*clarify*)
apply(*rule-tac x*=*ba* **in** *exI*)
apply(*rule-tac x*=*baa* **in** *exI*)
apply(*safe*)
apply(*drule-tac a*=*a* **and** *b*=*b* **and** *aa*=*ba* **and** *bb*=*baa* **in** *PR-and-MinBSpR-and-MinBSpR-imp-PR*)
apply(*safe*)
apply(*drule-tac a*=*ba* **and** *b*=*baa* **in** *PR-imp-LER*)
apply(*assumption*)
done

theorem *LERdia-and-SSRdia-imp-LERdia*: [|*LERdia*(*a*,*b*);*SSRdia*(*b*,*c*)|] ==> *LER-dia*(*a*,*c*)
apply(*unfold LERdia-def*,*unfold SSRdia-def*)
apply(*clarify*)
apply(*drule-tac a*=*caa* **and** *b*=*cb* **and** *c*=*b* **in** *MinBSpR-unique*)
apply(*safe*)
apply(*rule-tac x*=*ca* **in** *exI*)

apply(*rule-tac x=cba* **in** *exI*)
apply(*safe*)
apply(*drule-tac a=ca* **and** *b=cb* **and** *c=cba* **in** *LER-and-SSR-imp-LER*)
apply(*safe*)
done

theorem *SSRdia-and-LERdia-imp-LERdia*: [|*SSRdia*(*a*,*b*);*LERdia*(*b*,*c*)|] ==> *LERdia*(*a*,*c*)
apply(*unfold LERdia-def*,*unfold SSRdia-def*)
apply(*clarify*)
apply(*drule-tac a=caa* **and** *b=cb* **and** *c=b* **in** *MinBSpR-unique*)
apply(*safe*)
apply(*rule-tac x=ca* **in** *exI*)
apply(*rule-tac x=cba* **in** *exI*)
apply(*safe*)
apply(*drule-tac c=ca* **and** *a=cb* **and** *b=cba* **in** *SSR-and-LER-imp-LER*)
apply(*safe*)
done

theorem *SpR-and-SpR-and-SSR-imp-SSRdia*: [|*SpR*(*a*);*SpR*(*b*);*SSR*(*a*,*b*)|] ==> *SSRdia*(*a*,*b*)
apply(*unfold SSRdia-def*)
apply(*rule-tac x=a* **in** *exI*)
apply(*rule-tac x=b* **in** *exI*)
apply(*safe*)
apply(*simp add*: *SpR-iff-MinBSpR*)
apply(*simp add*: *SpR-iff-MinBSpR*)
done

theorem *SpR-and-SpR-and-SSRdia-imp-SSR*: [|*SpR*(*a*);*SpR*(*b*);*SSRdia*(*a*,*b*)|] ==> *SSR*(*a*,*b*)
apply(*unfold SSRdia-def*)
apply(*clarify*)
apply(*simp add*: *SpR-iff-MinBSpR*)
apply(*drule-tac a=a* **and** *b=ca* **and** *c=a* **in** *MinBSpR-unique*)
apply(*assumption*)
apply(*drule-tac a=b* **and** *b=cb* **and** *c=b* **in** *MinBSpR-unique*)
apply(*assumption*)
apply(*simp*)
done

theorem *SpR-and-SpR-imp-SSR-iff-SSRdia*: [|*SpR*(*a*);*SpR*(*b*)|] ==> (*SSR*(*a*,*b*) <−> *SSRdia*(*a*,*b*))
apply(*safe*)
apply(*rule SpR-and-SpR-and-SSR-imp-SSRdia*)
apply(*auto*)
apply(*rule SpR-and-SpR-and-SSRdia-imp-SSR*)
apply(*auto*)
done

```
theorem SSRdia-imp-LERdia: SSRdia(a,b) ==> LERdia(a,b)
apply(unfold LERdia-def)
apply(unfold SSRdia-def)
apply(clarify)
apply(rule-tac x=ca in exI)
apply(rule-tac x=cb in exI)
apply(safe)
apply(drule SSR-imp-LER)
apply(auto)
done
```

```
consts

RSSRdia :: Rg => Rg => o
NEGRdia :: Rg => Rg => o

defs

RSSRdia-def: RSSRdia(a,b) == (EX ca cb. (MinBSpR(ca,a) & MinBSpR(cb,b)
& RSSR(ca,cb)))
NEGRdia-def: NEGRdia(a,b) == (EX ca cb. (MinBSpR(ca,a) & MinBSpR(cb,b)
& NEGR(ca,cb)))
```

```
theorem RSSRdia-refl: RSSRdia(a,a)
apply(unfold RSSRdia-def)
apply(insert MinBSpR-exists [of a])
apply(insert MinBSpR-exists [of a])
apply(clarify)
apply(rule-tac x=b in exI)
apply(rule-tac x=ba in exI)
apply(safe)
apply(drule-tac a=b and b=ba and c=a in  MinBSpR-unique)
apply(insert RSSR-refl)
apply(auto)
done

theorem RSSRdia-sym: RSSRdia(a,b) ==> RSSRdia(b,a)
apply(unfold RSSRdia-def)
apply(clarify)
apply(rule-tac x=cb in exI)
apply(rule-tac x=ca in exI)
```

apply(*safe*)
apply(*rule-tac b=cb* **and** *a=ca* **in** *RSSR-sym*)
apply(*assumption*)
done

theorem *RSSRdia-between*: [|*RSSRdia*(*a*,*b*);*LERdia*(*a*,*c*);*LERdia*(*c*,*b*)|] ==> (*RSSRdia*(*c*,*a*) & *RSSRdia*(*c*,*b*))
apply(*unfold RSSRdia-def*,*unfold LERdia-def*)
apply(*clarify*)
apply(*drule-tac a=ca* **and** *b=caa* **and** *c=a* **in** *MinBSpR-unique*)
apply(*assumption*)
apply(*drule-tac a=cb* **and** *b=cbb* **and** *c=b* **in** *MinBSpR-unique*)
apply(*assumption*)
apply(*drule-tac a=cab* **and** *b=cba* **and** *c=c* **in** *MinBSpR-unique*)
apply(*assumption*)
apply(*safe*)
apply(*rule-tac x=cba* **in** *exI*)
apply(*rule-tac x=caa* **in** *exI*)
apply(*safe*)
apply(*drule-tac a=caa* **and** *b=cbb* **and** *c=cba* **in** *RSSR-between*)
apply(*safe*)
apply(*rule-tac x=cba* **in** *exI*)
apply(*rule-tac x=cbb* **in** *exI*)
apply(*safe*)
apply(*drule-tac a=caa* **and** *b=cbb* **and** *c=cba* **in** *RSSR-between*)
apply(*safe*)
done

theorem *SSRdia-and-RSSRdia-imp-RSSRdia*: [|*SSRdia*(*a*,*b*);*RSSRdia*(*b*,*c*)|] ==> *RSSRdia*(*a*,*c*)
apply(*unfold SSRdia-def*,*unfold RSSRdia-def*)
apply(*clarify*)
apply(*drule-tac a=caa* **and** *b=cb* **and** *c=b* **in** *MinBSpR-unique*)
apply(*assumption*)
apply(*rule-tac x=ca* **in** *exI*)
apply(*rule-tac x=cba* **in** *exI*)
apply(*simp*)
apply(*rule-tac a=ca* **and** *b=cb* **and** *c=cba* **in** *SSR-and-RSSR-imp-RSSR*)
apply(*auto*)
done

theorem *RSSRdia-and-SSRdia-imp-RSSRdia*: [|*RSSRdia*(*a*,*b*);*SSRdia*(*b*,*c*)|] ==> *RSSRdia*(*a*,*c*)
apply(*unfold SSRdia-def*,*unfold RSSRdia-def*)
apply(*clarify*)
apply(*drule-tac a=caa* **and** *b=cb* **and** *c=b* **in** *MinBSpR-unique*)

apply(*assumption*)
apply(*rule-tac x=ca* **in** *exI*)
apply(*rule-tac x=cba* **in** *exI*)
apply(*simp*)
apply(*rule-tac a=ca* **and** *b=cb* **and** *c=cba* **in** *RSSR-and-SSR-imp-RSSR*)
apply(*auto*)
done

theorem *SSRdia-imp-RSSRdia*: *SSRdia*(*a*,*b*) ==> *RSSRdia*(*a*,*b*)
apply(*insert RSSRdia-refl*)
apply(*insert RSSRdia-and-SSRdia-imp-RSSRdia*)
apply(*auto*)
done

theorem *NEGRdia-imp-LERdia*: *NEGRdia*(*a*,*b*) ==> *LERdia*(*a*,*b*)
apply(*unfold NEGRdia-def*)
apply(*unfold LERdia-def*)
apply(*clarify*)
apply(*rule-tac x=ca* **in** *exI*)
apply(*rule-tac x=cb* **in** *exI*)
apply(*safe*)
apply(*drule-tac a=ca* **and** *b=cb* **in** *NEGR-imp-LER*)
apply(*safe*)
done

theorem *NEGRdia-imp-LERdia-and-notSSRdia*: *NEGRdia*(*a*,*b*) ==> (*LERdia*(*a*,*b*) & ~*SSRdia*(*a*,*b*))
apply(*unfold NEGRdia-def*)
apply(*unfold LERdia-def*)
apply(*unfold SSRdia-def*)
apply(*safe*)
apply(*rule-tac x=ca* **in** *exI*)
apply(*rule-tac x=cb* **in** *exI*)
apply(*safe*)
apply(*drule-tac a=ca* **and** *b=cb* **in** *NEGR-imp-LER-and-notSSR*)
apply(*safe*)
apply(*drule-tac a=ca* **and** *b=caa* **and** *c=a* **in** *MinBSpR-unique*)
apply(*assumption*)
apply(*drule-tac a=cb* **and** *b=cba* **and** *c=b* **in** *MinBSpR-unique*)
apply(*assumption*)
apply(*drule-tac a=ca* **and** *b=cb* **in** *NEGR-imp-LER-and-notSSR*)
apply(*safe*)
done

theorem *NEGRdia-irrefl*: ~*NEGRdia*(*a*,*a*)
apply(*unfold NEGRdia-def*)
apply(*safe*)
apply(*drule-tac a=ca* **and** *b=cb* **and** *c=a* **in** *MinBSpR-unique*)
apply(*safe*)

apply(*insert NEGR-irrefl*)
apply(*erule-tac x=cb* **in** *allE*)
apply(*safe*)
done

theorem *NEGRdia-assym*: *NEGRdia*(*a*,*b*) ==> ~*NEGRdia*(*b*,*a*)
apply(*unfold NEGRdia-def*)
apply(*safe*)
apply(*drule-tac a=ca* **and** *b=cba* **and** *c=a* **in** *MinBSpR-unique*)
apply(*assumption*)
apply(*drule-tac a=cb* **and** *b=caa* **and** *c=b* **in** *MinBSpR-unique*)
apply(*safe*)
apply(*drule-tac a=cba* **and** *b=caa* **in** *NEGR-assym*)
apply(*safe*)
done

theorem *LERdia-and-NEGRdia-imp-NEGRdia*: [|*LERdia*(*a*,*b*);*NEGRdia*(*b*,*c*)|] ==> *NEGRdia*(*a*,*c*)
apply(*unfold NEGRdia-def*)
apply(*unfold LERdia-def*)
apply(*clarify*)
apply(*drule-tac a=caa* **and** *b=cb* **and** *c=b* **in** *MinBSpR-unique*)
apply(*safe*)
apply(*rule-tac x=ca* **in** *exI*)
apply(*rule-tac x=cba* **in** *exI*)
apply(*safe*)
apply(*drule-tac a=ca* **and** *b=cb* **and** *c=cba* **in** *LER-and-NEGR-imp-NEGR*)
apply(*safe*)
done

theorem *NEGRdia-and-LERdia-imp-NEGRdia*: [|*NEGRdia*(*a*,*b*);*LERdia*(*b*,*c*)|] ==> *NEGRdia*(*a*,*c*)
apply(*unfold NEGRdia-def*)
apply(*unfold LERdia-def*)
apply(*clarify*)
apply(*drule-tac a=caa* **and** *b=cb* **and** *c=b* **in** *MinBSpR-unique*)
apply(*safe*)
apply(*rule-tac x=ca* **in** *exI*)
apply(*rule-tac x=cba* **in** *exI*)
apply(*safe*)
apply(*drule-tac a=ca* **and** *b=cb* **and** *c=cba* **in** *NEGR-and-LER-imp-NEGR*)
apply(*safe*)
done

theorem *SSRdia-and-NEGRdia-imp-NEGRdia*: [|*SSRdia*(*a*,*b*);*NEGRdia*(*b*,*c*)|] ==> *NEGRdia*(*a*,*c*)
apply(*drule SSRdia-imp-LERdia*)
apply(*drule LERdia-and-NEGRdia-imp-NEGRdia*)
apply(*auto*)

done

theorem *NEGRdia-SSRdia-imp-NEGRdia*: [|*NEGRdia*(*a*,*b*);*SSRdia*(*b*,*c*)|] ==> *NEGRdia*(*a*,*c*)
apply(*drule SSRdia-imp-LERdia* [*of b c*])
apply(*drule NEGRdia-and-LERdia-imp-NEGRdia*)
apply(*auto*)
done

theorem *NEGRdia-trans*: [|*NEGRdia*(*a*,*b*);*NEGRdia*(*b*,*c*)|] ==> *NEGRdia*(*a*,*c*)
apply(*drule NEGRdia-imp-LERdia*)
apply(*drule LERdia-and-NEGRdia-imp-NEGRdia*)
apply(*auto*)
done

theorem *PR-and-NEGRdia-imp-NEGRdia*: [|*PR*(*a*,*b*);*NEGRdia*(*b*,*c*)|] ==> *NEGRdia*(*a*,*c*)
apply(*drule PR-imp-LERdia*)
apply(*drule LERdia-and-NEGRdia-imp-NEGRdia*)
apply(*auto*)
done

theorem *NEGRdia-PR-imp-NEGRdia*: [|*NEGRdia*(*a*,*b*);*PR*(*b*,*c*)|] ==> *NEGRdia*(*a*,*c*)
apply(*drule PR-imp-LERdia* [*of b c*])
apply(*drule NEGRdia-and-LERdia-imp-NEGRdia*)
apply(*auto*)
done

theorem *SpR-and-SpR-and-RSSR-imp-RSSRdia*: [|*SpR*(*a*);*SpR*(*b*);*RSSR*(*a*,*b*)|] ==> *RSSRdia*(*a*,*b*)
apply(*unfold RSSRdia-def*)
apply(*rule-tac x=a* **in** *exI*)
apply(*rule-tac x=b* **in** *exI*)
apply(*safe*)
apply(*simp add*: *SpR-iff-MinBSpR*)
apply(*simp add*: *SpR-iff-MinBSpR*)
done

theorem *SpR-and-SpR-and-RSSRdia-imp-RSSR*: [|*SpR*(*a*);*SpR*(*b*);*RSSRdia*(*a*,*b*)|] ==> *RSSR*(*a*,*b*)
apply(*unfold RSSRdia-def*)
apply(*clarify*)
apply(*simp add*: *SpR-iff-MinBSpR*)
apply(*drule-tac a=a* **and** *b=ca* **and** *c=a* **in** *MinBSpR-unique*)
apply(*assumption*)
apply(*drule-tac a=b* **and** *b=cb* **and** *c=b* **in** *MinBSpR-unique*)
apply(*assumption*)

apply(*simp*)
done

theorem *SpR-and-SpR-imp-RSSR-iff-RSSRdia*: [|*SpR*(*a*);*SpR*(*b*)|] ==> (*RSSR*(*a*,*b*) <−> *RSSRdia*(*a*,*b*))
apply(*safe*)
apply(*rule SpR-and-SpR-and-RSSR-imp-RSSRdia*)
apply(*auto*)
apply(*rule SpR-and-SpR-and-RSSRdia-imp-RSSR*)
apply(*auto*)
done

end

theory *QDistR*

imports *QSizeR RBG QDiaSizeR*

begin

consts

CLR :: *Rg* => *Rg* => *o*
SCLR :: *Rg* => *Rg* => *o*
NR :: *Rg* => *Rg* => *o*
SNR :: *Rg* => *Rg* => *o*
AR :: *Rg* => *Rg* => *o*
FAR :: *Rg* => *Rg* => *o*
MAR :: *Rg* => *Rg* => *o*

SpShR :: *Rg* => *o*

defs

CLR-def: *CLR*(*a*,*b*) == (*EX c*. (*SpR*(*c*) & *CGR*(*c*,*a*) & *CGR*(*c*,*b*) & *NEGR*(*c*,*a*)))
SCLR-def: *SCLR*(*a*,*b*) == ~*CGR*(*a*,*b*) & *CLR*(*a*,*b*)
NR-def: *NR*(*a*,*b*) == (*EX c*. (*SpR*(*c*) & *CGR*(*c*,*a*) & *CGR*(*c*,*b*) & (*NEGR*(*c*,*a*) | *RSSR*(*c*,*a*))))
SNR-def: *SNR*(*a*,*b*) == ~*CLR*(*a*,*b*) & *NR*(*a*,*b*)
AR-def: *AR*(*a*,*b*) == ~*NR*(*a*,*b*)
FAR-def: *FAR*(*a*,*b*) == (*ALL c*. (*SpR*(*c*) & *CGR*(*c*,*a*) & *CGR*(*c*,*b*) −−> (*NEGR*(*a*,*c*))))
MAR-def: *MAR*(*a*,*b*) == *AR*(*a*,*b*) & ~ *FAR*(*a*,*b*)

SpShR-def: *SpShR*(*a*) == (*EX b*. (*MinBSpR*(*b*,*a*) & *RSSR*(*a*,*b*)))

axioms

PR-imp-NR-imp-NR: *PR*(*a*,*b*) ==> (*ALL c*. (*NR*(*a*,*c*) −−> *NR*(*b*,*c*)))
LRSSR-and-NR-imp-NR: [|*LRSSR*(*a*,*b*);*NR*(*a*,*b*)|] ==> *NR*(*b*,*a*)

lemma *SSR-or-notSSR*: *ALL a b*. (*SSR*(*a*,*b*) | ~*SSR*(*a*,*b*))
apply(*insert excluded-middle* [*of SSR*(*a*,*b*)])
apply(*auto*)
done

theorem *CGR-imp-CLR*: *CGR*(*a*,*b*) ==> *CLR*(*a*,*b*)
apply(*unfold CLR-def*)
apply(*unfold CGR-def*)
apply(*clarify*)
apply(*fold CGR-def*)
apply(*drule-tac a*=*c* **in** *SpR-CoPPR-NEGR-exists*)
apply(*clarify*)
apply(*insert LER-total*)
apply(*rotate-tac 5*)
apply(*erule-tac x*=*a* **in** *allE*)
apply(*rotate-tac 5*)
apply(*erule-tac x*=*c* **in** *allE*)
apply(*erule disjE*)
prefer *2*
apply(*drule-tac a*=*ba* **and** *b*=*c* **and** *c*=*a* **in** *NEGR-and-LER-imp-NEGR*)
apply(*assumption*)
apply(*erule-tac x*=*ba* **in** *allE*)
apply(*clarify*)
apply(*drule-tac a*=*ba* **and** *b*=*a* **in** *OR-imp-CGR*)
apply(*drule-tac a*=*ba* **and** *b*=*b* **in** *OR-imp-CGR*)
apply(*rule-tac x*=*ba* **in** *exI*)
apply(*drule-tac a*=*ba* **and** *b*=*c* **in** *CoPPR-imp-SpR-and-SpR*)
apply(*safe*)
apply(*insert SSR-or-notSSR*)
apply(*rotate-tac 6*)
apply(*erule-tac x*=*a* **in** *allE*)
apply(*rotate-tac 6*)
apply(*erule-tac x*=*c* **in** *allE*)
apply(*safe*)
apply(*drule-tac a*=*a* **and** *b*=*c* **in** *SSR-sym*)
apply(*drule-tac a*=*c* **and** *b*=*a* **in** *SSR-imp-LER*)
apply(*drule-tac a*=*ba* **and** *b*=*c* **and** *c*=*a* **in** *NEGR-and-LER-imp-NEGR*)
apply(*assumption*)
apply(*erule-tac x*=*ba* **in** *allE*)
apply(*clarify*)

apply(*drule-tac a=ba* **and** *b=a* **in** *OR-imp-CGR*)
apply(*drule-tac a=ba* **and** *b=b* **in** *OR-imp-CGR*)
apply(*rule-tac x=ba* **in** *exI*)
apply(*drule-tac a=ba* **and** *b=c* **in** *CoPPR-imp-SpR-and-SpR*)
apply(*safe*)
apply(*drule-tac a=ba* **and** *b=c* **in** *CoPPR-imp-SpR-and-SpR*)
apply(*clarify*)
apply(*drule-tac a=c* **and** *b=a* **in** *SpR-and-LER-and-notSSR-imp-CoPPR-and-SSR-exists*)
apply(*assumption*)+
apply(*clarify*)
apply(*frule-tac a=ca* **and** *b=c* **in** *CoPPR-imp-SpR-and-SpR*)
apply(*clarify*)
apply(*drule-tac a=ca* **in** *SpR-CoPPR-NEGR-exists*)
apply(*clarify*)
apply(*drule-tac a=baa* **and** *b=ca* **and** *c=c* **in** *CoPPR-trans*)
apply(*assumption*)
apply(*erule-tac x=baa* **in** *allE*)
apply(*clarify*)
apply(*drule-tac a=ca* **and** *b=a* **in** *SSR-imp-LER*)
apply(*drule-tac a=baa* **and** *b=ca* **and** *c=a* **in** *NEGR-and-LER-imp-NEGR*)
apply(*assumption*)
apply(*drule-tac a=baa* **and** *b=a* **in** *OR-imp-CGR*)
apply(*drule-tac a=baa* **and** *b=b* **in** *OR-imp-CGR*)
apply(*rule-tac x=baa* **in** *exI*)
apply(*drule-tac a=baa* **and** *b=c* **in** *CoPPR-imp-SpR-and-SpR*)
apply(*safe*)
done

theorem *SCLR-imp-CLR*: *SCLR*(*a*,*b*) ==> *CLR*(*a*,*b*)
apply(*unfold SCLR-def*)
apply(*auto*)
done

theorem *CLR-imp-CGR-or-SCLR*: *CLR*(*a*,*b*) ==> (*CGR*(*a*,*b*) | *SCLR*(*a*,*b*))
apply(*unfold SCLR-def*,*unfold CLR-def*)
apply(*auto*)
done

theorem *CGR-imp-notSCLR*: *CGR*(*a*,*b*) ==> ~*SCLR*(*a*,*b*)
apply(*unfold SCLR-def*)
apply(*auto*)
done

theorem *CLR-imp-NR*: *CLR*(*a*,*b*) ==> *NR*(*a*,*b*)
apply(*unfold CLR-def*,*unfold NR-def*)
apply(*clarify*)
apply(*rule-tac x=c* **in** *exI*)
apply(*safe*)
done

theorem *SNR-imp-NR*: *SNR*(a,b) ==> *NR*(a,b)
apply(*unfold SNR-def*)
apply(*auto*)
done

theorem *NR-imp-CLR-or-SNR*: *NR*(a,b) ==> (*CLR*(a,b) | *SNR*(a,b))
apply(*unfold SNR-def*,*unfold CLR-def*)
apply(*clarify*)
done

theorem *CLR-imp-notSNR*: *CLR*(a,b) ==> ~*SNR*(a,b)
apply(*unfold SNR-def*)
apply(*auto*)
done

theorem *FAR-imp-AR*: *FAR*(a,b) ==> *AR*(a,b)
apply(*rule ccontr*)
apply(*unfold AR-def*)
apply(*auto*)
apply(*unfold FAR-def*,*unfold NR-def*)
apply(*safe*)
apply(*erule-tac* x=c **in** *allE*)
apply(*safe*)
apply(*drule-tac* a=c **and** b=a **in** *NEGR-assym*)
apply(*safe*)
apply(*erule-tac* x=c **in** *allE*)
apply(*safe*)
apply(*drule-tac* a=c **and** b=a **in** *RSSR-sym*)
apply(*drule-tac* a=a **and** b=c **in** *RSSR-imp-notNEGR*)
apply(*safe*)
done

theorem *MAR-imp-AR*: *MAR*(a,b) ==> *AR*(a,b)
apply(*unfold MAR-def*)
apply(*auto*)
done

theorem *AR-imp-MAR-or-FAR*: *AR*(a,b) ==> (*MAR*(a,b) | *FAR*(a,b))
apply(*unfold MAR-def*,*unfold FAR-def*)
apply(*clarify*)
apply(*erule-tac* x=c **in** *allE*)
apply(*auto*)
done

theorem *MAR-imp-notFAR*: *MAR*(a,b) ==> ~*FAR*(a,b)

apply(*unfold MAR-def*)
apply(*auto*)
done

theorem *NR-or-AR*: *NR*(*a*,*b*) | *AR*(*a*,*b*)
apply(*unfold AR-def*)
apply(*auto*)
done

theorem *CLR-refl*: (*ALL a. CLR*(*a*,*a*))
apply(*unfold CLR-def*)
apply(*clarify*)
apply(*insert SP-PPR-exists*)
apply(*drule allI*)
apply(*erule-tac x*=*a***in** *allE*)
apply(*clarify*)
apply(*frule-tac a*=*b* **in** *SpR-CoPPR-NEGR-exists*)
apply(*clarify*)
apply(*frule-tac a*=*ba* **and** *b*=*b* **in** *CoPPR-imp-SpR-and-SpR*)
apply(*drule-tac a*=*b* **and** *b*=*a* **in** *PPR-imp-PR*)
apply(*drule-tac a*=*ba* **and** *b*=*b* **and** *c*=*a* **in** *NEGR-and-PR-imp-NEGR*)
apply(*safe*)
apply(*rule-tac x*=*ba* **in** *exI*)
apply(*drule-tac a*=*ba* **and** *b*=*b* **in** *CoPPR-imp-PPR*)
apply(*drule-tac a*=*ba* **and** *b*=*b* **in** *PPR-imp-PR*)
apply(*drule-tac a*=*ba* **and** *b*=*b* **and** *c*=*a* **in** *PR-trans-rule*)
apply(*assumption*)
apply(*drule-tac a*=*ba* **and** *b*=*a* **in** *PR-imp-CGR*)
apply(*auto*)
done

theorem *NR-refl*: (*ALL a. NR*(*a*,*a*))
apply(*clarify*)
apply(*rule-tac a*=*a* **in** *CLR-imp-NR*)
apply(*insert CLR-refl*)
apply(*auto*)
done

theorem *SCLR-irrefl*: (*ALL a.* ~*SCLR*(*a*,*a*))
apply(*unfold SCLR-def*)
apply(*insert CGR-refl*)
apply(*insert CLR-refl*)
apply(*auto*)
done

theorem *SNR-irrefl*: (*ALL a.* ~*SNR*(*a*,*a*))
apply(*unfold SNR-def*)
apply(*insert CLR-refl*)

apply(*insert NR-refl*)
apply(*auto*)
done

theorem *AR-irrefl*: (*ALL a.* ~*AR*(*a*,*a*))
apply(*unfold AR-def*)
apply(*insert NR-refl*)
apply(*auto*)
done

theorem *FAR-irrefl*: (*ALL a.* ~*FAR*(*a*,*a*))
apply(*insert FAR-imp-AR*,*insert AR-irrefl*)
apply(*auto*)
done

theorem *CLR-and-PR-imp-CLR*: [|*CLR*(*a*,*b*);*PR*(*b*,*c*)|] ==> *CLR*(*a*,*c*)
apply(*unfold CLR-def*)
apply(*clarify*)
apply(*rule-tac x*=*ca* **in** *exI*)
apply(*clarify*)
apply(*drule-tac a*=*ca* **and** *b*=*b* **in** *CGR-sym*)
apply(*drule-tac a*=*b* **and** *b*=*c* **and** *c*=*ca* **in** *PR-and-CGR-imp-CGR*)
apply(*assumption*)
apply(*drule-tac a*=*c* **and** *b*=*ca* **in** *CGR-sym*)
apply(*assumption*)
done

theorem *NR-and-PR-imp-NR*: [|*NR*(*a*,*b*);*PR*(*b*,*c*)|] ==> *NR*(*a*,*c*)
apply(*unfold NR-def*)
apply(*clarify*)
apply(*rule-tac x*=*ca* **in** *exI*)
apply(*safe*)
apply(*drule-tac a*=*ca* **and** *b*=*b* **in** *CGR-sym*)
apply(*drule-tac a*=*b* **and** *b*=*c* **and** *c*=*ca* **in** *PR-and-CGR-imp-CGR*)
apply(*assumption*)
apply(*drule-tac a*=*c* **and** *b*=*ca* **in** *CGR-sym*)
apply(*assumption*)
apply(*drule-tac a*=*ca* **and** *b*=*b* **in** *CGR-sym*)
apply(*drule-tac a*=*b* **and** *b*=*c* **and** *c*=*ca* **in** *PR-and-CGR-imp-CGR*)
apply(*assumption*)
apply(*drule-tac a*=*c* **and** *b*=*ca* **in** *CGR-sym*)
apply(*assumption*)
done

theorem *PR-and-notNR-imp-notNR*: [|*PR*(*a*,*b*);~*NR*(*b*,*c*)|] ==> ~*NR*(*a*,*c*)

apply(*rule ccontr*)
apply(*auto*)
apply(*drule PR-imp-NR-imp-NR [of a b]*)
apply(*auto*)
done

theorem *PR-imp-NR*: *PR*(*a*,*b*) ==> (*NR*(*a*,*b*) & *NR*(*b*,*a*))
apply(*safe*)
apply(*drule PR-imp-CGR [of a b]*)
apply(*drule CGR-imp-CLR [of a b]*)
apply(*drule CLR-imp-NR [of a b]*)
apply(*assumption*)
apply(*drule PR-imp-NR-imp-NR [of a b]*)
apply(*erule-tac x=a* **in** *allE*)
apply(*insert NR-refl*)
apply(*erule-tac x=a* **in** *allE*)
apply(*auto*)
done

theorem *PR-and-AR-imp-AR*: [|*PR*(*a*,*b*);*AR*(*b*,*c*)|] ==> *AR*(*a*,*c*)
apply(*unfold AR-def*)
apply(*rule ccontr*)
apply(*auto*)
apply(*drule PR-and-notNR-imp-notNR [of a b c]*)
apply(*auto*)
done

theorem *LRSSR-and-CLR-imp-CLR*: [|*LRSSR*(*a*,*b*);*CLR*(*a*,*b*)|] ==> *CLR*(*b*,*a*)
apply(*unfold LRSSR-def*)
apply(*safe*)
apply(*unfold CLR-def*)
apply(*clarify*)
apply(*rule-tac x=c* **in** *exI*)
apply(*clarify*)
apply(*drule-tac a=c* **and** *b=a* **and** *c=b* **in** *NEGR-and-LER-imp-NEGR*)
apply(*assumption*,*assumption*)
apply(*clarify*)
apply(*rule-tac x=c* **in** *exI*)
apply(*clarify*)
apply(*drule-tac a=c* **and** *b=a* **and** *c=b* **in** *NEGR-and-RSSR-imp-NEGR*)
apply(*safe*)
done

theorem *LRSSR-and-SCLR-imp-SCLR*: [|*LRSSR*(*a*,*b*);*SCLR*(*a*,*b*)|] ==> *SCLR*(*b*,*a*)
apply(*unfold SCLR-def*)
apply(*safe*)
apply(*drule CGR-sym*)

apply(*safe*)
apply(*rule LRSSR-and-CLR-imp-CLR*)
apply(*auto*)
done

theorem *RSSR-and-SNR-imp-SNR*: [|*RSSR*(a,b);*SNR*(a,b)|] ==> *SNR*(b,a)
apply(*unfold SNR-def*)
apply(*drule RSSR-imp-LRSSR-and-LRSSR*)
apply(*safe*)
apply(*drule-tac* a=b **and** b=a **in** *LRSSR-and-CLR-imp-CLR*)
apply(*auto*)
apply(*rule LRSSR-and-NR-imp-NR*)
apply(*auto*)
done

theorem *LRSSR-and-SNR-imp-NR*: [|*LRSSR*(a,b);*SNR*(a,b)|] ==> *NR*(b,a)
apply(*unfold SNR-def*)
apply(*clarify*)
apply(*rule LRSSR-and-NR-imp-NR*)
apply(*auto*)
done

theorem *LRSSR-and-AR-imp-AR*: [|*LRSSR*(b,a);*AR*(a,b)|] ==> *AR*(b,a)
apply(*unfold AR-def*)
apply(*insert LRSSR-and-NR-imp-NR*)
apply(*auto*)
done

theorem *LRSSR-and-FAR-imp-FAR*: [|*LRSSR*(b,a);*FAR*(a,b)|] ==> *FAR*(b,a)
apply(*unfold LRSSR-def*)
apply(*safe*)
apply(*unfold FAR-def*)
apply(*safe*)
apply(*erule-tac* x=c **in** *allE*)
apply(*safe*)
apply(*drule-tac* a=b **and** b=a **and** c=c **in** *LER-and-NEGR-imp-NEGR*)
apply(*auto*)
apply(*erule-tac* x=c **in** *allE*)
apply(*safe*)
apply(*rule RSSR-and-NEGR-imp-NEGR*)
apply(*auto*)
done

theorem *LRSSR-and-MAR-imp-AR*: [|*LRSSR*(b,a);*MAR*(a,b)|] ==> *AR*(b,a)

apply(*unfold MAR-def*)
apply(*insert LRSSR-and-AR-imp-AR*)
apply(*auto*)
done

theorem *RSSR-and-MAR-imp-MAR*: [|*RSSR*(*a*,*b*);*MAR*(*a*,*b*)|] ==> *MAR*(*b*,*a*)
apply(*unfold MAR-def*)
apply(*drule RSSR-imp-LRSSR-and-LRSSR*)
apply(*safe*)
apply(*rule LRSSR-and-AR-imp-AR*)
apply(*auto*)
apply(*drule-tac a=b* **and** *b=a* **in** *LRSSR-and-FAR-imp-FAR*)
apply(*auto*)
done

end

theory *TNEMO* **imports** *FOL*

begin

typedecl *Ob*
typedecl *Ti*

arities *Ob* :: *term*
Ti :: *term*

consts

O :: *Ob* => *Ob* => *Ti* => *o*
P :: *Ob* => *Ob* => *Ti* => *o*
PP :: *Ob* => *Ob* => *Ti* => *o*
E :: *Ob* => *Ti* => *o*
Me :: *Ob* => *Ob* => *Ti* => *o*

pP :: *Ob* => *Ob* => *o*
pPP :: *Ob* => *Ob* => *o*
pO :: *Ob* => *Ob* => *o*
pMe :: *Ob* => *Ob* => *o*

cP :: *Ob* => *Ob* => *o*

bP :: *Ob* => *Ob* => *o*

axioms

P-exists1: (*ALL x*. (*EX t*. *E*(*x*,*t*)))
P-exists2: (*ALL x y t*. (*P*(*x*,*y*,*t*) -->(*E*(*x*,*t*) & *E*(*y*,*t*))))
P-trans: (*ALL x y z t*. (*P*(*x*,*y*,*t*) & *P*(*y*,*z*,*t*) --> *P*(*x*,*z*,*t*)))
P-ssuppl: (*ALL x y t*. ((*E*(*x*,*t*) & ~*P*(*x*,*y*,*t*)) --> (*EX z*. (*P*(*z*,*x*,*t*) & ~*O*(*z*,*y*,*t*)))))

defs

E-def: *E*(*x*,*t*) == *P*(*x*,*x*,*t*)
O-def: *O*(*x*,*y*,*t*) == (*EX z*. (*P*(*z*,*x*,*t*) & *P*(*z*,*y*,*t*)))
PP-def: *PP*(*x*,*y*,*t*) == *P*(*x*,*y*,*t*) & ~*P*(*y*,*x*,*t*)
Me-def: *Me*(*x*,*y*,*t*) == (*P*(*x*,*y*,*t*) & *P*(*y*,*x*,*t*))

pP-def: *pP*(*x*,*y*) == (*ALL t*. ((*E*(*x*,*t*) | *E*(*y*,*t*)) --> *P*(*x*,*y*,*t*)))
pPP-def: *pPP*(*x*,*y*) == (*ALL t*. ((*E*(*x*,*t*) | *E*(*y*,*t*)) --> *PP*(*x*,*y*,*t*)))
pO-def: *pO*(*x*,*y*) == (*ALL t*. ((*E*(*x*,*t*) | *E*(*y*,*t*)) --> *O*(*x*,*y*,*t*)))
pMe-def: *pMe*(*x*,*y*) == (*ALL t*. ((*E*(*x*,*t*) | *E*(*y*,*t*)) --> *Me*(*x*,*y*,*t*)))

cP-def: *cP*(*x*,*y*) == (*ALL t*. (*E*(*y*,*t*) --> *P*(*x*,*y*,*t*)))
bP-def: *bP*(*x*,*y*) == (*ALL t*. (*E*(*x*,*t*) --> *P*(*x*,*y*,*t*)))

lemma *P-exists2-rule*: *P*(*x*,*y*,*t*) ==>(*E*(*x*,*t*) & *E*(*y*,*t*))
apply(*insert P-exists2*)
apply(*auto*)
done

lemma *P-trans-rule*: [|*P*(*x*,*y*,*t*); *P*(*y*,*z*,*t*)|] ==> *P*(*x*,*z*,*t*)
apply(*insert P-trans*)
apply(*erule allE*,*erule allE*,*erule allE*,*erule allE*)
apply(*rule mp*)
apply(*assumption*)
apply(*rule conjI*)
apply(*assumption*)
apply(*assumption*)
done

lemma *P-Me-rule*: [|*P*(*x*,*y*,*t*);*P*(*y*,*x*,*t*)|] ==> *Me*(*x*,*y*,*t*)
apply(*drule conjI*)
apply(*assumption*)
apply(*fold Me-def*)
apply(*assumption*)
done

lemma *P-ssuppl-rule*: [|$E(x,t)$;~$P(x,y,t)$|] ==> (*EX z*. ($P(z,x,t)$ & ~$O(z,y,t)$)))
apply(*insert P-ssuppl*)
apply(*auto*)
done

lemma *ltb2*: (~(A & ~B) ==> (~A | B))
apply(*auto*)
done

lemma *P-ssuppl-rule-transpos*: ~(*EX z*. ($P(z,x,t)$ & ~$O(z,y,t)$))) ==> (~$E(x,t)$ | $P(x,y,t)$))
apply(*insert P-ssuppl*)
apply(*rule ltb2*)
apply(*rule notI*)
apply(*auto*)
done

theorem *P-refl*: $E(x,t)$ ==> $P(x,x,t)$
apply(*unfold E-def*)
apply(*assumption*)
done

theorem *Me-refl*: $E(x,t)$ ==> $Me(x,x,t)$
apply(*unfold Me-def*)
apply(*unfold E-def*)
apply(*auto*)
done

theorem *Me-exists2*: $Me(x,y,t)$ ==> ($E(x,t)$ & $E(y,t)$)
apply(*unfold Me-def*)
apply(*rule conjI*)
apply(*erule conjE*)
apply(*drule P-exists2-rule*)
apply(*auto*)
apply(*drule P-exists2-rule*)
apply(*auto*)
done

theorem *Me-sym*: $Me(x,y,t)$ ==> $Me(y,x,t)$
apply(*unfold Me-def*)
apply(*auto*)
done

theorem *Me-trans*: [|$Me(x,y,t)$; $Me(y,z,t)$|] ==> $Me(x,z,t)$
apply(*unfold Me-def*)

apply(*rule conjI*)
apply(*erule conjE*,*erule conjE*)
apply(*erule P-trans-rule*)
apply(*assumption*)
apply(*erule conjE*,*erule conjE*)
apply(*erule P-trans-rule*)
apply(*assumption*)
done

theorem *O-refl*: (*ALL x t. E*(*x*,*t*) *−−> O*(*x*,*x*,*t*))
apply(*rule allI*,*rule allI*)
apply(*unfold E-def*)
apply(*unfold O-def*)
apply(*auto*)
done

lemma *O-refl-rule*: *E*(*x*,*t*) *==> O*(*x*,*x*,*t*)
apply(*insert O-refl*)
apply(*auto*)
done

theorem *O-sym*: (*ALL x y t.* (*O*(*x*,*y*,*t*) *−−> O*(*y*,*x*,*t*)))
apply(*unfold O-def*)
apply(*auto*)
done

lemma *O-sym-rule*: *O*(*x*,*y*,*t*) *==> O*(*y*,*x*,*t*)
apply(*insert O-sym*)
apply(*auto*)
done

theorem *O-imp-E-and-E*: *O*(*x*,*y*,*t*) *==>* (*E*(*x*,*t*) & *E*(*y*,*t*))
apply(*rule conjI*)
apply(*unfold O-def*)
apply(*erule exE*)
apply(*erule conjE*)
apply(*drule P-exists2-rule*)
apply(*auto*)
apply(*rotate-tac 1*)
apply(*drule P-exists2-rule*)
apply(*auto*)
done

theorem *PP-imp-P*: (*ALL x y t.* (*PP*(*x*,*y*,*t*) *−−> P*(*x*,*y*,*t*)))
apply(*unfold PP-def*)
apply(*auto*)

done

lemma *PP-imp-P-rule*: $PP(x,y,t)$ ==> $P(x,y,t)$
apply(*unfold PP-def*)
apply(*auto*)
done

theorem *P-imp-Me-or-PP*: $P(x,y,t)$ ==> $(Me(x,y,t) \mid PP(x,y,t))$
apply(*unfold PP-def*,*unfold Me-def*)
apply(*auto*)
done

theorem *PP-or-Me-imp-P*: $(PP(x,y,t) \mid Me(x,y,t))$ ==> $P(x,y,t)$
apply(*unfold Me-def*)
apply(*safe*)
apply(*drule PP-imp-P-rule*)
apply(*assumption*)
done

theorem *PP-asym*: (*ALL x y t*. $PP(x,y,t)$ --> ~$PP(y,x,t)$)
apply(*unfold PP-def*)
apply(*auto*)
done

lemma *PP-asym-rule*: $PP(x,y,t)$ ==> ~$PP(y,x,t)$
apply(*insert PP-asym*)
apply(*auto*)
done

theorem *PP-trans*: (*ALL x y z t*. ($PP(x,y,t)$ & $PP(y,z,t)$-->$PP(x,z,t)$))
apply(*rule allI*, *rule allI*, *rule allI*,*rule allI*)
apply(*rule impI*)
apply(*unfold PP-def*)
apply(*rule conjI*)
apply(*erule conjE*,*erule conjE*,*erule conjE*)
apply(*rule P-trans-rule*)
apply(*assumption*)
apply(*assumption*)
apply(*erule conjE*, *erule conjE*, *erule conjE*)
apply(*drule P-trans-rule*)
apply(*assumption*)
apply(*rule notI*)
apply(*drule P-trans-rule*)
apply(*assumption*)

apply(*erule notE*)
apply(*assumption*)
done

lemma *PP-trans-rule*: **assumes** *p1*: $PP(x,y,t)$ **assumes** *p2*: $PP(y,z,t)$ **shows** $PP(x,z,t)$
apply(*insert PP-trans*)
apply(*erule allE*, *erule allE*,*erule allE*,*erule allE*)
apply(*rule mp*)
apply(*assumption*)
apply(*rule conjI*)
apply(*insert p1*)
apply(*assumption*)
apply(*insert p2*)
apply(*assumption*)
done

theorem *P-and-PP-imp-PP*: $[|P(x,y,t);PP(y,z,t)|] ==> PP(x,z,t)$
apply(*drule P-imp-Me-or-PP*)
apply(*safe*)
apply(*unfold Me-def*)
apply(*clarify*)
prefer *2*
apply(*drule PP-trans-rule* [*of x y t z*])
apply(*assumption*)
apply(*assumption*)
apply(*unfold PP-def*)
apply(*safe*)
apply(*drule P-trans-rule* [*of x y t z*])
apply(*assumption*)
apply(*assumption*)
apply(*drule P-trans-rule* [*of z x t y*])
apply(*auto*)
done

theorem *PP-and-P-imp-PP*: $[|PP(x,y,t);P(y,z,t)|] ==> PP(x,z,t)$
apply(*drule P-imp-Me-or-PP*)
apply(*safe*)
apply(*unfold Me-def*)
apply(*clarify*)
prefer *2*
apply(*drule PP-trans-rule* [*of x y t z*])
apply(*assumption*)
apply(*assumption*)
apply(*unfold PP-def*)
apply(*safe*)
apply(*drule P-trans-rule* [*of x y t z*])
apply(*assumption*)

apply(*assumption*)
apply(*drule P-trans-rule [of y z t x]*)
apply(*auto*)
done

theorem *P-and-notMe-imp-PP*: [|$P(x,y,t)$;~$Me(x,y,t)$|] ==> $PP(x,y,t)$
apply(*unfold Me-def*,*unfold PP-def*)
apply(*auto*)
done

theorem *P-imp-O*: $P(x,y,t)$ ==> $O(x,y,t)$
apply(*unfold O-def*)
apply(*rule exI*)
apply(*rule conjI*)
apply(*drule P-exists2-rule*)
apply(*erule conjE*)
apply(*unfold E-def*)
apply(*assumption*)
apply(*assumption*)
done

theorem *P-and-O*: [|$P(x,y,t)$;$O(x,z,t)$|] ==> $O(y,z,t)$
apply(*unfold O-def*)
apply(*erule exE*)
apply(*rule exI*)
apply(*rule conjI*)
apply(*erule conjE*)
prefer *2*
apply(*erule conjE*)
apply(*assumption*)
apply(*rule P-trans-rule*)
apply(*assumption*)
apply(*assumption*)
done

theorem *O-imp-O-imp-P*: (*ALL x y t*. ($E(x,t)$ & (*ALL z*. ($O(z,x,t)$ --> $O(z,y,t)$)))
--> $P(x,y,t)$))
apply(*rule allI*,*rule allI*,*rule allI*)
apply(*rule impI*)
apply(*erule conjE*)
apply(*rule-tac P*=~$E(x,t)$ **and** *Q*=$P(x,y,t)$ **in** *disjE*)
apply(*rule P-ssuppl-rule-transpos*)
apply(*rule notI*)
apply(*erule exE*)

apply(*erule allE*)
apply(*erule conjE*)
apply(*drule P-imp-O*)
apply(*drule mp*)
apply(*assumption*)
apply(*erule notE*)
apply(*assumption*)
apply(*erule-tac R=P(x,y,t)* **in** *notE*)
apply(*assumption*)
apply(*assumption*)
done

lemma *O-imp-O-imp-P-rule*: [|*E(x,t);(ALL z. (O(z,x,t) --> O(z,y,t)))*|] ==> *P(x,y,t)*
apply(*insert O-imp-O-imp-P*)
apply(*auto*)
done

lemma *ltb1*: (*ALL z.* (*A(z,x,t)* & *B(z,y,t)*))) ==> (*ALL z. A(z,x,t)*) & (*ALL z. B(z,y,t)*)
apply(*auto*)
done

theorem *O-iff-O-iff-Me* : (*ALL x y t.* ((*E(x,t)* & *E(y,t)* & (*ALL z.* (*O(z,x,t)* <-> *O(z,y,t)*)))) <-> *Me(x,y,t)*)))
apply(*rule allI*,*rule allI*,*rule allI*)
apply(*unfold iff-def*)
apply(*rule conjI*)
apply(*rule impI*)
apply(*unfold Me-def*)
apply(*erule conjE*,*erule conjE*)
apply(*drule ltb1*)
apply(*erule conjE*)
apply(*rule conjI*)
apply(*drule O-imp-O-imp-P-rule*)
prefer *2*
apply(*assumption*)
apply(*assumption*)
apply(*drule-tac x=y* **and** *y=x* **in** *O-imp-O-imp-P-rule*)
prefer *2*
apply(*assumption*)
apply(*assumption*)
apply(*auto*)
apply(*drule P-exists2-rule*)
apply(*erule conjE*)
apply(*assumption*)
apply(*drule P-exists2-rule*)
apply(*erule conjE*)

apply(*assumption*)
apply(*drule P-and-O*)
apply(*drule O-sym-rule*)
apply(*assumption*)
apply(*rule O-sym-rule*)
apply(*assumption*)
apply(*drule-tac x=y* **and** *y=x* **in** *P-and-O*)
apply(*drule O-sym-rule*)
apply(*assumption*)
apply(*rule O-sym-rule*)
apply(*assumption*)
done

theorem *P-iff-P-iff-Me*: (*ALL x y t.* ((*E*(*x*,*t*) & *E*(*y*,*t*) & (*ALL z.* (*P*(*z*,*x*,*t*) <−> *P*(*z*,*y*,*t*)))) <−> *Me*(*x*,*y*,*t*)))
apply(*rule allI*,*rule allI*,*rule allI*)
apply(*rule iffI*)
apply(*erule conjE*,*erule conjE*)
apply(*unfold Me-def*)
apply(*unfold E-def*)
apply(*rule conjI*)
apply(*erule-tac x=x* **in** *allE*)
apply(*erule iffE*)
apply(*drule-tac P=P*(*x*, *x*, *t*) **and** *Q=P*(*x*, *y*, *t*) **in** *mp*)
apply(*assumption*)
apply(*assumption*)
apply(*erule-tac x=y* **in** *allE*)
apply(*erule iffE*)
apply(*drule-tac P=P*(*y*, *y*, *t*) **and** *Q=P*(*y*, *x*, *t*) **in** *mp*)
apply(*assumption*)
apply(*assumption*)
apply(*rule conjI*)
apply(*fold E-def*)
apply(*fold Me-def*)
apply(*drule Me-exists2*)
apply(*erule conjE*)
apply(*assumption*)
apply(*rule conjI*)
apply(*drule Me-exists2*)
apply(*erule conjE*)
apply(*assumption*)
apply(*rule allI*)
apply(*rule iffI*)
apply(*unfold Me-def*)
apply(*erule conjE*)
apply(*drule-tac x=z* **and** *y=x* **and** *z=y* **in** *P-trans-rule*)
apply(*assumption*)

apply(*assumption*)
apply(*erule conjE*)
apply(*drule-tac x=z* **and** *y=y* **and** *z=x* **in** *P-trans-rule*)
apply(*auto*)
done

theorem *pP-refl*: *ALL x. pP(x,x)*
apply(*unfold pP-def*)
apply(*unfold E-def*)
apply(*auto*)
done

theorem *pP-trans*: [|*pP(x,y)*;*pP(y,z)*|] ==> *pP(x,z)*
apply(*unfold pP-def*)
apply(*auto*)
apply(*erule-tac x=t* **in** *allE*)
apply(*erule-tac x=t* **in** *allE*)
apply(*drule-tac P=E(x,t)* **and** *Q=E(y,t)* **in** *disjI1*)
apply(*clarify*)
apply(*frule P-exists2-rule*)
apply(*auto*)
apply(*rule-tac x=x* **and** *y=y* **and** *z=z* **and** *t=t* **in** *P-trans-rule*)
apply(*assumption*, *assumption*)
apply(*erule-tac x=t* **in** *allE*)
apply(*erule-tac x=t* **in** *allE*)
apply(*drule-tac P=E(y,t)* **and** *Q=E(z,t)* **in** *disjI2*)
apply(*clarify*)
apply(*frule P-exists2-rule*)
apply(*auto*)
apply(*rule-tac x=x* **and** *y=y* **and** *z=z* **and** *t=t* **in** *P-trans-rule*)
apply(*assumption*, *assumption*)
done

theorem *pPP-asym*: *pPP(x,y)* ==> ~*pPP(y,x)*
apply(*unfold pPP-def*)
apply(*clarify*)
apply(*insert P-exists1*)
apply(*erule-tac x=x* **in** *allE*)
apply(*erule exE*)
apply(*erule-tac x=t* **in** *allE*)
apply(*erule-tac x=t* **in** *allE*)
apply(*drule-tac P=E(x,t)* **and** *Q=E(y,t)* **in** *disjI1*)
apply(*clarify*)
apply(*frule PP-imp-P-rule*)
apply(*drule P-exists2-rule*)
apply(*drule PP-asym-rule*)
apply(*auto*)
done

theorem *pPP-trans*: [|*pPP*(*x*,*y*);*pPP*(*y*,*z*)|] ==> *pPP*(*x*,*z*)
apply(*unfold pPP-def*)
apply(*safe*)
apply(*rule-tac x*=*x* **and** *y*=*y* **and** *t*=*t* **in** *PP-trans-rule*)
apply(*auto*)
apply(*erule-tac x*=*t* **in** *allE*)
apply(*erule-tac x*=*t* **in** *allE*)
apply(*drule-tac P*=*E*(*x*,*t*) **and** *Q*=*E*(*y*,*t*) **in** *disjI1*)
apply(*drule mp*, *assumption*)
apply(*frule PP-imp-P-rule*)
apply(*drule P-exists2-rule*)
apply(*auto*)
apply(*erule-tac x*=*t* **in** *allE*)
apply(*erule-tac x*=*t* **in** *allE*)
apply(*drule-tac P*=*E*(*y*,*t*) **and** *Q*=*E*(*z*,*t*) **in** *disjI2*)
apply(*drule mp*, *assumption*)
apply(*frule PP-imp-P-rule*)
apply(*drule P-exists2-rule*)
apply(*auto*)
apply(*rule-tac x*=*x* **and** *y*=*y* **and** *t*=*t* **in** *PP-trans-rule*)
apply(*auto*)
done

theorem *pPP-imp-pP-and-notpP*: *pPP*(*x*,*y*) ==> (*pP*(*x*,*y*) & ~*pP*(*y*,*x*))
apply(*unfold pPP-def*,*unfold pP-def*)
apply(*unfold PP-def*)
apply(*rule conjI*)
apply(*safe*)
apply(*erule-tac x*=*t* **in** *allE*)
apply(*force*)
apply(*erule-tac x*=*t* **in** *allE*)
apply(*force*)
apply(*insert P-exists1*)
apply(*erule-tac x*=*x* **in** *allE*)
apply(*erule exE*)
apply(*erule-tac x*=*t* **in** *allE*)
apply(*erule-tac x*=*t* **in** *allE*)
apply(*auto*)
done

theorem *pO-refl*: (*ALL x*. *pO*(*x*,*x*))
apply(*unfold pO-def*)

```
apply(insert O-refl)
apply(auto)
done

theorem pO-sym: pO(x,y) ==> pO(y,x)
apply(unfold pO-def)
apply(insert O-sym)
apply(auto)
done

theorem SharedpP-imp-pO: (EX z. (pP(z,x) & pP(z,y))) ==> pO(x,y)
apply(unfold pP-def,unfold pO-def)
apply(safe)
apply(erule-tac x=t in allE)
apply(erule-tac x=t in allE)
apply(drule-tac P=E(z,t) and Q=E(x,t) in disjI2)
apply(clarify)
apply(unfold O-def)
apply(rule-tac x=z in exI)
apply(frule-tac x=z and y=x and t=t in P-exists2-rule)
apply(safe)
apply(rule-tac x=z in exI)
apply(erule-tac x=t in allE)
apply(erule-tac x=t in allE)
apply(drule-tac P=E(z,t) and Q=E(y,t) in disjI2)
apply(clarify)
apply(drule-tac x=z and y=y and t=t in P-exists2-rule)
apply(auto)
done

theorem cP-refl: (ALL x. (cP(x,x)))
apply(unfold cP-def)
apply(insert P-exists1)
apply(unfold E-def)
apply(auto)
done

theorem cP-trans: [|cP(x,y);cP(y,z)|] ==> cP(x,z)
apply(unfold cP-def)
apply(safe)
apply(erule-tac x=t in allE)
apply(erule-tac x=t in allE)
apply(safe)
apply(drule P-exists2-rule)
```

apply(*safe*)
apply(*drule-tac x=x* **and** *y=y* **and** *z=z* **and** *t=t* **in** *P-trans-rule*)
apply(*auto*)
done

theorem *bP-refl*: (*ALL x*. (*bP*(*x*,*x*)))
apply(*unfold bP-def*)
apply(*insert P-exists1*)
apply(*unfold E-def*)
apply(*auto*)
done

theorem *bP-trans*: [|*bP*(*x*,*y*);*bP*(*y*,*z*)|] ==> *bP*(*x*,*z*)
apply(*unfold bP-def*)
apply(*safe*)
apply(*erule-tac x=t* **in** *allE*)
apply(*erule-tac x=t* **in** *allE*)
apply(*safe*)
apply(*drule P-exists2-rule*)
apply(*safe*)
apply(*drule-tac x=x* **and** *y=y* **and** *z=z* **and** *t=t* **in** *P-trans-rule*)
apply(*auto*)
done

theorem *pP-imp-cP-and-bP*: *pP*(*x*,*y*) ==> (*cP*(*x*,*y*) & *bP*(*x*,*y*))
apply(*unfold pP-def*,*unfold cP-def*,*unfold bP-def*)
apply(*safe*)
apply(*erule-tac x=t* **in** *allE*)
apply(*auto*)
done

theorem *cP-and-bP-imp-pP*: [|*cP*(*x*,*y*);*bP*(*x*,*y*)|] ==> *pP*(*x*,*y*)
apply(*unfold pP-def*,*unfold cP-def*,*unfold bP-def*)
apply(*safe*)
apply(*erule-tac x=t* **in** *allE*)
apply(*erule-tac x=t* **in** *allE*)
apply(*auto*)
done

end
theory *TORL*

imports *TNEMO EMR*

begin

consts

L :: Ob => Rg => Ti => o
LocIn :: Ob => Ob => Ti => o
PCoin :: Ob => Ob => Ti => o
ContIn :: Ob => Ob => Ti => o

pLocIn :: Ob => Ob => o
pPCoin :: Ob => Ob => o
pContIn :: Ob => Ob => o

axioms

L-exists: (ALL x t. (E(x,t) <-> (EX a. L(x,a,t))))
L-P-PR: (ALL x y a b t. (L(x,a,t) & L(y,b,t) & P(x,y,t) --> PR(a,b)))
P-and-L-and-L-imp-P: (ALL x y a t. (P(x,y,t) & L(x,a,t) & L(y,a,t) --> P(y,x,t)))

defs

LocIn-def: LocIn(x,y,t) == (EX a b. (L(x,a,t) & L(y,b,t) & PR(a,b)))
PCoin-def: PCoin(x,y,t) == (EX a b. (L(x,a,t) & L(y,b,t) & OR(a,b)))
ContIn-def: ContIn(x,y,t) == LocIn(x,y,t) & ~O(x,y,t)

pLocIn-def: pLocIn(x,y) == (ALL t. ((E(x,t) | E(y,t)) --> LocIn(x,y,t)))
pPCoin-def: pPCoin(x,y) == (ALL t. ((E(x,t) | E(y,t)) --> PCoin(x,y,t)))
pContIn-def: pContIn(x,y) == (ALL t. ((E(x,t) | E(y,t)) --> ContIn(x,y,t)))

lemma L-exists1: E(x,t) ==> (EX a. L(x,a,t))
apply(insert L-exists)
apply(auto)
done

lemma L-exists2: (EX a. L(x,a,t)) ==> E(x,t)
apply(insert L-exists)
apply(auto)
done

lemma L-P-PR-rule: [|L(x,a,t); L(y,b,t); P(x,y,t)|] ==> PR(a,b)
apply(insert L-P-PR)
apply(erule-tac x=x **in** allE)
apply(erule-tac x=y **in** allE)
apply(erule-tac x=a **in** allE)
apply(erule-tac x=b **in** allE)
apply(erule-tac x=t **in** allE)

apply(*auto*)
done

lemma *P-and-L-and-L-imp-P-rule*: $[|P(x,y,t);L(x,a,t);L(y,a,t)|] ==> P(y,x,t)$
apply(*insert P-and-L-and-L-imp-P*)
apply(*auto*)
done

theorem *L-unique*: $[|L(x,a,t);L(x,b,t)|] ==> a=b$
apply(*rule PR-antisym-rule*)
apply(*insert L-P-PR-rule [of x a t x b]*)
apply(*insert L-exists*)
apply(*unfold E-def*)
apply(*auto*)
apply(*insert L-P-PR-rule [of x b t x a]*)
apply(*auto*)
done

theorem *LocIn-imp-E-and-E*: $LocIn(x,y,t) ==> (E(x,t)\ \&\ E(y,t))$
apply(*unfold LocIn-def*)
apply(*insert L-exists*)
apply(*auto*)
done

theorem *P-imp-L*: $P(x,y,t) ==> (EX\ a\ b.\ (L(x,a,t)\ \&\ L(y,b,t)\ \&\ PR(a,b)))$
apply(*frule P-exists2-rule*)
apply(*erule conjE*)
apply(*drule L-exists1*)
apply(*drule L-exists1*)
apply(*clarify*)
apply(*frule-tac x=x* **and** *y=y* **and** *a=a* **and** *b=aa* **and** *t=t* **in** *L-P-PR-rule*)
apply(*auto*)
done

theorem *L-and-L-and-PR-imp-L*: $[|L(x,a,t);L(y,b,t);PR(a,b)|] ==> LocIn(x,y,t)$
apply(*unfold LocIn-def*)
apply(*auto*)
done

theorem *LocIn-imp-PCoin*: $LocIn(x,y,t) ==> PCoin(x,y,t)$
apply(*unfold LocIn-def,unfold PCoin-def*)
apply(*insert PR-imp-OR*)
apply(*auto*)
done

```
theorem E-imp-LocIn: E(x,t) ==> LocIn(x,x,t)
apply(unfold LocIn-def)
apply(drule L-exists1)
apply(erule exE)
apply(rule-tac x=a in exI)
apply(rule-tac x=a in exI)
apply(insert PR-refl)
apply(auto)
done

theorem LocIn-trans: [|LocIn(x,y,t);LocIn(y,z,t)|] ==> LocIn(x,z,t)
apply(unfold LocIn-def)
apply(clarify)
apply(rotate-tac 1)
apply(drule-tac x=y and a=aa and t=t and b=b in L-unique)
apply(assumption)
apply(rule-tac x=a in exI)
apply(rule-tac x=ba in exI)
apply(rule conjI)
apply(assumption)
apply(rule conjI)
apply(assumption)
apply(rule PR-trans-rule)
apply(assumption)
apply(auto)
done

theorem PCoin-sym: PCoin(x,y,t) ==> PCoin(y,x,t)
apply(unfold PCoin-def)
apply(insert OR-sym)
apply(auto)
done

theorem P-imp-LocIn: P(x,y,t) ==> LocIn(x,y,t)
apply(unfold LocIn-def)
apply(frule P-exists2-rule)
apply(erule conjE)
apply(drule L-exists1)
apply(drule-tac x=y in L-exists1)
apply(erule exE)
apply(erule exE)
apply(rule-tac x=a in  exI)
apply(rule-tac x=aa in exI)
apply(auto)
apply(rule-tac a=a and b=aa in L-P-PR-rule)
apply(auto)
done
```

theorem *P-and-LocIn-imp-P*: [|P(x,y,t);*LocIn*(y,x,t)|] ==> P(y,x,t)
apply(*unfold LocIn-def*)
apply(*clarify*)
apply(*frule-tac x*=x **and** y=y **and** t=t **in** *P-imp-L*)
apply(*clarify*)
apply(*drule-tac x*=x **and** a=b **and** b=aa **and** t=t **in** *L-unique*)
apply(*drule-tac x*=y **and** a=a **and** b=ba **and** t=t **in** *L-unique*)
apply(*safe*)
apply(*drule-tac x*=y **and** a=a **and** b=ba **and** t=t **in** *L-unique*)
apply(*safe*)
apply(*drule-tac a*=aa **and** b=ba **in** *PR-antisym-rule*)
apply(*safe*)
apply(*drule-tac x*=x **and** y=y **and** a=ba **and** t=t **in** *P-and-L-and-L-imp-P-rule*)
apply(*auto*)
done

theorem *LocIn-and-P-imp-LocIn*: [|*LocIn*(x,y,t);P(y,z,t)|] ==> *LocIn*(x,z,t)
apply(*insert P-imp-LocIn*)
apply(*insert LocIn-trans*)
apply(*auto*)
done

theorem *P-and-LocIn-imp-LocIn*: [|P(x,y,t);*LocIn*(y,z,t)|] ==> *LocIn*(x,z,t)
apply(*drule P-imp-LocIn*)
apply(*rotate-tac 1*)
apply(*drule LocIn-trans*)
apply(*auto*)
done

theorem *O-imp-PCoin*: O(x,y,t) ==> *PCoin*(x,y,t)
apply(*unfold PCoin-def*)
apply(*unfold O-def*)
apply(*erule exE*)
apply(*erule conjE*)
apply(*frule P-exists2-rule*)
apply(*rotate-tac 1*)
apply(*frule P-exists2-rule*)
apply(*clarify*)
apply(*drule L-exists1*)
apply(*drule L-exists1*)
apply(*drule L-exists1*)
apply(*drule L-exists1*)
apply(*clarify*)
apply(*rule-tac x*=aa **in** *exI*)
apply(*rule-tac x*=ac **in** *exI*)
apply(*auto*)
apply(*unfold OR-def*)
apply(*frule-tac x*=z **and** a=ab **and** y=y **and** b=ac **and** t=t **in** *L-P-PR-rule*)
apply(*auto*)

apply(*drule-tac x=z* **and** *a=ab* **and** *y=x* **and** *b=aa* **and** *t=t* **in** *L-P-PR-rule*)
apply *auto*
done

theorem *pLocIn-imp-pPCoin*: *pLocIn*(*x*,*y*) ==> *pPCoin*(*x*,*y*)
apply(*unfold pLocIn-def*,*unfold pPCoin-def*)
apply(*insert LocIn-imp-PCoin*)
apply(*auto*)
done

theorem *pLocIn-refl*: (*ALL x. pLocIn*(*x*,*x*))
apply(*unfold pLocIn-def*)
apply(*insert E-imp-LocIn*)
apply(*auto*)
done

theorem *pLocIn-trans*: [|*pLocIn*(*x*,*y*);*pLocIn*(*y*,*z*)|] ==> *pLocIn*(*x*,*z*)
apply(*unfold pLocIn-def*)
apply(*rule allI*)
apply(*safe*)
apply(*rule-tac x=x* **and** *y=y* **and** *z=z* **and** *t=t* **in** *LocIn-trans*)
apply(*erule-tac x=t* **in** *allE*)
apply(*drule-tac P=E*(*x*,*t*) **and** *Q=E*(*y*,*t*) **in** *disjI1*)
apply(*safe*)
apply(*erule-tac x=t* **in** *allE*)
apply(*erule-tac x=t* **in** *allE*)
apply(*drule-tac P=E*(*x*,*t*) **and** *Q=E*(*y*,*t*) **in** *disjI1*)
apply(*clarify*)
apply(*drule LocIn-imp-E-and-E*)
apply(*force*)
apply(*rule-tac x=x* **and** *y=y* **and** *z=z* **and** *t=t* **in** *LocIn-trans*)
apply(*erule-tac x=t* **in** *allE*)
apply(*erule-tac x=t* **in** *allE*)
apply(*drule-tac P=E*(*y*,*t*) **and** *Q=E*(*z*,*t*) **in** *disjI2*)
apply(*safe*)
apply(*drule LocIn-imp-E-and-E*)
apply(*force*)
apply(*erule-tac x=t* **in** *allE*)
apply(*erule-tac x=t* **in** *allE*)
apply(*drule-tac P=E*(*y*,*t*) **and** *Q=E*(*z*,*t*) **in** *disjI2*)
apply(*clarify*)
done

theorem *pPCoin-sym*: *pPCoin*(*x*,*y*) ==> *pPCoin*(*y*,*x*)

apply(*unfold pPCoin-def*)
apply(*insert PCoin-sym*)
apply(*auto*)
done

theorem *pP-imp-pLocIn*: *pP*(*x*,*y*) ==> *pLocIn*(*x*,*y*)
apply(*unfold pP-def*,*unfold pLocIn-def*)
apply(*insert P-imp-LocIn*)
apply(*auto*)
done

theorem *pLocIn-and-pP-imp-pLocIn*: [|*pLocIn*(*x*,*y*);*pP*(*y*,*z*)|] ==> *pLocIn*(*x*,*z*)
apply(*unfold pLocIn-def*,*unfold pP-def*)
apply(*safe*)
apply(*rule-tac x*=*x* **and** *y*=*y* **and** *z*=*z* **and** *t*=*t* **in** *LocIn-and-P-imp-LocIn*)
apply(*auto*)
apply(*insert LocIn-imp-E-and-E*)
apply(*auto*)
apply(*rule-tac x*=*x* **and** *y*=*y* **and** *z*=*z* **and** *t*=*t* **in** *LocIn-and-P-imp-LocIn*)
apply(*auto*)
apply(*insert P-exists2-rule*)
apply(*auto*)
done

theorem *pP-and-pLocIn-imp-pLocIn*: [|*pP*(*x*,*y*);*pLocIn*(*y*,*z*)|] ==> *pLocIn*(*x*,*z*)
apply(*unfold pLocIn-def*,*unfold pP-def*)
apply(*safe*)
apply(*rule-tac x*=*x* **and** *y*=*y* **and** *z*=*z* **and** *t*=*t* **in** *P-and-LocIn-imp-LocIn*)
apply(*auto*)
apply(*insert P-exists2-rule*)
apply(*auto*)
apply(*rule-tac x*=*x* **and** *y*=*y* **and** *z*=*z* **and** *t*=*t* **in** *P-and-LocIn-imp-LocIn*)
apply(*insert LocIn-imp-E-and-E*)
apply(*auto*)
done

theorem *pP-and-pLocIn-imp-pP*: [|*pP*(*x*,*y*);*pLocIn*(*y*,*x*)|] ==> *pP*(*y*,*x*)
apply(*unfold pP-def*,*unfold pLocIn-def*)
apply(*insert P-and-LocIn-imp-P*)
apply(*auto*)
done

theorem *pO-imp-pPCoin*: *pO*(*x*,*y*) ==> *pPCoin*(*x*,*y*)
apply(*unfold pO-def*,*unfold pPCoin-def*)
apply(*insert O-imp-PCoin*)
apply(*auto*)
done

theorem *pContIn-imp-pLocIn-and-notpO*: *pContIn*(*x*,*y*) ==> *pLocIn*(*x*,*y*) & ~*pO*(*x*,*y*)
apply(*unfold pContIn-def*,*unfold pLocIn-def*,*unfold pO-def*)
apply(*safe*)
apply(*erule-tac x*=*t* **in** *allE*)
apply(*safe*)
apply(*unfold ContIn-def*)
apply(*safe*)
apply(*erule-tac x*=*t* **in** *allE*)
apply(*safe*)
apply(*insert P-exists1*)
apply(*erule-tac x*=*x* **in** *allE*)
apply(*clarify*)
apply(*erule-tac x*=*t* **in** *allE*)
apply(*erule-tac x*=*t* **in** *allE*)
apply(*safe*)
done

theorem *pContIn-imp-pLocIn-and-notO*: *pContIn*(*x*,*y*) ==> (*pLocIn*(*x*,*y*) & (*ALL* *t*. ~*O*(*x*,*y*,*t*)))
apply(*unfold pContIn-def*,*unfold pLocIn-def*)
apply(*safe*)
apply(*erule-tac x*=*t* **in** *allE*)
apply(*safe*)
apply(*unfold ContIn-def*)
apply(*safe*)
apply(*erule-tac x*=*t* **in** *allE*)
apply(*safe*)
apply(*insert P-exists1*)
apply(*erule-tac x*=*x* **in** *allE*)
apply(*clarify*)
apply(*frule-tac x*=*x* **and** *y*=*y* **and** *t*=*t* **in** *O-imp-E-and-E*)
apply(*clarify*)
apply(*erule-tac x*=*t* **in** *allE*)
apply(*safe*)
done

theorem *pLocIn-and-notO-imp-pContIn*: [|*pLocIn*(*x*,*y*);(*ALL t*. ~*O*(*x*,*y*,*t*))|] ==> *pContIn*(*x*,*y*)
apply(*unfold pContIn-def*,*unfold pLocIn-def*)
apply(*unfold ContIn-def*)
apply(*safe*)
apply(*erule-tac x*=*t* **in** *allE*)
apply(*safe*)
apply(*erule-tac x*=*t* **in** *allE*)
apply(*safe*)
apply(*erule-tac x*=*t* **in** *allE*)
apply(*safe*)
apply(*erule-tac x*=*t* **in** *allE*)

apply(*erule-tac x=t* **in** *allE*)
apply(*safe*)
apply(*erule-tac x=t* **in** *allE*)
apply(*erule-tac x=t* **in** *allE*)
apply(*safe*)
done

end

theory *QSizeO*

imports *TORL QSizeR*

begin

consts

SS :: *Ob* => *Ob* => *Ti* => *o*
LE :: *Ob* => *Ob* => *Ti* => *o*
RSS :: *Ob* => *Ob* => *Ti* => *o*
NEG :: *Ob* => *Ob* => *Ti* => *o*
SSC :: *Ob* => *Ob* => *Ti* => *o*

pSS :: *Ob* => *Ob* => *o*
pLE :: *Ob* => *Ob* => *o*
pRSS :: *Ob* => *Ob* => *o*
pNEG :: *Ob* => *Ob* => *o*
pSSC :: *Ob* => *Ob* => *o*

defs

SS-def: *SS*(*x*,*y*,*t*) == (*EX a b*. (*L*(*x*,*a*,*t*) & *L*(*y*,*b*,*t*) & *SSR*(*a*,*b*)))
LE-def: *LE*(*x*,*y*,*t*) == (*EX a b*. (*L*(*x*,*a*,*t*) & *L*(*y*,*b*,*t*) & *LER*(*a*,*b*)))
RSS-def: *RSS*(*x*,*y*,*t*) == (*EX a b*. (*L*(*x*,*a*,*t*) & *L*(*y*,*b*,*t*) & *RSSR*(*a*,*b*)))
NEG-def: *NEG*(*x*,*y*,*t*) == (*EX a b*. (*L*(*x*,*a*,*t*) & *L*(*y*,*b*,*t*) & *NEGR*(*a*,*b*)))
SSC-def: *SSC*(*x*,*y*,*t*) == (*EX a b*. (*L*(*x*,*a*,*t*) & *L*(*y*,*b*,*t*) & *SSCR*(*a*,*b*)))

pSS-def: *pSS*(*x*,*y*) == (*ALL t*. ((*E*(*x*,*t*) | *E*(*y*,*t*)) --> *SS*(*x*,*y*,*t*)))
pLE-def: *pLE*(*x*,*y*) == (*ALL t*. ((*E*(*x*,*t*) | *E*(*y*,*t*)) --> *LE*(*x*,*y*,*t*)))
pRSS-def: *pRSS*(*x*,*y*) == (*ALL t*. ((*E*(*x*,*t*) | *E*(*y*,*t*)) --> *RSS*(*x*,*y*,*t*)))
pNEG-def: *pNEG*(*x*,*y*) == (*ALL t*. ((*E*(*x*,*t*) | *E*(*y*,*t*)) --> *NEG*(*x*,*y*,*t*)))
pSSC-def: *pSSC*(*x*,*y*) == (*ALL t*. ((*E*(*x*,*t*) | *E*(*y*,*t*)) --> *SSC*(*x*,*y*,*t*)))

theorem *SS-imp-E-and-E*: *SS*(*x*,*y*,*t*) ==> (*E*(*x*,*t*) & *E*(*y*,*t*))

apply(*unfold SS-def*)
apply(*insert L-exists2*)
apply(*auto*)
done

theorem *SS-refl*: *ALL x t.* (*E*(*x*,*t*) *<−>* *SS*(*x*,*x*,*t*))
apply(*unfold SS-def*)
apply(*insert SSR-refl*)
apply(*insert L-exists*)
apply(*auto*)
done

theorem *SS-sym*: *SS*(*x*,*y*,*t*) *==>* *SS*(*y*,*x*,*t*)
apply(*unfold SS-def*)
apply(*insert SSR-sym*)
apply(*auto*)
done

theorem *SS-trans*: [|*SS*(*x*,*y*,*t*);*SS*(*y*,*z*,*t*)|] *==>* *SS*(*x*,*z*,*t*)
apply(*unfold SS-def*)
apply(*clarify*)
apply(*drule-tac x=y* **and** *a=aa* **and** *b=b* **in** *L-unique*)
apply(*assumption*)
apply(*drule sym*)
apply(*frule-tac a=aa* **and** *b=b* **and** *P=% aa. SSR*(*aa*,*ba*) **in** *ssubst*)
apply(*assumption*)
apply(*drule-tac a=a* **and** *b=b* **and** *c=ba* **in** *SSR-trans*)
apply(*clarify*)
apply(*auto*)
done

theorem *P-and-SS-imp-P*: [|*P*(*x*,*y*,*t*);*SS*(*x*,*y*,*t*)|] *==>* *P*(*y*,*x*,*t*)
apply(*unfold SS-def*)
apply(*clarify*)
apply(*frule-tac x=x* **and** *y=y* **and** *a=a* **and** *b=b* **and** *t=t* **in** *L-P-PR-rule*)
apply(*safe*)
apply(*frule-tac a=a* **and** *b=b* **in** *PR-and-SSR-imp-PR*)
apply(*assumption*)
apply(*drule-tac a=a* **and** *b=b* **in** *PR-antisym-rule*)
apply(*safe*)
apply(*rule-tac x=x* **and** *y=y* **and** *a=b* **and** *t=t* **in** *P-and-L-and-L-imp-P-rule*)
apply(*auto*)
done

theorem *P-and-P-imp-SS*: [|*P*(*x*,*y*,*t*);*P*(*y*,*x*,*t*)|] *==>* *SS*(*x*,*y*,*t*)
apply(*unfold SS-def*)
apply(*drule-tac x=x* **and** *y=y* **and** *t=t* **in** *P-imp-L*)

apply(*drule-tac x=y* **and** *y=x* **and** *t=t* **in** *P-imp-L*)
apply(*clarify*)
apply(*drule-tac x=x* **and** *a=a* **and** *b=ba* **and** *t=t* **in** *L-unique*)
apply(*assumption*)
apply(*drule-tac x=y* **and** *a=b* **and** *b=aa* **and** *t=t* **in** *L-unique*)
apply(*safe*)
apply(*rule-tac x=ba* **in** *exI*)
apply(*rule-tac x=aa* **in** *exI*)
apply(*safe*)
apply(*rule-tac a=ba* **and** *b=aa* **in** *PR-and-PR-imp-SSR*)
apply(*auto*)
done

theorem *LE-total*: [|*E*(*x*,*t*);*E*(*y*,*t*)|] ==> (*LE*(*x*,*y*,*t*) | *LE*(*y*,*x*,*t*))
apply(*drule-tac x=x* **and** *t=t* **in** *L-exists1*)
apply(*drule-tac x=y* **and** *t=t* **in** *L-exists1*)
apply(*clarify*)
apply(*unfold LE-def*)
apply(*rule-tac x=a* **in** *exI*)
apply(*rule-tac x=aa* **in** *exI*)
apply(*safe*)
apply(*auto*)
apply(*erule-tac x=aa* **in** *allE*)
apply(*clarify*)
apply(*erule-tac x=a* **in** *allE*)
apply(*safe*)
apply(*insert LER-total*)
apply(*auto*)
done

theorem *LE-and-LE-imp-SS*: [|*LE*(*x*,*y*,*t*);*LE*(*y*,*x*,*t*)|] ==> *SS*(*x*,*y*,*t*)
apply(*unfold LE-def*)
apply(*clarify*)
apply(*drule-tac x=x* **and** *a=a* **and** *b=ba* **and** *t=t* **in** *L-unique*)
apply(*drule-tac x=y* **and** *a=aa* **and** *b=b* **and** *t=t* **in** *L-unique*)
apply(*safe*)
apply(*unfold SS-def*)
apply(*rule-tac x=ba* **in** *exI*)
apply(*rule-tac x=b* **in** *exI*)
apply(*safe*)
apply(*drule-tac x=y* **and** *a=aa* **and** *b=b* **and** *t=t* **in** *L-unique*)
apply(*safe*)
apply(*rule-tac a=ba* **and** *b=b* **in** *LER-and-LER-imp-SSR*)
apply(*auto*)
done

theorem *LE-imp-E-and-E*: *LE*(*x*,*y*,*t*) ==> (*E*(*x*,*t*) & *E*(*y*,*t*))
apply(*unfold LE-def*)
apply(*insert L-exists2*)

apply(*auto*)
done

theorem *LE-refl*: *ALL x t. E*(*x*,*t*) <−> *LE*(*x*,*x*,*t*)
apply(*unfold LE-def*)
apply(*insert L-exists*)
apply(*insert LER-refl*)
apply(*auto*)
apply(*rule-tac x*=*a* **in** *exI*)
apply(*auto*)
done

theorem *LE-trans*: [|*LE*(*x*,*y*,*t*);*LE*(*y*,*z*,*t*)|] ==> *LE*(*x*,*z*,*t*)
apply(*unfold LE-def*)
apply(*clarify*)
apply(*drule-tac x*=*y* **and** *a*=*aa* **and** *b*=*b* **in** *L-unique*)
apply(*assumption*)
apply(*drule sym*)
apply(*frule-tac a*=*aa* **and** *b*=*b* **and** *P*=% *aa. LER*(*aa*,*ba*) **in** *ssubst*)
apply(*assumption*)
apply(*drule-tac a*=*a* **and** *b*=*b* **and** *c*=*ba* **in** *LER-trans*)
apply(*auto*)
done

theorem *LE-and-LE-imp-SS*: [|*LE*(*x*,*y*,*t*);*LE*(*y*,*x*,*t*)|] ==> *SS*(*x*,*y*,*t*)
apply(*unfold LE-def*)
apply(*unfold SS-def*)
apply(*clarify*)
apply(*drule-tac x*=*x* **and** *a*=*a* **and** *b*=*ba* **in** *L-unique*)
apply(*assumption*)
apply(*drule-tac x*=*y* **and** *a*=*aa* **and** *b*=*b* **in** *L-unique*)
apply(*assumption*)
apply(*insert LER-and-LER-imp-SSR*)
apply(*auto*)
done

theorem *SS-and-LE-imp-LE*: [|*SS*(*x*,*y*,*t*);*LE*(*y*,*z*,*t*)|] ==> *LE*(*x*,*z*,*t*)
apply(*unfold LE-def*,*unfold SS-def*)
apply(*clarify*)
apply(*drule-tac x*=*y* **and** *a*=*aa* **and** *b*=*b* **in** *L-unique*)
apply(*assumption*)
apply(*drule sym*)
apply(*frule-tac a*=*aa* **and** *b*=*b* **and** *P*=% *aa. LER*(*aa*,*ba*) **in** *ssubst*)
apply(*assumption*)
apply(*drule-tac c*=*a* **and** *a*=*b* **and** *b*=*ba* **in** *SSR-and-LER-imp-LER*)
apply(*auto*)
done

theorem *LE-and-SS-imp-LE*: [|*LE*(*x*,*y*,*t*);*SS*(*y*,*z*,*t*)|] ==> *LE*(*x*,*z*,*t*)
apply(*unfold LE-def*,*unfold SS-def*)
apply(*clarify*)
apply(*drule-tac x*=*y* **and** *a*=*aa* **and** *b*=*b* **in** *L-unique*)
apply(*assumption*)
apply(*drule sym*)
thm *ssubst*
apply(*drule-tac a*=*aa* **and** *b*=*b* **and** *P*=% *aa*. *SSR*(*aa*,*ba*) **in** *ssubst*)
apply(*assumption*)
apply(*drule-tac a*=*a* **and** *b*=*b* **and** *c*=*ba* **in** *LER-and-SSR-imp-LER*)
apply(*auto*)
done

theorem *RSS-imp-E-and-E*: *RSS*(*x*,*y*,*t*) ==> (*E*(*x*,*t*) & *E*(*y*,*t*))
apply(*unfold RSS-def*)
apply(*insert L-exists2*)
apply(*auto*)
done

theorem *RSS-refl*: *ALL x t*. (*E*(*x*,*t*) <−> *RSS*(*x*,*x*,*t*))
apply(*unfold RSS-def*)
apply(*insert RSSR-refl*)
apply(*insert L-exists*)
apply(*auto*)
done

theorem *RSS-sym*: *RSS*(*x*,*y*,*t*) ==> *RSS*(*y*,*x*,*t*)
apply(*unfold RSS-def*)
apply(*insert RSSR-sym*)
apply(*auto*)
done

theorem *RSS-and-SS-imp-RSS*: [|*RSS*(*x*,*y*,*t*);*SS*(*y*,*z*,*t*)|] ==> *RSS*(*x*,*z*,*t*)
apply(*unfold RSS-def*,*unfold SS-def*)
apply(*clarify*)
apply(*drule-tac x*=*y* **and** *a*=*aa* **and** *b*=*b* **in** *L-unique*)
apply(*assumption*)
apply(*drule sym*)
apply(*frule-tac a*=*aa* **and** *b*=*b* **and** *P*=% *aa*. *SSR*(*aa*,*ba*) **in** *ssubst*)
apply(*assumption*)
apply(*drule-tac a*=*a* **and** *b*=*b* **and** *c*=*ba* **in** *RSSR-and-SSR-imp-RSSR*)
apply(*auto*)
done

theorem *SS-and-RSS-imp-RSS*: [|*SS*(*x*,*y*,*t*);*RSS*(*y*,*z*,*t*)|] ==> *RSS*(*x*,*z*,*t*)

```
apply(unfold RSS-def,unfold SS-def)
apply(clarify)
apply(drule-tac x=y and a=aa and b=b in L-unique)
apply(assumption)
apply(drule sym)
apply(frule-tac a=aa and b=b and P=% aa. RSSR(aa,ba) in ssubst)
apply(assumption)
apply(drule-tac a=a and b=b and c=ba in SSR-and-RSSR-imp-RSSR)
apply(auto)
done


theorem NEG-imp-LE: NEG(x,y,t) ==> LE(x,y,t)
apply(unfold NEG-def,unfold LE-def)
apply(insert NEGR-imp-LER)
apply(auto)
done

theorem NEG-imp-E-and-E: NEG(x,y,t) ==>(E(x,t) & E(y,t))
apply(drule NEG-imp-LE)
apply(drule LE-imp-E-and-E)
apply(auto)
done

theorem NEG-irrefl: ALL x t. ~NEG(x,x,t)
apply(unfold NEG-def)
apply(auto)
apply(drule-tac x=x and a=a and b=b and t=t in L-unique)
apply(clarify)
apply(insert NEGR-irrefl)
apply(auto)
done

theorem NEG-assym: NEG(x,y,t) ==> ~NEG(y,x,t)
apply(unfold NEG-def)
apply(safe)
apply(drule-tac x=x and a=a and b=ba and t=t in L-unique)
apply(assumption)
apply(drule-tac x=y and a=b and b=aa and t=t in L-unique)
apply(assumption)
apply(insert NEGR-assym)
apply(auto)
done


theorem NEG-trans: [|NEG(x,y,t);NEG(y,z,t)|] ==> NEG(x,z,t)
apply(unfold NEG-def)
apply(clarify)
apply(drule-tac x=y and a=aa and b=b in L-unique)
```

apply(*assumption*)
apply(*drule sym*)
apply(*frule-tac a=aa* **and** *b=b* **and** *P=% aa. NEGR*(*aa*,*ba*) **in** *ssubst*)
apply(*assumption*)
apply(*drule-tac a=a* **and** *b=b* **and** *c=ba* **in** *NEGR-trans*)
apply(*auto*)
done

theorem *NEG-and-LE-imp-NEG*: [|*NEG*(*x*,*y*,*t*);*LE*(*y*,*z*,*t*)|] ==> *NEG*(*x*,*z*,*t*)
apply(*unfold NEG-def*,*unfold LE-def*)
apply(*clarify*)
apply(*drule-tac x=y* **and** *a=aa* **and** *b=b* **in** *L-unique*)
apply(*assumption*)
apply(*drule sym*)
apply(*frule-tac a=aa* **and** *b=b* **and** *P=% aa. LER*(*aa*,*ba*) **in** *ssubst*)
apply(*assumption*)
apply(*drule-tac a=a* **and** *b=b* **and** *c=ba* **in** *NEGR-and-LER-imp-NEGR*)
apply(*auto*)
done

theorem *LE-and-NEG-imp-NEG*: [|*LE*(*x*,*y*,*t*);*NEG*(*y*,*z*,*t*)|] ==> *NEG*(*x*,*z*,*t*)
apply(*unfold NEG-def*,*unfold LE-def*)
apply(*clarify*)
apply(*drule-tac x=y* **and** *a=aa* **and** *b=b* **in** *L-unique*)
apply(*assumption*)
apply(*drule sym*)
apply(*frule-tac a=aa* **and** *b=b* **and** *P=% aa. NEGR*(*aa*,*ba*) **in** *ssubst*)
apply(*assumption*)
apply(*drule-tac a=a* **and** *b=b* **and** *c=ba* **in** *LER-and-NEGR-imp-NEGR*)
apply(*auto*)
done

theorem *P-imp-LE*: *P*(*x*,*y*,*t*) ==> *LE*(*x*,*y*,*t*)
apply(*unfold LE-def*)
apply(*drule-tac x=x* **and** *y=y* **and** *t=t* **in** *P-imp-L*)
apply(*safe*)
apply(*rule-tac x=a* **in** *exI*)
apply(*rule-tac x=b* **in** *exI*)
apply(*safe*)
apply(*rule PR-imp-LER*)
apply(*auto*)
done

theorem *NEG-and-P-imp-NEG*: [|*NEG*(*x*,*y*,*t*);*P*(*y*,*z*,*t*)|] ==> *NEG*(*x*,*z*,*t*)
apply(*drule-tac x=y* **and** *y=z* **and** *t=t* **in** *P-imp-LE*)
apply(*rule-tac x=x* **and** *z=z* **and** *t=t* **in** *NEG-and-LE-imp-NEG*)
apply(*auto*)
done

theorem *P-and-NEG-imp-NEG*: [|*P*(*x*,*y*,*t*);*NEG*(*y*,*z*,*t*)|] ==> *NEG*(*x*,*z*,*t*)
apply(*drule-tac x*=*x* **and** *y*=*y* **and** *t*=*t* **in** *P-imp-LE*)
thm *LE-and-NEG-imp-NEG*
apply(*rule-tac x*=*x* **and** *z*=*z* **and** *t*=*t* **in** *LE-and-NEG-imp-NEG*)
apply(*auto*)
done

theorem *SSC-refl*: *ALL x t*. (*E*(*x*,*t*) <−> *SSC*(*x*,*x*,*t*))
apply(*unfold SSC-def*)
apply(*insert SSCR-refl*)
apply(*insert L-exists*)
apply(*auto*)
done

theorem *SSC-sym*: *SSC*(*x*,*y*,*t*) ==> *SSC*(*y*,*x*,*t*)
apply(*unfold SSC-def*)
apply(*insert SSCR-sym*)
apply(*auto*)
done

theorem *pSS-refl*: *ALL x*. *pSS*(*x*,*x*)
apply(*unfold pSS-def*)
apply(*insert SS-refl*)
apply(*auto*)
done

theorem *pSS-sym*: *pSS*(*x*,*y*) ==> *pSS*(*y*,*x*)
apply(*unfold pSS-def*)
apply(*insert SS-sym*)
apply(*auto*)
done

theorem *pSS-trans*: [|*pSS*(*x*,*y*);*pSS*(*y*,*z*)|] ==> *pSS*(*x*,*z*)
apply(*unfold pSS-def*)
apply(*safe*)
apply(*erule-tac x*=*t* **in** *allE*)
apply(*erule-tac x*=*t* **in** *allE*)
apply(*safe*)
apply(*drule-tac x*=*x* **and** *y*=*y* **in** *SS-imp-E-and-E*)
apply(*safe*)
apply(*drule SS-trans*)
apply(*auto*)
apply(*erule-tac x*=*t* **in** *allE*)
apply(*erule-tac x*=*t* **in** *allE*)
apply(*safe*)

apply(*drule-tac x=y* **and** *y=z* **in** *SS-imp-E-and-E*)
apply(*auto*)
apply(*drule SS-trans*)
apply(*auto*)
done

theorem *pP-and-pSS-imp-pP*: [|*pP*(*x*,*y*);*pSS*(*x*,*y*)|] ==> *pP*(*y*,*x*)
apply(*unfold pP-def*, *unfold pSS-def*)
apply(*safe*)
apply(*erule-tac x=t* **in** *allE*)
apply(*erule-tac x=t* **in** *allE*)
apply(*safe*)
apply(*rule-tac y=y* **and** *x=x* **and** *t=t* **in** *P-and-SS-imp-P*)
apply(*safe*)
apply(*erule-tac x=t* **in** *allE*)
apply(*erule-tac x=t* **in** *allE*)
apply(*safe*)
apply(*rule-tac y=y* **and** *x=x* **and** *t=t* **in** *P-and-SS-imp-P*)
apply(*safe*)
done

theorem *pLE-refl*: *ALL x*. *pLE*(*x*,*x*)
apply(*unfold pLE-def*)
apply(*insert LE-refl*)
apply(*auto*)
done

theorem *pLE-trans*: [|*pLE*(*x*,*y*);*pLE*(*y*,*z*)|] ==> *pLE*(*x*,*z*)
apply(*unfold pLE-def*)
apply(*safe*)
apply(*erule-tac x=t* **in** *allE*)
apply(*erule-tac x=t* **in** *allE*)
apply(*safe*)
apply(*drule-tac x=x* **and** *y=y* **in** *LE-imp-E-and-E*)
apply(*safe*)
apply(*drule LE-trans*)
apply(*auto*)
apply(*erule-tac x=t* **in** *allE*)
apply(*erule-tac x=t* **in** *allE*)
apply(*safe*)
apply(*drule-tac x=y* **and** *y=z* **in** *LE-imp-E-and-E*)
apply(*auto*)
apply(*drule LE-trans*)
apply(*auto*)

done

theorem *pLE-and-pLE-imp-pSS*: [|*pLE*(*x*,*y*);*pLE*(*y*,*x*)|] ==> *pSS*(*x*,*y*)
apply(*unfold pLE-def*,*unfold pSS-def*)
apply(*rule allI*)
apply(*erule-tac x*=*t* **in** *allE*)
apply(*erule-tac x*=*t* **in** *allE*)
apply(*safe*)
apply(*insert LE-and-LE-imp-SS*)
apply(*auto*)
done

theorem *pSS-and-pLE-imp-pLE*: [|*pSS*(*x*,*y*);*pLE*(*y*,*z*)|] ==> *pLE*(*x*,*z*)
apply(*unfold pLE-def*,*unfold pSS-def*)
apply(*safe*)
apply(*erule-tac x*=*t* **in** *allE*)
apply(*erule-tac x*=*t* **in** *allE*)
apply(*safe*)
apply(*drule-tac x*=*x* **and** *y*=*y* **in** *SS-imp-E-and-E*)
apply(*safe*)
apply(*drule SS-and-LE-imp-LE*)
apply(*auto*)
apply(*erule-tac x*=*t* **in** *allE*)
apply(*erule-tac x*=*t* **in** *allE*)
apply(*safe*)
apply(*drule-tac x*=*y* **and** *y*=*z* **in** *LE-imp-E-and-E*)
apply(*auto*)
apply(*drule SS-and-LE-imp-LE*)
apply(*auto*)
done

theorem *pLE-and-pSS-imp-pLE*: [|*pLE*(*x*,*y*);*pSS*(*y*,*z*)|] ==> *pLE*(*x*,*z*)
apply(*unfold pLE-def*,*unfold pSS-def*)
apply(*safe*)
apply(*erule-tac x*=*t* **in** *allE*)
apply(*erule-tac x*=*t* **in** *allE*)
apply(*safe*)
apply(*drule-tac x*=*x* **and** *y*=*y* **in** *LE-imp-E-and-E*)
apply(*safe*)
apply(*drule LE-and-SS-imp-LE*)
apply(*auto*)
apply(*erule-tac x*=*t* **in** *allE*)
apply(*erule-tac x*=*t* **in** *allE*)
apply(*safe*)
apply(*drule-tac x*=*y* **and** *y*=*z* **in** *SS-imp-E-and-E*)
apply(*auto*)
apply(*drule LE-and-SS-imp-LE*)
apply(*auto*)
done

```
theorem pRSS-refl: ALL x. pRSS(x,x)
apply(unfold pRSS-def)
apply(insert RSS-refl)
apply(auto)
done

theorem pRSS-sym: pRSS(x,y) ==> pRSS(y,x)
apply(unfold pRSS-def)
apply(insert RSS-sym)
apply(auto)
done


theorem pRSS-and-pSS-imp-pRSS: [|pRSS(x,y);pSS(y,z)|] ==> pRSS(x,z)
apply(unfold pSS-def,unfold pRSS-def)
apply(safe)
apply(erule-tac x=t in allE)
apply(erule-tac x=t in allE)
apply(safe)
apply(drule-tac x=x and y=y in RSS-imp-E-and-E)
apply(safe)
apply(drule RSS-and-SS-imp-RSS)
apply(auto)
apply(erule-tac x=t in allE)
apply(erule-tac x=t in allE)
apply(safe)
apply(drule-tac x=y and y=z in SS-imp-E-and-E)
apply(auto)
apply(drule RSS-and-SS-imp-RSS)
apply(auto)
done


theorem pSS-and-pRSS-imp-pRSS: [|pSS(x,y);pRSS(y,z)|] ==> pRSS(x,z)
apply(unfold pSS-def,unfold pRSS-def)
apply(safe)
apply(erule-tac x=t in allE)
apply(erule-tac x=t in allE)
apply(safe)
apply(drule-tac x=x and y=y in SS-imp-E-and-E)
apply(safe)
apply(drule SS-and-RSS-imp-RSS)
apply(auto)
apply(erule-tac x=t in allE)
apply(erule-tac x=t in allE)
```

apply(*safe*)
apply(*drule-tac x=y* **and** *y=z* **in** *RSS-imp-E-and-E*)
apply(*auto*)
apply(*drule SS-and-RSS-imp-RSS*)
apply(*auto*)
done

theorem *pNEG-irrefl*: *ALL x. ~pNEG(x,x)*
apply(*insert P-exists1*)
apply(*unfold pNEG-def*)
apply(*auto*)
apply(*erule-tac x=x* **in** *allE*)
apply(*erule exE*)
apply(*rule-tac x=t* **in** *exI*)
apply(*safe*)
apply(*insert NEG-irrefl*)
apply(*erule-tac x=x* **in** *allE*)
apply(*erule-tac x=t* **in** *allE*)
apply(*auto*)
done

theorem *pNEG-assym*: *pNEG(x,y) ==> ~pNEG(y,x)*
apply(*unfold pNEG-def*)
apply(*safe*)
apply(*insert P-exists1*)
apply(*erule-tac x=x* **in** *allE*)
apply(*erule exE*)
apply(*erule-tac x=t* **in** *allE*)
apply(*erule-tac x=t* **in** *allE*)
apply(*safe*)
apply(*insert NEG-assym*)
apply(*auto*)
done

theorem *pNEG-trans*: *[|pNEG(x,y);pNEG(y,z)|] ==> pNEG(x,z)*
apply(*unfold pNEG-def*)
apply(*safe*)
apply(*erule-tac x=t* **in** *allE*)
apply(*erule-tac x=t* **in** *allE*)
apply(*safe*)
apply(*drule-tac x=x* **and** *y=y* **in** *NEG-imp-E-and-E*)
apply(*safe*)
apply(*drule NEG-trans*)
apply(*auto*)
apply(*erule-tac x=t* **in** *allE*)
apply(*erule-tac x=t* **in** *allE*)
apply(*safe*)
apply(*drule-tac x=y* **and** *y=z* **in** *NEG-imp-E-and-E*)

```
apply(auto)
apply(drule NEG-trans)
apply(auto)
done

theorem pNEG-and-pLE-imp-pNEG: [|pNEG(x,y);pLE(y,z)|] ==> pNEG(x,z)
apply(unfold pNEG-def,unfold pLE-def)
apply(safe)
apply(erule-tac x=t in allE)
apply(erule-tac x=t in allE)
apply(safe)
apply(drule-tac x=x and y=y in NEG-imp-E-and-E)
apply(safe)
apply(drule NEG-and-LE-imp-NEG)
apply(auto)
apply(erule-tac x=t in allE)
apply(erule-tac x=t in allE)
apply(safe)
apply(drule-tac x=y and y=z in LE-imp-E-and-E)
apply(auto)
apply(drule NEG-and-LE-imp-NEG)
apply(auto)
done

theorem pLE-and-pNEG-imp-pNEG: [|pLE(x,y);pNEG(y,z)|] ==> pNEG(x,z)
apply(unfold pNEG-def,unfold pLE-def)
apply(safe)
apply(erule-tac x=t in allE)
apply(erule-tac x=t in allE)
apply(safe)
apply(drule-tac x=x and y=y in LE-imp-E-and-E)
apply(safe)
apply(drule LE-and-NEG-imp-NEG)
apply(auto)
apply(erule-tac x=t in allE)
apply(erule-tac x=t in allE)
apply(safe)
apply(drule-tac x=y and y=z in NEG-imp-E-and-E)
apply(auto)
apply(drule LE-and-NEG-imp-NEG)
apply(auto)
done

theorem pP-imp-pLE: pP(x,y) ==> pLE(x,y)
apply(unfold pP-def,unfold pLE-def)
apply(insert P-imp-LE)
apply(auto)
done
```

```
theorem pNEG-and-pP-imp-pNEG: [|pNEG(x,y);pP(y,z)|] ==> pNEG(x,z)
apply(drule-tac x=y and y=z  in pP-imp-pLE)
apply(rule-tac x=x and z=z  in pNEG-and-pLE-imp-pNEG)
apply(auto)
done

theorem pP-and-pNEG-imp-pNEG: [|pP(x,y);pNEG(y,z)|] ==> pNEG(x,z)
apply(drule-tac x=x and y=y in pP-imp-pLE)
apply(rule-tac x=x and z=z in pLE-and-pNEG-imp-pNEG)
apply(auto)
done


theorem pSSC-refl: ALL x. pSSC(x,x)
apply(unfold pSSC-def)
apply(insert SSC-refl)
apply(auto)
done

theorem pSSC-sym: pSSC(x,y) ==> pSSC(y,x)
apply(unfold pSSC-def)
apply(insert SSC-sym)
apply(auto)
done


end
theory TMTL

imports TNEMO TORL RBG

begin

consts

C :: Ob => Ob => Ti => o
EC :: Ob => Ob => Ti => o
DC :: Ob => Ob => Ti => o
pC :: Ob => Ob => o
pEC :: Ob => Ob => o
pDC :: Ob => Ob => o


defs

C-def: C(x,y,t) == (EX a b. (L(x,a,t) & L(y,b,t) & CGR(a,b)))
EC-def: EC(x,y,t) == (EX a b. (L(x,a,t) & L(y,b,t) & ECR(a,b)))
DC-def: DC(x,y,t) == (EX a b. (L(x,a,t) & L(y,b,t) & DCR(a,b)))
pC-def: pC(x,y) == (ALL t. ((E(x,t) | E(y,t)) --> C(x,y,t)))
```

pEC-def: *pEC*(*x*,*y*) == (*ALL t*. ((*E*(*x*,*t*) | *E*(*y*,*t*)) --> *EC*(*x*,*y*,*t*)))
pDC-def: *pDC*(*x*,*y*) == (*ALL t*. ((*E*(*x*,*t*) | *E*(*y*,*t*)) --> *DC*(*x*,*y*,*t*)))

theorem *E-imp-C*: *E*(*x*,*t*) ==> *C*(*x*,*x*,*t*)
apply(*unfold C-def*)
apply(*drule L-exists1*)
apply(*erule exE*)
apply(*rule-tac x=a* **in** *exI*)
apply(*rule-tac x=a* **in** *exI*)
apply(*insert CGR-refl*)
apply(*auto*)
done

theorem *C-sym*: *C*(*x*,*y*,*t*) ==> *C*(*y*,*x*,*t*)
apply(*unfold C-def*)
apply(*clarify*)
apply(*rule-tac x=b* **in** *exI*)
apply(*rule-tac x=a* **in** *exI*)
apply(*drule CGR-sym*)
apply(*auto*)
done

theorem *C-imp-E-and-E*: *C*(*x*,*y*,*t*) ==> (*E*(*x*,*t*) & *E*(*y*,*t*))
apply(*unfold C-def*)
apply(*insert L-exists*)
apply(*auto*)
done

theorem *P-imp-C-imp-C*: *P*(*x*,*y*,*t*) ==> (*ALL z*. (*C*(*z*,*x*,*t*) --> *C*(*z*,*y*,*t*)))
apply(*unfold C-def*)
apply(*safe*)
apply(*frule P-exists2-rule*)
apply(*clarify*)
apply(*drule-tac x=y* **and** *t=t* **in** *L-exists1*)
apply(*erule exE*)
apply(*rule-tac x=a* **in** *exI*)
apply(*rule-tac x=aa* **in** *exI*)
apply(*auto*)
apply(*drule-tac x=x* **and** *a=b* **and** *y=y* **and** *b=aa* **and** *t=t* **in** *L-P-PR-rule*)
apply(*safe*)
apply(*drule-tac a=b* **and** *b=aa* **in** *PR-imp-CGR-imp-CGR*)
apply(*erule-tac x=a* **in** *allE*)
apply(*auto*)
done

theorem *L-and-L-and-C-imp-CGR*: [|*L*(*x*,*a*,*t*);*L*(*y*,*b*,*t*);*CGR*(*a*,*b*)|] ==> *C*(*x*,*y*,*t*)

apply(*unfold C-def*)
apply(*auto*)
done

theorem *EC-imp-C-and-notO*: $EC(x,y,t)$ ==> $(C(x,y,t)$ & ~$O(x,y,t))$
apply(*unfold EC-def*,*unfold C-def*,*unfold O-def*,*unfold ECR-def*,*unfold OR-def*)
apply(*clarify*)
apply(*auto*)
apply(*drule P-imp-L*)
apply(*drule P-imp-L*)
apply(*clarify*)
apply(*drule-tac* x=z **and** a=aa **and** b=ab **and** t=t **in** *L-unique*)
apply(*clarify*)
apply(*drule-tac* x=y **and** a=b **and** b=ba **and** t=t **in** *L-unique*)
apply(*clarify*)
apply(*drule-tac* x=x **and** a=a **and** b=bb **and** t=t **in** *L-unique*)
apply(*clarify*)
apply(*auto*)
done

theorem *EC-irrefl*: (*ALL x t*. (~$EC(x,x,t)$))
apply(*unfold EC-def*)
apply(*auto*)
apply(*drule L-unique*)
apply(*assumption*)
apply(*insert ECR-irrefl*)
apply(*auto*)
done

theorem *EC-sym*: $EC(x,y,t)$ ==> $EC(y,x,t)$
apply(*unfold EC-def*)
apply(*insert ECR-sym*)
apply(*auto*)
done

theorem *DC-irrefl*: (*ALL x t*. (~$DC(x,x,t)$))
apply(*unfold DC-def*)
apply(*auto*)
apply(*drule L-unique*)
apply(*assumption*)
apply(*insert DCR-irrefl*)
apply(*auto*)
done

theorem *DC-sym*: *DC*(*x*,*y*,*t*) ==> *DC*(*y*,*x*,*t*)
apply(*unfold DC-def*)
apply(*insert DCR-sym*)
apply(*auto*)
done

theorem *DC-imp-notC*: *DC*(*x*,*y*,*t*) ==> ~*C*(*x*,*y*,*t*)
apply(*unfold C-def*,*unfold DC-def*,*unfold DCR-def*)
apply(*clarify*)
apply(*drule-tac x*=*x* **and** *a*=*a* **and** *b*=*aa* **and** *t*=*t* **in** *L-unique*)
apply(*assumption*)
apply(*drule-tac x*=*y* **and** *a*=*b* **and** *b*=*ba* **and** *t*=*t* **in** *L-unique*)
apply(*auto*)
done

theorem *P-imp-C*: *P*(*x*,*y*,*t*) ==> *C*(*x*,*y*,*t*)
apply(*unfold C-def*)
apply(*drule P-imp-L*)
apply(*clarify*)
apply(*rule-tac x*=*a* **in** *exI*)
apply(*rule-tac x*=*b* **in** *exI*)
apply(*drule PR-imp-CGR*)
apply(*auto*)
done

theorem *O-imp-C*: *O*(*x*,*y*,*t*) ==> *C*(*x*,*y*,*t*)
apply(*unfold O-def*)
apply(*clarify*)
apply(*drule-tac x*=*z* **and** *y*=*x* **and** *t*=*t* **in** *P-imp-L*)
apply(*drule-tac x*=*z* **and** *y*=*y* **and** *t*=*t* **in** *P-imp-L*)
apply(*safe*)
apply(*frule-tac x*=*z* **and** *a*=*a* **and** *b*=*aa* **and** *t*=*t* **in** *L-unique*)
apply(*safe*)
apply(*unfold C-def*)
apply(*rule-tac x*=*b* **in** *exI*)
apply(*rule-tac x*=*ba* **in** *exI*)
apply(*safe*)
apply(*rule-tac a*=*b* **and** *b*=*ba* **in** *OR-imp-CGR*)
apply(*unfold OR-def*)
apply(*auto*)
done

theorem *P-and-C-imp-C*: [|*P*(*x*,*y*,*t*);*C*(*x*,*z*,*t*)|] ==> *C*(*y*,*z*,*t*)
apply(*drule-tac x*=*x* **and** *y*=*y* **and** *t*=*t* **in** *P-imp-C-imp-C*)
apply(*erule-tac x*=*z* **in** *allE*)
apply(*insert C-sym*)
apply(*auto*)

done

theorem *PCoin-imp-C*: *PCoin*(*x*,*y*,*t*) ==> *C*(*x*,*y*,*t*)
apply(*unfold PCoin-def*,*unfold C-def*)
apply(*clarify*)
apply(*rule-tac x*=*a* **in** *exI*)
apply(*rule-tac x*=*b* **in** *exI*)
apply(*drule OR-imp-CGR*)
apply(*auto*)
done

theorem *LocIn-imp-C*: *LocIn*(*x*,*y*,*t*) ==> *C*(*x*,*y*,*t*)
apply(*insert LocIn-imp-PCoin*,*insert PCoin-imp-C*)
apply(*auto*)
done

theorem *PCoin-or-EC-or-notC*: (*ALL x y t*. (*PCoin*(*x*,*y*,*t*) | *EC*(*x*,*y*,*t*) | ~*C*(*x*,*y*,*t*)))
apply(*unfold PCoin-def*,*unfold EC-def*,*unfold C-def*)
apply(*auto*)
apply(*erule-tac x*=*a* **in** *allE*)
apply(*erule-tac x*=*a* **in** *allE*)
apply(*auto*)
apply(*erule-tac x*=*b* **in** *allE*)
apply(*auto*)
apply(*erule-tac x*=*b* **in** *allE*)
apply(*auto*)
apply(*insert ECR-def*)
apply(*auto*)
done

theorem *C-and-LocIn-imp-C*: [|*C*(*x*,*y*,*t*);*LocIn*(*y*,*z*,*t*)|] ==> *C*(*x*,*z*,*t*)
apply(*unfold C-def*,*unfold LocIn-def*)
apply(*clarify*)
apply(*rule-tac x*=*a* **in** *exI*)
apply(*rule-tac x*=*ba* **in** *exI*)
apply(*auto*)
apply(*frule-tac x*=*y* **and** *a*=*aa* **and** *b*=*b* **and** *t*=*t* **in** *L-unique*)
apply(*clarify*)
apply(*insert CGR-sym*)
apply(*insert PR-and-CGR-imp-CGR*)
apply(*auto*)
done

theorem *pEC-irrefl*: *ALL x*. ~*pEC*(*x*,*x*)
apply(*unfold pEC-def*)
apply(*insert ECR-irrefl*)
apply(*unfold EC-def*)

apply(*unfold ECR-def*)
apply(*auto*)
apply(*insert P-exists1*)
apply(*erule-tac x=x* **in** *allE*)
apply(*erule exE*)
apply(*rule-tac x=t* **in** *exI*)
apply(*safe*)
apply(*drule-tac x=x* **and** *a=a* **and** *b=b* **and** *t=t* **in** *L-unique*)
apply(*auto*)
done

theorem *pEC-sym*: *pEC*(*x*,*y*) ==> *pEC*(*y*,*x*)
apply(*unfold pEC-def*)
apply(*insert EC-sym*)
apply(*auto*)
done

theorem *pDC-irrefl*: *ALL x*. ~*pDC*(*x*,*x*)
apply(*unfold pDC-def*)
apply(*insert DC-irrefl*)
apply(*safe*)
apply(*insert P-exists1*)
apply(*rotate-tac 2*)
apply(*erule-tac x=x* **in** *allE*)
apply(*erule exE*)
apply(*erule-tac x=x* **in** *allE*)
apply(*erule-tac x=t* **in** *allE*)
apply(*erule-tac x=t* **in** *allE*)
apply(*safe*)
done

theorem *pDC-sym*: *pDC*(*x*,*y*) ==> *pDC*(*y*,*x*)
apply(*unfold pDC-def*)
apply(*insert DC-sym*)
apply(*auto*)
done

theorem *pDC-imp-notC*: *pDC*(*x*,*y*) ==> (*ALL t*. ~ *C*(*x*,*y*,*t*))
apply(*unfold pDC-def*)
apply(*safe*)
apply(*erule-tac x=t* **in** *allE*)
apply(*frule-tac x=x* **and** *y=y* **and** *t=t* **in** *C-imp-E-and-E*)
apply(*unfold DC-def*)
apply(*auto*)
apply(*unfold C-def*,*unfold DCR-def*)
apply(*clarify*)
apply(*drule-tac x=x* **and** *a=a* **and** *b=aa* **and** *t=t* **in** *L-unique*)
apply(*assumption*)

```
apply(drule-tac x=y and a=b and b=ba and t=t in L-unique)
apply(auto)
done


theorem pDC-imp-not-pC: pDC(x,y) ==>  ~ pC(x,y)
apply(unfold pDC-def,unfold pC-def)
apply(safe)
apply(insert  P-exists1)
apply(erule-tac x=x in allE)
apply(clarify)
apply(erule-tac x=t in allE)
apply(erule-tac x=t in allE)
apply(insert DC-imp-notC)
apply(auto)
done

theorem pP-imp-pC: pP(x,y) ==> pC(x,y)
apply(unfold pP-def,unfold pC-def)
apply(insert P-imp-C)
apply(auto)
done

theorem pPCoin-imp-pC: pPCoin(x,y) ==> pC(x,y)
apply(unfold pPCoin-def,unfold pC-def)
apply(insert PCoin-imp-C)
apply(auto)
done




theorem pC-and-pLocIn-imp-pC: [|pC(x,y);pLocIn(y,z)|] ==> pC(x,z)
apply(unfold pC-def,unfold pLocIn-def)
apply(safe)
apply(rule-tac x=x and y=y and z=z and t=t in C-and-LocIn-imp-C)
apply(auto)
apply(insert C-imp-E-and-E)
apply(auto)
apply(rule-tac x=x and y=y and z=z and t=t in C-and-LocIn-imp-C)
apply(insert LocIn-imp-E-and-E)
apply(auto)
done


theorem pEC-imp-pC-and-notpO: pEC(x,y) ==> (pC(x,y) & ~pO(x,y))
apply(unfold pEC-def,unfold pC-def,unfold pO-def)
apply(insert EC-imp-C-and-notO)
apply(safe)
```

apply(*erule-tac x=t* **in** *allE*)
apply(*force*)
apply(*force*)
apply(*insert P-exists1*)
apply(*erule-tac x=x* **in** *allE*)
apply(*erule exE*)
apply(*auto*)
done

theorem *pP-imp-pC-imp-pC*: *pP(x,y) ==> (ALL z. (pC(z,x) --> pC(z,y)))*
apply(*unfold pP-def,unfold pC-def*)
apply(*safe*)
apply(*erule-tac x=t* **in** *allE*)
apply(*erule-tac x=t* **in** *allE*)
apply(*safe*)
apply(*drule-tac x=z* **and** *y=x* **and** *t=t* **in** *C-imp-E-and-E*)
apply(*safe*)
apply(*drule P-imp-C-imp-C*)
apply(*drule E-imp-C*)
apply(*auto*)
apply(*erule-tac x=t* **in** *allE*)
apply(*erule-tac x=t* **in** *allE*)
apply(*safe*)
apply(*drule P-exists2-rule*)
apply(*auto*)
apply(*drule P-imp-C-imp-C*)
apply(*drule E-imp-C*)
apply(*auto*)
done

theorem *pC-sym*: *pC(x,y) ==> pC(y,x)*
apply(*unfold pC-def*)
apply(*safe*)
apply(*rule-tac x=x* **and** *y=y* **in** *C-sym*)
apply(*auto*)
apply(*rule-tac x=x* **and** *y=y* **in** *C-sym*)
apply(*auto*)
done

theorem *pO-imp-pC*: *pO(x,y) ==> pC(x,y)*
apply(*unfold pO-def,unfold pC-def*)
apply(*insert O-imp-C*)
apply(*auto*)
done

theorem *pP-and-pC-imp-pC*: *[|pP(x,y);pC(x,z)|] ==> pC(y,z)*

apply(*unfold pP-def*,*unfold pC-def*)
apply(*safe*)
apply(*erule-tac x=t* **in** *allE*)
apply(*erule-tac x=t* **in** *allE*)
apply(*safe*)
apply(*drule-tac x=x* **and** *y=y* **and** *t=t* **in** *P-exists2-rule*)
apply(*safe*)
apply(*rule-tac y=y* **and** *z=z* **and** *t=t* **in** *P-and-C-imp-C*)
apply(*auto*)
apply(*erule-tac x=t* **in** *allE*)
apply(*erule-tac x=t* **in** *allE*)
apply(*safe*)
apply(*drule-tac x=x* **and** *y=z* **and** *t=t* **in** *C-imp-E-and-E*)
apply(*safe*)
apply(*rule-tac y=y* **and** *z=z* **and** *t=t* **in** *P-and-C-imp-C*)
apply(*auto*)
done

end

theory *Adjacency*

imports *QSizeR QSizeO RBG TORL TMTL*

begin

consts

rAdj :: *Rg* => *Rg* => *o*

Adj :: *Ob* => *Ob* => *Ti* => *o*
pAdj :: *Ob* => *Ob* => *o*
Att :: *Ob* => *Ob* => *o*

defs

rAdj-def: *rAdj*(*a*,*b*) == ~*CGR*(*a*,*b*) & (*EX c*. (*SpR*(*c*) & *NEGR*(*c*,*a*) & *NEGR*(*c*,*b*) & *CGR*(*c*,*a*) & *CGR*(*c*,*b*)))

Adj-def: *Adj*(*x*,*y*,*t*) == (*EX a b* . (*L*(*x*,*a*,*t*) & *L*(*y*,*b*,*t*) & *rAdj*(*a*,*b*)))
pAdj-def: *pAdj*(*x*,*y*) == (*ALL t*. ((*E*(*x*,*t*) | *E*(*y*,*t*)) --> *Adj*(*x*,*y*,*t*)))

Att-def: *Att*(*x*,*y*) == *pDC*(*x*,*y*) & (*EX z1 z2*. (*pPP*(*z1*,*x*) & *pPP*(*z2*,*y*) & *pNEG*(*z1*,*x*) & *pNEG*(*z2*,*y*) & *pAdj*(*z1*,*z2*)))

theorem *rAdj-irrefl*: *ALL a.* (~*rAdj*(*a*,*a*))
apply(*rule allI*)
apply(*unfold rAdj-def*)
apply(*insert CGR-refl*)
apply(*auto*)
done

theorem *rAdj-sym*: *rAdj*(*a*,*b*) ==> *rAdj*(*b*,*a*)
apply(*unfold rAdj-def*)
apply(*insert CGR-sym*)
apply(*auto*)
done

theorem *rAdj-and-PR-and-PR-and-notCGR-imp-rAdj*: [|*rAdj*(*a*,*b*);*PR*(*a*,*aa*);*PR*(*b*,*bb*);~*CGR*(*aa*,*bb*)|] ==> *rAdj*(*aa*,*bb*)
apply(*unfold rAdj-def*)
apply(*safe*)
apply(*rule-tac x*=*c* **in** *exI*)
apply(*safe*)
apply(*drule-tac a*=*c* **and** *b*=*a* **and** *c*=*aa* **in** *NEGR-and-PR-imp-NEGR*)
apply(*assumption*)
apply(*assumption*)
apply(*drule-tac a*=*c* **and** *b*=*b* **and** *c*=*bb* **in** *NEGR-and-PR-imp-NEGR*)
apply(*assumption*)
apply(*assumption*)
apply(*drule-tac a*=*a* **and** *b*=*aa* **in** *PR-imp-CGR-imp-CGR*)
apply(*erule-tac x*=*c* **in** *allE*)
apply(*safe*)
apply(*drule-tac a*=*b* **and** *b*=*bb* **in** *PR-imp-CGR-imp-CGR*)
apply(*erule-tac x*=*c* **in** *allE*)
apply(*safe*)
done

theorem *Adj-irrefl*: *ALL x t.* (~*Adj*(*x*,*x*,*t*))
apply(*rule allI*,*rule allI*)
apply(*unfold Adj-def*)
apply(*safe*)
apply(*drule-tac x*=*x* **and** *a*=*a* **and** *b*=*b* **and** *t*=*t* **in** *L-unique*)
apply(*auto*)
apply(*insert rAdj-irrefl*)
apply(*auto*)
done

theorem *Adj-sym*: *Adj*(*x*,*y*,*t*) ==> *Adj*(*y*,*x*,*t*)
apply(*unfold Adj-def*)
apply(*safe*)
apply(*rule-tac x=b* **in** *exI*)
apply(*rule-tac x=a* **in** *exI*)
apply(*insert rAdj-sym*)
apply(*auto*)
done

theorem *Adj-imp-notC*: *Adj*(*x*,*y*,*t*) ==> ~*C*(*x*,*y*,*t*)
apply(*unfold Adj-def*)
apply(*clarify*)
apply(*unfold C-def*)
apply(*clarify*)
apply(*drule-tac x=x* **and** *a=a* **and** *b=aa* **and** *t=t* **in** *L-unique*)
apply(*assumption*)
apply(*drule-tac x=y* **and** *a=b* **and** *b=ba* **and** *t=t* **in** *L-unique*)
apply(*assumption*)
apply(*unfold rAdj-def*)
apply(*auto*)
done

theorem *Adj-exists*: *Adj*(*x*,*y*,*t*) ==> (*E*(*x*,*t*) & *E*(*y*,*t*))
apply(*unfold Adj-def*)
apply(*safe*)
apply(*rule L-exists2*)
apply(*auto*)
apply(*rule L-exists2*)
apply(*auto*)
done

theorem *Adj-and-P-and-P-and-notC-imp-Adj*: [|*Adj*(*x*,*y*,*t*);*P*(*x*,*xx*,*t*);*P*(*y*,*yy*,*t*);~*C*(*xx*,*yy*,*t*)|]
==> *Adj*(*xx*,*yy*,*t*)
apply(*unfold Adj-def*)
apply(*clarify*)
apply(*frule-tac x=x* **and** *y=xx* **and** *t=t* **in** *P-exists2-rule*)
apply(*frule-tac x=y* **and** *y=yy* **and** *t=t* **in** *P-exists2-rule*)
apply(*clarify*)
apply(*drule-tac x=xx* **and** *t=t* **in** *L-exists1*)
apply(*drule-tac x=yy* **and** *t=t* **in** *L-exists1*)
apply(*clarify*)
apply(*rule-tac x=aa* **in** *exI*)
apply(*rule-tac x=ab* **in** *exI*)
apply(*safe*)
thm *rAdj-and-PR-and-PR-and-notCGR-imp-rAdj*
apply(*rule-tac a=a* **and** *b=b* **and** *aa=aa* **and** *bb=ab* **in** *rAdj-and-PR-and-PR-and-notCGR-imp-rAdj*)
apply(*safe*)
apply(*drule-tac x=x* **and** *y=xx* **and** *t=t* **and** *a=a* **and** *b=aa* **in** *L-P-PR-rule*)
apply(*assumption*)

apply(*assumption*)
apply(*assumption*)
apply(*drule-tac x=y* **and** *y=yy* **and** *t=t* **and** *a=b* **and** *b=ab* **in** *L-P-PR-rule*)
apply(*assumption*)
apply(*assumption*)
apply(*assumption*)
apply(*unfold C-def*)
apply(*auto*)
done

theorem *pAdj-and-pP-and-pP-and-pDC-imp-pAdj*: [|*pAdj*(*x*,*y*);*pP*(*x*,*xx*);*pP*(*y*,*yy*);*pDC*(*xx*,*yy*)|]
==> *pAdj*(*xx*,*yy*)
apply(*drule-tac x=xx* **and** *y=yy* **in** *pDC-imp-notC*)
apply(*unfold pAdj-def*, *unfold pP-def*)
apply(*clarify*)
apply(*rule Adj-and-P-and-P-and-notC-imp-Adj*)
apply(*auto*)
apply(*erule-tac x=t* **in** *allE*)
apply(*auto*)
apply(*erule-tac x=t* **in** *allE*)
apply(*safe*)
apply(*drule P-exists2-rule*)
apply(*safe*)
apply(*erule-tac x=t* **in** *allE*)
apply(*safe*)
apply(*rotate-tac 1*)
apply(*erule-tac x=t* **in** *allE*)
apply(*safe*)
apply(*drule P-exists2-rule*)
apply(*erule conjE*)
apply(*clarify*)
apply(*rotate-tac 2*)
apply(*erule-tac x=t* **in** *allE*)
apply(*safe*)
apply(*drule P-exists2-rule*)
apply(*clarify*)
apply(*rotate-tac 1*)
apply(*erule-tac x=t* **in** *allE*)
apply(*safe*)
apply(*drule Adj-exists*)
apply(*auto*)
apply(*rotate-tac 1*)
apply(*erule-tac x=t* **in** *allE*)
apply(*safe*)
apply(*drule P-exists2-rule*)
apply(*clarify*)
apply(*rotate-tac 3*)
apply(*erule-tac x=t* **in** *allE*)

```
apply(safe)
apply(drule Adj-exists)
apply(auto)
done


theorem pAdj-irrefl: ALL x. (~pAdj(x,x))
apply(unfold pAdj-def)
apply(insert P-exists1)
apply(insert Adj-irrefl)
apply(auto)
done


theorem pAdj-sym: pAdj(x,y) ==> pAdj(y,x)
apply(unfold pAdj-def)
apply(insert Adj-sym)
apply(auto)
done


theorem Att-imp-pAdj: Att(x,y) ==> pAdj(x,y)
apply(unfold Att-def)
apply(clarify)
apply(drule-tac x=z1 and y=x in pPP-imp-pP-and-notpP)
apply(drule-tac x=z2 and y=y in pPP-imp-pP-and-notpP)
apply(rule pAdj-and-pP-and-pP-and-pDC-imp-pAdj)
apply(auto)
done


theorem Att-sym: Att(x,y) ==> Att(y,x)
apply(unfold Att-def)
apply(safe)
apply(drule pDC-sym)
apply(safe)
apply(drule pAdj-sym)
apply(rule-tac x=z2 in exI)
apply(rule-tac x=z1 in exI)
apply(clarify)
done

theorem Att-irrefl: (ALL x. ~Att(x,x))
apply(unfold Att-def)
apply(insert pDC-irrefl)
apply(auto)
done
```

end
theory *Collections*

imports *TNEMO*

begin

typedecl *Co*

arities *Co* :: *term*

consts

In :: *Ob* => *Co* => *o*
Union :: *Co* => *Co* => *Co* => *o*
Intersect :: *Co* => *Co* => *Co* => *o*
Subseteq :: *Co* => *Co* => *o*
Fp :: *Co* => *Ti* => *o*
Pp :: *Co* => *Ti* => *o*
Np :: *Co* => *Ti* => *o*

DCo :: *Co* => *Ti* => *o*

axioms

Co-members: (*ALL p*. (*EX x y*. (*In*(*x*,*p*) & *In*(*y*,*p*) & *x* ~= *y*)))
Co-ext: (*ALL p q*. (*p*=*q* <-> (*ALL x*. (*In*(*x*,*p*) <-> *In*(*x*,*q*)))))
Co-union: (*ALL p q*. (*EX r*. (*Union*(*p*,*q*,*r*))))
Co-intersect: (*ALL p q*. (*EX x y*. (*x* ~= *y* & *In*(*x*,*p*) & *In*(*x*,*q*) & *In*(*y*,*p*) & *In*(*y*,*q*))) --> (*EX r*. (*Intersect*(*p*,*q*,*r*))))

defs

Subseteq-def: *Subseteq*(*p*,*q*) == (*ALL x*. (*In*(*x*,*p*) --> *In*(*x*,*q*)))
Union-def: *Union*(*p*,*q*,*r*) == (*ALL x*. (*In*(*x*,*r*) <-> (*In*(*x*,*p*) | *In*(*x*,*q*))))
Intersect-def: *Intersect*(*p*,*q*,*r*) == (*ALL x*. (*In*(*x*,*r*) <-> (*In*(*x*,*p*) & *In*(*x*,*q*))))
Fp-def: *Fp*(*p*,*t*) == (*ALL x*. (*In*(*x*,*p*) --> *E*(*x*,*t*)))
Pp-def: *Pp*(*p*,*t*) == (*EX x*. (*In*(*x*,*p*) & *E*(*x*,*t*)))
Np-def: *Np*(*p*,*t*) == (~*Pp*(*p*,*t*))

DCo-def: *DCo*(*p*,*t*) == (*ALL x y*. (*In*(*x*,*p*) & *In*(*y*,*p*) & *O*(*x*,*y*,*t*) --> *x*=*y*))

lemma *Co-ext-rule1*: *p*=*q* ==> (*ALL x*. (*In*(*x*,*p*) <-> *In*(*x*,*q*)))

apply(*insert Co-ext*)
apply(*auto*)
done

lemma *Co-ext-rule2*: (*ALL x*. (*In*(*x*,*p*) <−> *In*(*x*,*q*))) ==> *p*=*q*
apply(*insert Co-ext*)
apply(*auto*)
done

theorem *Union-unique*: [|*Union*(*p*,*q*,*r1*);*Union*(*p*,*q*,*r2*)|] ==> *r1*=*r2*
apply(*unfold Union-def*)
apply(*insert Co-ext*)
apply(*blast*)
done

theorem *Intersect-unique*: [|*Intersect*(*p*,*q*,*r1*);*Intersect*(*p*,*q*,*r2*)|] ==> *r1*=*r2*
apply(*unfold Intersect-def*)
apply(*insert Co-ext*)
apply(*blast*)
done

theorem *Subseteq-refl*: (*ALL p*. *Subseteq*(*p*,*p*))
apply(*unfold Subseteq-def*)
apply(*rule allI*,*rule allI*)
apply(*rule impI*)
apply(*assumption*)
done

theorem *Subseteq-antisym*: (*ALL p q*. (*Subseteq*(*p*,*q*) & *Subseteq*(*q*,*p*) −−> *p*=*q*))
apply(*rule allI*,*rule allI*)
apply(*rule impI*)
apply(*rule Co-ext-rule2*)
apply(*rule allI*)
apply(*rule iffI*)
apply(*erule conjE*)
apply(*unfold Subseteq-def*)
apply(*erule-tac x*=*x* **in** *allE*)
apply(*drule mp*)
apply(*assumption*)
apply(*assumption*)
apply(*erule conjE*)

apply(*rotate-tac 2*)
apply(*erule-tac x=x* **in** *allE*)
apply(*drule mp*)
apply(*assumption*)
apply(*assumption*)
done

lemma *Subseteq-antisym-rule*: [|*Subseteq*(*p*,*q*);*Subseteq*(*q*,*p*)|]==>*p*=*q*
apply(*insert Subseteq-antisym*)
apply(*auto*)
done

theorem *Subseteq-trans*: (*ALL p q r*. (*Subseteq*(*p*,*q*) & *Subseteq*(*q*,*r*) −−> *Subseteq*(*p*,*r*)))
apply(*rule allI*,*rule allI*,*rule allI*)
apply(*rule impI*)
apply(*unfold Subseteq-def*)
apply(*rule allI*)
apply(*erule conjE*)
apply(*erule-tac x=x* **in** *allE*)
apply(*erule-tac x=x* **in** *allE*)
apply(*rule impI*)
apply(*drule mp*)
apply(*assumption*)
apply(*drule mp*)
apply(*assumption*)
apply(*assumption*)
done

lemma *Subseteq-trans-rule*: [|*Subseteq*(*p*,*q*); *Subseteq*(*q*,*r*)|] ==> *Subseteq*(*p*,*r*)
apply(*insert Subseteq-trans*)
apply(*erule-tac x=p* **in** *allE*)
apply(*erule-tac x=q* **in** *allE*)
apply(*erule-tac x=r* **in** *allE*)
apply(*drule-tac P= Subseteq*(*p*, *q*) **and** *Q=Subseteq*(*q*, *r*) **in** *conjI*)
apply(*assumption*)
apply(*drule mp*)
apply(*auto*)
done

theorem *Fp-imp-Pp*: *Fp*(*p*,*t*) ==> *Pp*(*p*,*t*)
apply(*unfold Pp-def*)
apply(*unfold Fp-def*)
apply(*insert Co-members*)
apply(*rotate-tac 1*)
apply(*erule-tac x=p* **in** *allE*)

apply(*erule exE*,*erule exE*)
apply(*erule conjE*,*erule conjE*)
apply(*rule-tac x=x* **in** *exI*)
apply(*erule-tac x=x* **in** *allE*)
apply(*auto*)
done

theorem *Subseteq-and-Fp-imp-Fp*: [|*Subseteq*(*p*,*q*);*Fp*(*q*,*t*)|] ==> *Fp*(*p*,*t*)
apply(*unfold Subseteq-def*)
apply(*unfold Fp-def*)
apply(*auto*)
done

theorem *Subseteq-and-Np-imp-Np*: [|*Subseteq*(*p*,*q*);*Np*(*q*,*t*)|] ==> *Np*(*p*,*t*)
apply(*unfold Subseteq-def*)
apply(*unfold Np-def*)
apply(*unfold Pp-def*)
apply(*auto*)
done

theorem *Subseteq-and-Pp-imp-Pp*: [|*Subseteq*(*p*,*q*);*Pp*(*p*,*t*)|] ==> *Pp*(*q*,*t*)
apply(*unfold Subseteq-def*)
apply(*unfold Pp-def*)
apply(*auto*)
done

theorem *Np-imp-Do*: *Np*(*p*,*t*) ==> *DCo*(*p*,*t*)
apply(*unfold Np-def*)
apply(*unfold Pp-def*)
apply(*unfold DCo-def*)
apply(*unfold O-def*)
apply(*auto*)
apply(*erule-tac x=x* **in** *allE*)
apply(*auto*)
apply(*drule P-exists2-rule*)
apply(*auto*)
done

theorem *DCo-subseteq*: (*ALL p q t*. (*DCo*(*p*,*t*) & *Subseteq*(*q*,*p*) --> *DCo*(*q*,*t*)))
apply(*rule allI*,*rule allI*,*rule allI*)
apply(*rule impI*)
apply(*unfold DCo-def*)
apply(*rule allI*,*rule allI*)
apply(*erule conjE*)
apply(*erule-tac x=x* **in** *allE*)
apply(*erule-tac x=y* **in** *allE*)
apply(*unfold Subseteq-def*)
apply(*auto*)

done

lemma *DCo-subseteq-rule*: [|*DCo*(*p*,*t*); *Subseteq*(*q*,*p*)|] ==> *DCo*(*q*,*t*)
apply(*insert DCo-subseteq*)
apply(*auto*)
done

end
theory *SumsPartitions*

imports *TNEMO Collections*

begin

consts

SumPp :: *Co* => *Ob* => *Ti* => *o*
PtPp :: *Co* => *Ob* => *Ti* => *o*
SumFp :: *Co* => *Ob* => *Ti* => *o*
PtFp :: *Co* => *Ob* => *Ti* => *o*

cSumFp :: *Co* => *Ob* => *o*
bSumFp :: *Co* => *Ob* => *o*
pSumFp :: *Co* => *Ob* => *o*

defs

SumPp-def: *SumPp*(*p*,*x*,*t*) == (*Pp*(*p*,*t*) & (*ALL z*. (*O*(*z*,*x*,*t*) <-> (*EX y*. (*In*(*y*,*p*) & *O*(*z*,*y*,*t*))))))
SumFp-def: *SumFp*(*p*,*x*,*t*) == (*Fp*(*p*,*t*) & (*ALL z*. (*O*(*z*,*x*,*t*) <-> (*EX y*. (*In*(*y*,*p*) & *O*(*z*,*y*,*t*))))))
PtPp-def: *PtPp*(*p*,*x*,*t*) == (*SumPp*(*p*,*x*,*t*) & *DCo*(*p*,*t*))
PtFp-def: *PtFp*(*p*,*x*,*t*) == (*SumFp*(*p*,*x*,*t*) & *DCo*(*p*,*t*))

cSumFp-def: *cSumFp*(*p*,*x*) == (*ALL t*. (*E*(*x*,*t*) --> *SumFp*(*p*,*x*,*t*)))
bSumFp-def: *bSumFp*(*p*,*x*) == (*ALL t*. (*Fp*(*p*,*t*) --> *SumFp*(*p*,*x*,*t*)))
pSumFp-def: *pSumFp*(*p*,*x*) == *cSumFp*(*p*,*x*) & *bSumFp*(*p*,*x*)

theorem *SumPp-and-In-and-E-imp-P*: (*ALL p x y t*. (*SumPp*(*p*,*x*,*t*) & *In*(*y*,*p*) & *E*(*y*,*t*) --> *P*(*y*,*x*,*t*)))
apply(*rule allI*,*rule allI*,*rule allI*,*rule allI*,*rule impI*,*erule conjE*,*erule conjE*)
apply(*rule-tac x*=*y* **and** *y*=*x* **and** *t*=*t* **in** *O-imp-O-imp-P-rule*)
apply(*assumption*)
apply(*unfold SumPp-def*)

apply(*auto*)
done

lemma *SumPp-and-In-and-E-imp-P-rule*: [|*SumPp*(*p*,*x*,*t*); *In*(*y*,*p*);*E*(*y*,*t*)|] ==> *P*(*y*,*x*,*t*)
apply(*insert SumPp-and-In-and-E-imp-P*)
apply(*auto*)
done

theorem *SumFp-and-In-imp-P*: [|*SumFp*(*p*,*x*,*t*); *In*(*y*,*p*)|] ==> *P*(*y*,*x*,*t*)
apply(*unfold SumFp-def*)
apply(*rule-tac x*=*y* **and** *y*=*x* **and** *t*=*t* **in** *O-imp-O-imp-P-rule*)
apply(*unfold Fp-def*)
apply(*auto*)
done

theorem *SumPp-imp-E*: (*ALL p x t*. (*SumPp*(*p*,*x*,*t*) --> *E*(*x*,*t*)))
apply(*rule allI*,*rule allI*,*rule allI*,*rule impI*)
apply(*unfold SumPp-def*)
apply(*erule conjE*)
apply(*unfold Pp-def*)
apply(*erule exE*)
apply(*erule conjE*)
apply(*drule O-refl-rule*)
apply(*erule-tac x*=*xa* **in** *allE*)
apply(*drule conjI*)
apply(*assumption*)
apply(*auto*)
apply(*rotate-tac 2*)
apply(*drule O-imp-E-and-E*)
apply(*auto*)
done

lemma *SumPp-imp-E-rule*: *SumPp*(*p*,*x*,*t*) ==> *E*(*x*,*t*)
apply(*insert SumPp-imp-E*)
apply(*auto*)
done

theorem *SumPp-and-SumPp-imp-Me*: (*ALL p x y t*. (*SumPp*(*p*,*x*,*t*) & *SumPp*(*p*,*y*,*t*) --> *Me*(*x*,*y*,*t*)))
apply(*rule allI*,*rule allI*,*rule allI*,*rule allI*)
apply(*rule impI*)
apply(*erule conjE*)
apply(*unfold Me-def*)
apply(*frule SumPp-imp-E-rule*)

apply(*rotate-tac 1*)
apply(*frule SumPp-imp-E-rule*)
apply(*rule conjI*)
apply(*rule O-imp-O-imp-P-rule*)
apply(*unfold SumPp-def*)
apply(*auto*)
apply(*rule O-imp-O-imp-P-rule*)
apply(*auto*)
done

lemma *SumPp-and-SumPp-imp-Me-rule*: $[|SumPp(p,x,t);SumPp(p,y,t)|] ==> Me(x,y,t)$
apply(*insert SumPp-and-SumPp-imp-Me*)
apply(*auto*)
done

theorem *SumPp-and-P-impl-Overlap*: $[|SumPp(p,y,t);P(x,y,t)|] ==> (EX\ z.\ (In(z,p)$
$\&\ O(x,z,t)))$
apply(*unfold SumPp-def*)
apply(*erule conjE*)
apply(*erule-tac x=x* **in** *allE*)
apply(*drule P-imp-O*)
apply(*auto*)
done

theorem *SumPp-and-SumPp-and-Subseteq-impl-P*: $[|SumPp(p,x,t);SumPp(q,y,t);Subseteq(p,q)|]$
$==> P(x,y,t)$
apply(*rule O-imp-O-imp-P-rule*)
apply(*drule SumPp-imp-E-rule*)
apply(*assumption*)
apply(*unfold SumPp-def*)
apply(*erule conjE,erule conjE*)
apply(*rule allI*)
apply(*rule impI*)
apply(*erule-tac x=z* **in** *allE*)
apply(*erule iffE*)
apply(*drule mp*)
apply(*assumption*)
apply(*erule exE*)
apply(*unfold Subseteq-def*)
apply(*erule-tac x=ya* **in** *allE*)
apply(*erule conjE*)
apply(*drule-tac P=In(ya,p)* **and** *Q=In(ya,q)* **in** *mp*)
apply(*assumption*)
apply(*drule-tac P=In(ya, p)* **and** *Q=O(z, ya, t)* **in** *conjI*)
apply(*assumption*)
apply(*rotate-tac 7*)
apply(*drule-tac x=ya* **in** *exI*)
apply(*auto*)

done

theorem *SumFp-imp-SumPp*: *SumFp*(p,x,t) ==> *SumPp*(p,x,t)
apply(*unfold SumFp-def*,*unfold SumPp-def*)
apply(*insert Fp-imp-Pp*)
apply(*auto*)
done

theorem *PtFp-imp-PtPp*: *PtFp*(p,x,t) ==> *PtPp*(p,x,t)
apply(*unfold PtFp-def*,*unfold PtPp-def*)
apply(*insert SumFp-imp-SumPp*)
apply(*auto*)
done

theorem *cSumFp-imp-E-imp-Fp*: *cSumFp*(p,x) ==> (*ALL* t. (*E*(x,t) −−> *Fp*(p,t)))
apply(*unfold cSumFp-def*,*unfold SumFp-def*)
apply(*auto*)
done

theorem *cSumFp-and-In-imp-cP*: [|*cSumFp*(p,x);*In*(y,p)|] ==> *cP*(y,x)
apply(*unfold cSumFp-def*,*unfold cP-def*)
apply(*safe*)
apply(*erule-tac* x=t **in** *allE*)
apply(*clarify*)
apply(*rule SumFp-and-In-imp-P*)
apply(*auto*)
done

theorem *bSumFp-and-Fp-imp-E*: *bSumFp*(p,x) −−> (*ALL* t. (*Fp*(p,t) −−> *E*(x,t)))
apply(*unfold bSumFp-def*)
apply(*safe*)
apply(*erule-tac* x=t **in** *allE*)
apply(*safe*)
apply(*drule SumFp-imp-SumPp*)
apply(*drule SumPp-imp-E-rule*)
apply(*assumption*)
done

theorem *bSumFp-and-bSumFp-and-Fp-imp-Me*: [|*bSumFp*(p,x);*bSumFp*(p,y);*Fp*(p,t)|]
==> *Me*(x,y,t)
apply(*unfold bSumFp-def*)
apply(*erule-tac* x=t **in** *allE*)
apply(*erule-tac* x=t **in** *allE*)
apply(*safe*)
apply(*drule SumFp-imp-SumPp*)
apply(*drule SumFp-imp-SumPp*)
apply(*insert SumPp-and-SumPp-imp-Me*)

apply(*auto*)
done

theorem *bSumFp-and-cSumFp-and-Subseteq-imp-cP*: [|*bSumFp*(*p*,*x*);*cSumFp*(*q*,*y*);*Subseteq*(*p*,*q*)|] ==> *cP*(*x*,*y*)
apply(*unfold bSumFp-def*,*unfold cSumFp-def*,*unfold cP-def*)
apply(*safe*)
apply(*erule-tac x*=*t* **in** *allE*)
apply(*erule-tac x*=*t* **in** *allE*)
apply(*safe*)
apply(*unfold SumFp-def*)
apply(*drule Subseteq-and-Fp-imp-Fp*)
apply(*force*)
apply(*force*)
apply(*fold SumFp-def*)
apply(*drule SumFp-imp-SumPp*)
apply(*drule SumFp-imp-SumPp*)
apply(*drule-tac p*=*p* **and** *x*=*x* **and** *q*=*q* **and** *y*=*y* **and** *t*=*t* **in** *SumPp-and-SumPp-and-Subseteq-impl-P*)
apply(*auto*)
done

theorem *cSumFp-and-In-imp-pSumFp*: [|*cSumFp*(*p*,*x*);*In*(*x*,*p*)|] ==> *pSumFp*(*p*,*x*)
apply(*unfold pSumFp-def*)
apply(*safe*)
apply(*unfold bSumFp-def*)
apply(*unfold cSumFp-def*)
apply(*safe*)
apply(*erule-tac x*=*t* **in** *allE*)
apply(*auto*)
apply(*unfold Fp-def*)
apply(*auto*)
done

theorem *pSumFp-imp-Fp-iff-SumFp*: *pSumFp*(*p*,*x*) ==> (*ALL t*. (*Fp*(*p*,*t*) <-> *SumFp*(*p*,*x*,*t*)))
apply(*unfold pSumFp-def*,*unfold cSumFp-def*,*unfold bSumFp-def*)
apply(*auto*)
apply(*erule-tac x*=*t* **in** *allE*)
apply(*erule-tac x*=*t* **in** *allE*)
apply(*drule SumFp-imp-SumPp*)
apply(*drule SumPp-imp-E-rule*)
apply(*auto*)
apply(*unfold SumFp-def*)
apply(*auto*)
done

theorem *pSumFp-imp-E-iff-SumFp*: *pSumFp*(*p*,*x*) ==> (*ALL t*. (*E*(*x*,*t*) <-> *SumFp*(*p*,*x*,*t*)))

apply(*unfold pSumFp-def*,*unfold cSumFp-def*,*unfold bSumFp-def*)
apply(*auto*)
apply(*drule SumFp-imp-SumPp*)
apply(*drule SumPp-imp-E-rule*)
apply(*auto*)
done

theorem *pSumFp-and-pSumFp-imp-E-iff-E*: [|*pSumFp*(*p*,*x*);*pSumFp*(*p*,*y*)|] ==>
(*ALL t*. (*E*(*x*,*t*) <−> *E*(*y*,*t*)))
apply(*unfold pSumFp-def*,*unfold cSumFp-def*,*unfold bSumFp-def*)
apply(*auto*)
apply(*erule-tac x*=*t* **in** *allE*)
apply(*erule-tac x*=*t* **in** *allE*)
apply(*auto*)
apply(*erule-tac x*=*t* **in** *allE*)
apply(*erule-tac x*=*t* **in** *allE*)
apply(*unfold SumFp-def*)
apply(*erule conjE*)
apply(*fold SumFp-def*)
apply(*clarify*)
apply(*drule SumFp-imp-SumPp*)
apply(*drule SumPp-imp-E-rule*)
apply(*auto*)
apply(*erule-tac x*=*t* **in** *allE*)
apply(*erule-tac x*=*t* **in** *allE*)
apply(*erule-tac x*=*t* **in** *allE*)
apply(*erule-tac x*=*t* **in** *allE*)
apply(*clarify*)
apply(*unfold SumFp-def*)
apply(*erule conjE*)
apply(*fold SumFp-def*)
apply(*clarify*)
apply(*drule-tac x*=*x* **in** *SumFp-imp-SumPp*)
apply(*drule SumPp-imp-E-rule*)
apply(*assumption*)
done

theorem *pSumFp-and-pSumFp-imp-E-iff-ME*: [|*pSumFp*(*p*,*x*);*pSumFp*(*p*,*y*)|] ==>
(*ALL t*. (*E*(*x*,*t*) <−> *Me*(*x*,*y*,*t*)))
apply(*unfold pSumFp-def*,*unfold cSumFp-def*,*unfold bSumFp-def*)
apply(*auto*)
apply(*erule-tac x*=*t* **in** *allE*)
apply(*erule-tac x*=*t* **in** *allE*)
apply(*erule-tac x*=*t* **in** *allE*)
apply(*erule-tac x*=*t* **in** *allE*)
apply(*clarify*)
apply(*unfold SumFp-def*)
apply(*erule conjE*)
apply(*fold SumFp-def*)

apply(*clarify*)
apply(*drule SumFp-imp-SumPp*)
apply(*drule SumFp-imp-SumPp*)
apply(*drule SumPp-and-SumPp-imp-Me-rule*)
apply(*auto*)
apply(*drule Me-exists2*)
apply(*auto*)
done

end
theory *Universals* **imports** *FOL*

begin

typedecl *Un*

arities *Un* :: *term*

consts

IsA :: *Un* => *Un* => *o*
IsAPr :: *Un* => *Un* => *o*
IsARoot :: *Un* => *o*
IsAI :: *Un* => *Un* => *o*
IsAO :: *Un* => *Un* => *o*

axioms

IsA-refl: (*ALL c*. (*IsA*(*c*,*c*)))
IsA-antisym: (*ALL c d*. (*IsA*(*c*,*d*) & *IsA*(*d*,*c*) --> *c* = *d*))
IsA-trans: (*ALL c d e*. (*IsA*(*c*,*d*) & *IsA*(*d*,*e*) --> *IsA*(*c*,*e*)))
IsA-wsp-IsAI: (*ALL c d*. (*IsAPr*(*c*,*d*) --> (*EX e*. (*IsAPr*(*e*,*d*) & ~*IsAI*(*e*,*c*)))))

IsA-npo: *ALL c d*. (*IsAO*(*c*,*d*) --> *IsAI*(*c*,*d*))
IsA-root: (*EX c*. *IsARoot*(*c*))

defs

IsAPr-def: *IsAPr*(*c*,*d*) == *IsA*(*c*,*d*) & ~*IsA*(*d*,*c*)
IsARoot-def: *IsARoot*(*d*) == (*ALL c*. *IsA*(*c*,*d*))
IsAI-def: *IsAI*(*c*,*d*) == *IsA*(*c*,*d*) | *IsA*(*d*,*c*)
IsAO-def: *IsAO*(*c*,*d*) == (*EX e*. (*IsA*(*e*,*c*) & *IsA*(*e*,*d*)))

lemma *IsA-antisym-rule* : [|*IsA*(*c*,*d*);*IsA*(*d*,*c*)|] ==> *c* = *d*
apply(*insert IsA-antisym*)
apply(*erule-tac x*=*c* **in** *allE*,*erule-tac x*=*d* **in** *allE*)

apply(*drule mp*)
apply(*auto*)
done

lemma *IsA-trans-rule*: [|*IsA*(*c*,*d*);*IsA*(*d*,*e*)|] ==> *IsA*(*c*,*e*)
apply(*insert IsA-trans*)
apply(*erule-tac x*=*c* **in** *allE*,*erule-tac x*=*d* **in** *allE*, *erule-tac x*=*e* **in** *allE*)
apply(*drule mp*)
apply(*auto*)
done

lemma *IsA-wsp-IsAI-rule*: *IsAPr*(*c*,*d*) ==> (*EX e*. (*IsAPr*(*e*,*d*) & ~*IsAI*(*e*,*c*)))

apply(*insert IsA-wsp-IsAI*)
apply(*erule-tac x*=*c* **in** *allE*,*erule-tac x*=*d* **in** *allE*)
apply(*drule mp*)
apply(*auto*)
done

lemma *IsA-npo-rule*: [|*IsA*(*e*,*c*);*IsA*(*e*,*d*)|] ==> *IsAI*(*c*,*d*)
apply(*insert IsA-npo*)
apply(*unfold IsAO-def*)
apply(*auto*)
done

lemma *IsA-npo-rule1*: *IsAO*(*c*,*d*) ==> *IsAI*(*c*,*d*)
apply(*insert IsA-npo*)
apply(*auto*)
done

theorem *IsA-impl-Id-or-IsAPr*: *IsA*(*c*,*d*) ==> (*c*=*d* | *IsAPr*(*c*,*d*))
apply(*safe*)
apply(*unfold IsAPr-def*)
apply(*safe*)
apply(*drule IsA-antisym-rule*)
apply(*auto*)
done

theorem *IsARoot-unique*: [|*IsARoot*(*c*);*IsARoot*(*d*)|] ==> *c*=*d*
apply(*unfold IsARoot-def*)
apply(*insert IsA-antisym-rule*)
apply(*auto*)
done

theorem *IsAI-refl*: (*ALL c. IsAI*(*c*,*c*))
apply(*unfold IsAI-def*)
apply(*rule allI*)
apply(*insert IsA-refl*)
apply(*erule-tac x*=*c* **in** *allE*)
apply(*auto*)
done

theorem *IsA-imp-IsAI*: *IsA*(*c*,*d*) ==> *IsAI*(*c*,*d*)
apply(*unfold IsAI-def*)
apply(*auto*)
done

theorem *IsAPr-imp-IsA*: *IsAPr*(*c*,*d*) ==> *IsA*(*c*,*d*)
apply(*unfold IsAPr-def*)
apply(*erule conjE*)
apply(*assumption*)
done

theorem *IsAI-imp-IsAI-imp-IsA-rule*: (*ALL e.* (*IsAI*(*e*,*c*)−−>*IsAI*(*e*,*d*))) ==> *IsA*(*c*,*d*)
apply(*frule-tac x*=*c* **in** *spec*)
apply(*insert IsAI-refl*)
apply(*rotate-tac 2*)
apply(*erule-tac x*=*c* **in** *allE*)
apply(*drule mp*)
apply(*assumption*)
apply(*unfold IsAI-def*)
apply(*rotate-tac 2*)
apply(*erule disjE*)
apply(*assumption*)
apply(*drule-tac c*=*d* **and** *d*=*c* **in** *IsA-impl-Id-or-IsAPr*)
apply(*fold IsAI-def*)
apply(*safe*)
apply(*insert IsA-refl*)
apply(*auto*)
apply(*drule-tac c*=*d* **and** *d*=*c* **in** *IsA-wsp-IsAI-rule*)
apply(*erule exE*)
apply(*erule conjE*)
apply(*erule-tac x*=*e* **in** *allE*)
apply(*auto*)
apply(*drule-tac c*=*e* **and** *d*=*c* **in** *IsAPr-imp-IsA*)
apply(*drule-tac c*=*e* **and** *d*=*c* **in** *IsA-imp-IsAI*)
apply(*auto*)
done

lemma *IsAI-imp-IsAI-imp-IsA*: (*ALL c d.* (*ALL e.* (*IsAI*(*e*,*c*)−−>*IsAI*(*e*,*d*))) −−> *IsA*(*c*,*d*))
apply(*insert IsAI-imp-IsAI-imp-IsA-rule*)
apply(*auto*)
done

lemma *ltb3*: (*ALL e.* (*A*(*e*,*c*) & *B*(*e*,*d*))) ==> (*ALL e.* *A*(*e*,*c*)) & (*ALL e.* *B*(*e*,*d*))
apply(*auto*)
done

theorem *IsAI-iff-IsAI-iff-eq*: (*ALL c d.* (*ALL e.* (*IsAI*(*e*,*c*) <−> *IsAI*(*e*,*d*))) <−> *c*=*d*)
apply(*clarify*)
apply(*rule iffI*)
prefer *2*
apply(*auto*)
apply(*unfold iff-def*)
apply(*drule ltb3*)
apply(*rule IsA-antisym-rule*)
apply(*insert IsAI-imp-IsAI-imp-IsA*)
apply(*auto*)
done

theorem *IsA-wsp*: *IsAPr*(*c*,*d*) ==> (*EX e.* (*IsAPr*(*e*,*d*) & ~(*EX f.* *IsA*(*f*,*e*) & *IsA*(*f*,*c*))))
apply(*drule IsA-wsp-IsAI-rule*)
apply(*clarify*)
apply(*rule-tac x*=*e* **in** *exI*)
apply(*safe*)
apply(*drule-tac e*=*f* **and** *c*=*e* **and** *d*=*c* **in** *IsA-npo-rule*)
apply(*auto*)
done

theorem *IsA-and-IsAI-impl-IsAI*: [|*IsA*(*c*,*d*);*IsAI*(*c*,*e*)|] ==> *IsAI*(*d*,*e*)
apply(*unfold IsAI-def*)
apply(*safe*)
apply(*drule-tac e*=*c* **and** *c*=*d* **and** *d*=*e* **in** *IsA-npo-rule*)
apply(*assumption*)
apply(*unfold IsAI-def*)
apply(*auto*)
apply(*drule-tac c*=*e* **and** *d*=*c* **and** *e*=*d* **in** *IsA-trans-rule*)
apply(*auto*)
done

theorem *IsA-and-not-IsAI-impl-not-IsAI*: [|*IsA*(*c*,*d*);~*IsAI*(*d*,*e*)|] ==> ~*IsAI*(*c*,*e*)
apply(*unfold IsAI-def*)

apply(*safe*)
apply(*drule-tac e=c* **and** *c=d* **and** *d=e* **in** *IsA-npo-rule*)
apply(*assumption*)
apply(*unfold IsAI-def*)
apply(*auto*)
apply(*drule-tac c=e* **and** *d=c* **and** *e=d* **in** *IsA-trans-rule*)
apply(*auto*)
done

theorem *IsAI-impl-IsA-and-IsA*: *IsAI(c,d) ==> IsAO(c,d)*
apply(*unfold IsAI-def,unfold IsAO-def*)
apply(*safe*)
apply(*insert IsA-refl*)
apply(*rule-tac x=c* **in** *exI*)
apply(*force*)
apply(*rule-tac x=d* **in** *exI*)
apply(*auto*)
done

theorem *IsAI-iff-IsAO*: *ALL c d. (IsAI(c,d) <−> IsAO(c,d))*
apply(*auto*)
apply(*rule IsAI-impl-IsA-and-IsA*)
apply(*assumption*)
apply(*drule IsA-npo-rule1*)
apply(*assumption*)
done

end
theory *Instantiation*

imports *TNEMO Universals*

begin

consts

Inst :: Ob => Un => Ti => o

axioms

Inst-IsA: *(ALL c d t x. (IsA(c,d) −−> (Inst(x,c,t) −−> Inst(x,d,t))))*
Inst-IsAI: *(ALL x c d t. ((Inst(x,c,t) & Inst(x,d,t) −−> IsAI(c,d))))*
Inst-E: *(ALL x c t. (Inst(x,c,t) −−> E(x,t)))*
Inst-Un: *(ALL c. (EX x t. (Inst(x,c,t))))*
Inst-Ob: *(ALL x t. (E(x,t) −−> (EX c. (Inst(x,c,t)))))*

lemma *Inst-IsA-rule*: *IsA(c,d) ==> (ALL x t. (Inst(x,c,t) −−> Inst(x,d,t)))*

apply(*insert Inst-IsA*)
apply(*auto*)
done

lemma *Inst-IsAI-rule*: [|*Inst*(*x*,*c*,*t*);*Inst*(*x*,*d*,*t*)|] ==> *IsAI*(*c*,*d*)
apply(*insert Inst-IsAI*)
apply(*auto*)
done

lemma *Inst-Ob-rule*: (*E*(*x*,*t*) ==> (*EX c*. (*Inst*(*x*,*c*,*t*))))
apply(*insert Inst-Ob*)
apply(*auto*)
done

end
theory *ExtensionsOfUniversals*

imports *Instantiation Collections*

begin

consts

ExtCo :: *Co* => *Un* => *Ti* => *o*
ExtOb :: *Ob* => *Un* => *Ti* => *o*
DUn :: *Un* => *o*

axioms

Inst-impl-ExtOb-or-ExtCo: *Inst*(*x*,*c*,*t*) ==> (*ExtOb*(*x*,*c*,*t*) | (*EX p*. *ExtCo*(*p*,*c*,*t*)))

defs

ExtCo-def: *ExtCo*(*p*,*c*,*t*) == (*ALL x*. (*In*(*x*,*p*) <−> *Inst*(*x*,*c*,*t*)))
ExtOb-def: *ExtOb*(*x*,*c*,*t*) == (*Inst*(*x*,*c*,*t*) & (*ALL y*. (*Inst*(*y*,*c*,*t*) −−> *x*=*y*)))
DUn-def: *DUn*(*c*) == (*ALL t*. (*ALL p*. *ExtCo*(*p*,*c*,*t*) −−> *DCo*(*p*,*t*)))

theorem *DistinctInsts-impl-ExtCo*: [|*Inst*(*x*,*c*,*t*);*Inst*(*y*,*c*,*t*); (*x*~=*y*)|] ==> (*EX p*. *ExtCo*(*p*,*c*,*t*))
apply(*drule Inst-impl-ExtOb-or-ExtCo* [*of x c t*])
apply(*erule disjE*)
apply(*rule-tac x*=*p* **in** *exI*)
apply(*unfold ExtOb-def*)
apply(*erule conjE*)
apply(*erule-tac x*=*y* **in** *allE*)

```
apply(auto)
done

theorem ExtOb-unique: [|ExtOb(x,c,t);ExtOb(y,c,t)|] ==> x=y
apply(unfold ExtOb-def)
apply(auto)
done

theorem ExtCo-unique: [|ExtCo(p,c,t);ExtCo(q,c,t)|] ==> p = q
apply(rule Co-ext-rule2)
apply(auto)
apply(unfold ExtCo-def)
apply(auto)
done

theorem ExtCo-Fp: ExtCo(p,c,t) ==> Fp(p,t)
apply(unfold Fp-def,unfold ExtCo-def)
apply(auto)
apply(insert Inst-E)
apply(auto)
done

theorem ExtCo-impl-2members: ExtCo(p,c,t) ==> (EX x y. (In(x,p) & In(y,p)
& x ~= y))
apply(insert Co-members)
apply(auto)
done

theorem IsA-impl-Subseteq: [|IsA(c,d);ExtCo(p,c,t);ExtCo(q,d,t)|] ==> Subseteq(p,q)
apply(unfold ExtCo-def)
apply(unfold Subseteq-def)
apply(auto)
apply(drule Inst-IsA-rule)
apply(auto)
done


theorem ExtOb-impl-notExtCo: ExtOb(x,c,t) ==> (~(EX p. ExtCo(p,c,t)))
apply(clarify)
apply(insert Co-members)
apply(erule-tac x=p in allE)
apply(unfold ExtOb-def)
apply(unfold ExtCo-def)
apply(clarify)
apply(frule-tac x=xa in spec)
apply(erule-tac x=y in allE)
apply(frule-tac x=xa in spec)
apply(erule-tac x=y in allE)
apply(auto)
```

done

theorem ExtCo-impl-notExtOb: ExtCo(p,c,t) ==> (~(EX x. ExtOb(x,c,t)))
apply(clarify)
apply(insert Co-members)
apply(erule-tac x=p **in** allE)
apply(unfold ExtOb-def)
apply(unfold ExtCo-def)
apply(clarify)
apply(frule-tac x=xa **in** spec)
apply(erule-tac x=y **in** allE)
apply(frule-tac x=xa **in** spec)
apply(erule-tac x=y **in** allE)
apply(auto)
done

theorem NoExt-or-ExtOb-or-ExtDCo-impl-DUn: (ALL t. ((~ (EX x. Inst(x,c,t))) | (EX x. ExtOb(x,c,t)) | (EX p. ExtCo(p,c,t) & DCo(p,t)))) ==> DUn(c)
apply(unfold DUn-def)
apply(clarify)
apply(erule-tac x=t **in** allE)
apply(safe)
apply(unfold ExtCo-def)
apply(insert Co-members)
apply(force)
apply(fold ExtCo-def)
apply(insert ExtCo-impl-notExtOb)
apply(force)
apply(insert ExtCo-unique)
apply(force)
done

theorem DUn-impl-NoExt-or-ExtOb-or-ExtDCo: DUn(c) ==> (ALL t. ((~ (EX x. Inst(x,c,t))) | (EX x. ExtOb(x,c,t)) | (EX p. ExtCo(p,c,t) & DCo(p,t))))
apply(clarify)
apply(drule-tac x=x **and** c=c **and** t=t **in** Inst-impl-ExtOb-or-ExtCo)
apply(auto)
apply(unfold DUn-def)
apply(erule-tac x=t **in** allE)
apply(erule-tac x=p **in** allE)
apply(erule-tac x=x **in** allE)
apply(erule-tac x=p **in** allE)
apply(auto)
done

theorem DUn-iff-NoExt-or-ExtOb-or-ExtDCo: DUn(c) <-> (ALL t. ((~ (EX x. Inst(x,c,t))) | (EX x. ExtOb(x,c,t)) | (EX p. ExtCo(p,c,t) & DCo(p,t))))

apply(*rule iffI*)
apply(*rule DUn-impl-NoExt-or-ExtOb-or-ExtDCo*)
apply(*assumption*)
apply(*rule NoExt-or-ExtOb-or-ExtDCo-impl-DUn*)
apply(*assumption*)
done

theorem *DUn-and-O-impl-Id*: [|*DUn*(*c*);*Inst*(*x*,*c*,*t*);*Inst*(*y*,*c*,*t*);*O*(*x*,*y*,*t*)|] ==> *x* = *y*
apply(*frule-tac x*=*x* **and** *c*=*c* **and** *t*=*t* **in** *Inst-impl-ExtOb-or-ExtCo*)
apply(*frule-tac x*=*y* **and** *c*=*c* **and** *t*=*t* **in** *Inst-impl-ExtOb-or-ExtCo*)
apply(*safe*)
apply(*unfold ExtOb-def*)
apply(*clarify*)
apply(*erule-tac x*=*y* **in** *allE*)
apply(*force*)
apply(*fold ExtOb-def*)
apply(*drule ExtCo-impl-notExtOb*)
apply(*force*)
apply(*drule ExtCo-impl-notExtOb*)
apply(*force*)
apply(*drule-tac p*=*p* **and** *c*=*c* **and** *t*=*t* **and** *q*=*pa* **in** *ExtCo-unique*)
apply(*assumption*)
apply(*unfold DUn-def*)
apply(*erule-tac x*=*t* **in** *allE*)
apply(*erule-tac x*=*pa* **in** *allE*)
apply(*auto*)
apply(*unfold ExtCo-def*,*unfold DCo-def*)
apply(*frule-tac x*=*x* **in** *spec*)
apply(*erule-tac x*=*y* **in** *allE*)
apply(*erule-tac x*=*x* **in** *allE*)
apply(*erule-tac x*=*y* **in** *allE*)
apply(*auto*)
done

end

theory *PartonomicInclusion*

imports *TNEMO Collections*

begin

consts

P1 :: *Co* => *Co* => *Ti* => *o*
P2 :: *Co* => *Co* => *Ti* => *o*
P12 :: *Co* => *Co* => *Ti* => *o*
DP1 :: *Co* => *Co* => *Ti* => *o*
DP2 :: *Co* => *Co* => *Ti* => *o*
DP12 :: *Co* => *Co* => *Ti* => *o*

defs

P1-def: *P1*(*p*,*q*,*t*) == (*ALL x*. (*In*(*x*,*p*) --> (*EX y*. (*In*(*y*,*q*) & *P*(*x*,*y*,*t*)))))
P2-def: *P2*(*p*,*q*,*t*) == (*ALL y*. (*In*(*y*,*q*) --> (*EX x*. (*In*(*x*,*p*) & *P*(*x*,*y*,*t*)))))
P12-def: *P12*(*p*,*q*,*t*) == *P1*(*p*,*q*,*t*) & *P2*(*p*,*q*,*t*)

DP1-def: *DP1*(*p*,*q*,*t*) == *P1*(*p*,*q*,*t*) & *DCo*(*p*,*t*) & *DCo*(*q*,*t*)
DP2-def: *DP2*(*p*,*q*,*t*) == *P2*(*p*,*q*,*t*) & *DCo*(*p*,*t*) & *DCo*(*q*,*t*)
DP12-def: *DP12*(*p*,*q*,*t*) == *DP1*(*p*,*q*,*t*) & *DP2*(*p*,*q*,*t*)

theorem *P1-imp-Fp-Pp*: *P1*(*p*,*q*,*t*) ==> (*Fp*(*p*,*t*) & *Pp*(*q*,*t*))
apply(*unfold P1-def*,*unfold Fp-def*,*unfold Pp-def*)
apply(*rule conjI*)
apply(*rule allI*)
apply(*rule impI*)
apply(*erule-tac x*=*x* **in** *allE*)
apply(*auto*)
apply(*drule P-exists2-rule*)
apply(*auto*)
apply(*insert Co-members*)
apply(*rotate-tac 1*)
apply(*erule-tac x*=*p* **in** *allE*)
apply(*erule exE*)
apply(*erule exE*)
apply(*erule-tac x*=*x* **in** *allE*)
apply(*auto*)
apply(*rule exI*)
apply(*rule conjI*)
apply(*assumption*)
apply(*drule P-exists2-rule*)
apply(*auto*)
done

theorem *P2-imp-Pp-Fp*: *P2*(*p*,*q*,*t*) ==>(*Pp*(*p*,*t*) & *Fp*(*q*,*t*))
apply(*unfold P2-def*,*unfold Fp-def*, *unfold Pp-def*)
apply(*rule conjI*)
apply(*insert Co-members*)
apply(*rotate-tac 1*)

apply(*erule-tac x=q* **in** *allE*)
apply(*erule exE*,*erule exE*)
apply(*erule-tac x=y* **in** *allE*)
apply(*erule conjE*,*erule conjE*)
apply(*drule mp*)
apply(*assumption*)
apply(*erule exE*)
apply(*erule conjE*)
apply(*drule P-exists2-rule*)
apply(*erule conjE*)
apply(*rule-tac x=xa* **in** *exI*)
apply(*rule conjI*)
apply(*assumption*,*assumption*)
apply(*rule allI*)
apply(*rule impI*)
apply(*erule-tac x=x* **in** *allE*)
apply(*auto*)
apply(*drule P-exists2-rule*)
apply(*auto*)
done

theorem *P12-imp-Pp-Fp*: *P12*(*p*,*q*,*t*) ==> (*Fp*(*p*,*t*) & *Fp*(*q*,*t*))
apply(*rule conjI*)
apply(*rule-tac P=Fp*(*p*,*t*) **and** *Q=Pp*(*q*,*t*) **in** *conjunct1*)
apply(*rule P1-imp-Fp-Pp*)
apply(*unfold P12-def*)
apply(*erule conjE*)
apply(*assumption*)
apply(*rule-tac Q=Fp*(*q*,*t*) **and** *P=Pp*(*p*,*t*) **in** *conjunct2*)
apply(*rule P2-imp-Pp-Fp*)
apply(*erule conjE*)
apply(*assumption*)
done

theorem *Fp-iff-P1*: (*ALL p t*. (*Fp*(*p*,*t*) <-> *P1*(*p*,*p*,*t*)))
apply(*rule allI*,*rule allI*,*rule iffI*)
apply(*unfold P1-def*)
apply(*rule allI*)
apply(*rule impI*)
apply(*rule-tac x=x* **in** *exI*)
apply(*auto*)
apply(*unfold Fp-def*)
apply(*erule-tac x=x* **in** *allE*)
apply(*auto*)
apply(*unfold E-def*)
apply(*auto*)
apply(*erule-tac x=x* **in** *allE*)

apply(*auto*)
apply(*fold E-def*)
apply(*drule P-exists2-rule*)
apply(*auto*)
done

theorem *P1-iff-P2*: (*ALL p t*. (*P1*(*p*,*p*,*t*) <−> *P2*(*p*,*p*,*t*)))
apply(*auto*)
apply(*unfold P2-def*)
apply(*unfold P1-def*)
apply(*rule allI*)
apply(*rule impI*)
apply(*erule-tac x=y* **in** *allE*)
apply(*drule mp*)
apply(*assumption*)
apply(*erule exE*)
apply(*erule conjE*)
apply(*rule-tac x=y* **in** *exI*)
apply(*rule conjI*)
apply(*assumption*)
apply(*drule P-exists2-rule*)
apply(*erule conjE*)
apply(*unfold E-def*)
apply(*assumption*)
apply(*rule allI*)
apply(*rule impI*)
apply(*erule-tac x=x* **in** *allE*)
apply(*drule mp*)
apply(*assumption*)
apply(*erule exE*)
apply(*erule conjE*)
apply(*rule-tac x=x* **in** *exI*)
apply(*auto*)
apply(*drule P-exists2-rule*)
apply(*unfold E-def*)
apply(*auto*)
done

theorem *P2-iff-P12*: (*ALL p t*. (*P2*(*p*,*p*,*t*) <−> *P12*(*p*,*p*,*t*)))
apply(*auto*)
apply(*unfold P12-def*,*unfold P1-def*,*unfold P2-def*)
apply(*auto*)
apply(*erule-tac x=x* **in** *allE*)
apply(*auto*)
apply(*rule-tac x=x* **in** *exI*)
apply(*auto*)
apply(*drule P-exists2-rule*)
apply(*unfold E-def*)
apply(*auto*)

done

theorem *P1-trans*: $[|P1(p,q,t);P1(q,r,t)|] ==> P1(p,r,t)$
apply(*unfold P1-def*)
apply(*rule allI*)
apply(*rule impI*)
apply(*erule-tac x=x* **in** *allE*)
apply(*drule mp*)
apply(*assumption*)
apply(*erule exE*)
apply(*erule conjE*)
apply(*erule-tac x=y* **in** *allE*)
apply(*drule mp*)
apply(*assumption*)
apply(*erule exE*)
apply(*erule conjE*)
apply(*rule-tac x=ya* **in** *exI*)
apply(*rule conjI*)
apply(*assumption*)
apply(*drule-tac x=x* **and** *y=y* **and** *z=ya* **in** *P-trans-rule*)
apply(*auto*)
done

theorem *P2-trans*: $[|P2(p,q,t);P2(q,r,t)|] ==> P2(p,r,t)$
apply(*unfold P2-def*)
apply(*auto*)
apply(*rotate-tac 1*)
apply(*erule-tac x=y* **in** *allE*)
apply(*auto*)
apply(*erule-tac x=x* **in** *allE*)
apply(*auto*)
apply(*rule-tac x=xa* **in** *exI*)
apply(*auto*)
apply(*drule-tac x=xa* **and** *y=x* **and** *z=y* **in** *P-trans-rule*)
apply(*auto*)
done

theorem *P12-trans*: $[|P12(p,q,t);P12(q,r,t)|] ==> P12(p,r,t)$
apply(*unfold P12-def*)
apply(*auto*)
apply(*drule P1-trans*)
apply(*auto*)
apply(*drule P2-trans*)
apply(*auto*)
done

theorem *DP1-trans*: $[|DP1(p,q,t);DP1(q,r,t)|] ==> DP1(p,r,t)$
apply(*unfold DP1-def*)

apply(*auto*)
apply(*drule P1-trans*)
apply(*auto*)
done

theorem *DP2-trans*: [|$DP2(p,q,t)$;$DP2(q,r,t)$|] ==> $DP2(p,r,t)$
apply(*unfold DP2-def*)
apply(*auto*)
apply(*drule P2-trans*)
apply(*auto*)
done

theorem *DP12-trans*: [|$DP12(p,q,t)$;$DP12(q,r,t)$|] ==> $DP12(p,r,t)$
apply(*unfold DP12-def*)
apply(*auto*)
apply(*drule DP1-trans*)
apply(*auto*)
apply(*rule DP2-trans*)
apply(*auto*)
done

theorem *Subseteq-and-P1-imp-P1-1*: [|$Subseteq(r,p)$;$P1(p,q,t)$|] ==> $P1(r,q,t)$
apply(*unfold Subseteq-def*, *unfold P1-def*)
apply(*clarify*)
apply(*erule-tac x=x* **in** *allE*)
apply(*clarify*)
apply(*erule-tac x=x* **in** *allE*)
apply(*clarify*)
done

theorem *Subseteq-and-P1-imp-P1-2*: [|$Subseteq(q,r)$;$P1(p,q,t)$|] ==> $P1(p,r,t)$
apply(*unfold Subseteq-def*, *unfold P1-def*)
apply(*clarify*)
apply(*rotate-tac 1*)
apply(*erule-tac x=x* **in** *allE*)
apply(*clarify*)
apply(*erule-tac x=y* **in** *allE*)
apply(*clarify*)
apply(*rule-tac x=y* **in** *exI*)
apply(*clarify*)
done

theorem *Subseteq-and-P2-imp-P2-1*: [|$Subseteq(r,q)$;$P2(p,q,t)$|] ==> $P2(p,r,t)$
apply(*unfold Subseteq-def*, *unfold P2-def*)

apply(*clarify*)
apply(*erule-tac x=y* **in** *allE*)
apply(*clarify*)
apply(*erule-tac x=y* **in** *allE*)
apply(*clarify*)
done

theorem *Subseteq-and-P2-imp-P2-2*: [|*Subseteq*(*p*,*r*);*P2*(*p*,*q*,*t*)|] ==> *P2*(*r*,*q*,*t*)
apply(*unfold Subseteq-def*, *unfold P2-def*)
apply(*clarify*)
apply(*rotate-tac 1*)
apply(*erule-tac x=y* **in** *allE*)
apply(*clarify*)
apply(*erule-tac x=x* **in** *allE*)
apply(*clarify*)
apply(*rule-tac x=x* **in** *exI*)
apply(*clarify*)
done

theorem *Subseteq-and-P12-imp-P1-1*: [|*Subseteq*(*r*,*p*);*P12*(*p*,*q*,*t*)|] ==> *P1*(*r*,*q*,*t*)
apply(*unfold P12-def*)
apply(*clarify*)
apply(*drule Subseteq-and-P1-imp-P1-1*)
apply(*auto*)
done

theorem *Subseteq-and-P12-imp-P2-2*: [|*Subseteq*(*p*,*r*);*P12*(*p*,*q*,*t*)|] ==> *P2*(*r*,*q*,*t*)
apply(*unfold P12-def*)
apply(*clarify*)
apply(*drule Subseteq-and-P2-imp-P2-2*)
apply(*auto*)
done

theorem *Subseteq-and-P12-imp-P2-1*: [|*Subseteq*(*r*,*q*);*P12*(*p*,*q*,*t*)|] ==> *P2*(*p*,*r*,*t*)
apply(*unfold P12-def*)
apply(*clarify*)
apply(*drule Subseteq-and-P2-imp-P2-1*)
apply(*auto*)
done

theorem *Subseteq-and-P12-imp-P1-2*: [|*Subseteq*(*q*,*r*);*P12*(*p*,*q*,*t*)|] ==> *P1*(*p*,*r*,*t*)
apply(*unfold P12-def*)
apply(*clarify*)
apply(*drule Subseteq-and-P1-imp-P1-2*)
apply(*auto*)
done

```
theorem Subseteq-and-DP1-imp-DP1: [|Subseteq(r,p);DP1(p,q,t)|] ==> DP1(r,q,t)
apply(unfold DP1-def)
apply(clarify)
apply(frule Subseteq-and-P1-imp-P1-1)
apply(auto)
apply(drule-tac p=p and q=r in DCo-subseteq-rule)
apply(auto)
done

theorem Subseteq-and-DP1-imp-P1: [|Subseteq(q,r);DP1(p,q,t)|] ==> P1(p,r,t)
apply(unfold DP1-def)
apply(clarify)
apply(frule Subseteq-and-P1-imp-P1-2)
apply(auto)
done


theorem Subseteq-and-DP2-imp-DP2: [|Subseteq(r,q);DP2(p,q,t)|] ==> DP2(p,r,t)
apply(unfold DP2-def)
apply(clarify)
apply(frule Subseteq-and-P2-imp-P2-1)
apply(auto)
apply(drule-tac p=q and q=r in DCo-subseteq-rule)
apply(auto)
done

theorem Subseteq-and-DP2-imp-P2: [|Subseteq(p,r);DP2(p,q,t)|] ==> P2(r,q,t)
apply(unfold DP2-def)
apply(clarify)
apply(frule Subseteq-and-P2-imp-P2-2)
apply(auto)
done


theorem Subseteq-and-DP12-imp-DP1: [|Subseteq(r,p);DP12(p,q,t)|] ==> DP1(r,q,t)
apply(unfold DP12-def)
apply(clarify)
apply(rule Subseteq-and-DP1-imp-DP1)
apply(auto)
done


theorem Subseteq-and-DP12-imp-P2: [|Subseteq(p,r);DP12(p,q,t)|] ==> P2(r,q,t)
apply(unfold DP12-def)
apply(clarify)
apply(rule Subseteq-and-DP2-imp-P2)
```

apply(*auto*)
done

theorem *Subseteq-and-DP12-imp-DP2*: [|*Subseteq*(*r*,*q*);*DP12*(*p*,*q*,*t*)|] ==> *DP2*(*p*,*r*,*t*)
apply(*unfold DP12-def*)
apply(*clarify*)
apply(*rule Subseteq-and-DP2-imp-DP2*)
apply(*auto*)
done

theorem *Subseteq-and-DP12-imp-P1*: [|*Subseteq*(*q*,*r*);*DP12*(*p*,*q*,*t*)|] ==> *P1*(*p*,*r*,*t*)
apply(*unfold DP12-def*)
apply(*clarify*)
apply(*rule Subseteq-and-DP1-imp-P1*)
apply(*auto*)
done

end

theory *UniversalParthood*

imports *ExtensionsOfUniversals PartonomicInclusion*

begin

consts

UPt1 :: *Un* => *Un* => *Ti* => *o*
UPt2 :: *Un* => *Un* => *Ti* => *o*
UPt12 :: *Un* => *Un* => *Ti* => *o*

UP1 :: *Un* => *Un* => *o*
UP2 :: *Un* => *Un* => *o*
UP12 :: *Un* => *Un* => *o*

defs

UPt1-def: *UPt1*(*c*,*d*,*t*) == (*ALL x*. (*Inst*(*x*,*c*,*t*) --> (*EX y*. (*Inst*(*y*,*d*,*t*) & *P*(*x*,*y*,*t*)))))
UPt2-def: *UPt2*(*c*,*d*,*t*) == (*ALL y*. (*Inst*(*y*,*d*,*t*) --> (*EX x*. (*Inst*(*x*,*c*,*t*) & *P*(*x*,*y*,*t*)))))
UPt12-def: *UPt12*(*c*,*d*,*t*) == *UPt1*(*c*,*d*,*t*) & *UPt2*(*c*,*d*,*t*)

UP1-def: *UP1*(*c*,*d*) == (*ALL t*. *UPt1*(*c*,*d*,*t*))
UP2-def: *UP2*(*c*,*d*) == (*ALL t*. *UPt2*(*c*,*d*,*t*))
UP12-def: *UP12*(*c*,*d*) == (*ALL t*. *UPt12*(*c*,*d*,*t*))

theorem *ExtCo-and-ExtCo*: [|*ExtCo*(p,c,t);*ExtCo*(q,d,t)|] ==> (*P1*(p,q,t) <−> *UPt1*(c,d,t))
apply(*unfold iff-def*)
apply(*unfold ExtCo-def*)
apply(*unfold P1-def*)
apply(*unfold UPt1-def*)
apply(*auto*)
done

theorem *ExtOb-and-ExtOb*: [|*ExtOb*(x,c,t);*ExtOb*(y,d,t)|] ==> (*P*(x,y,t) <−> *UPt1*(c,d,t))
apply(*unfold iff-def*)
apply(*unfold ExtOb-def*)
apply(*unfold UPt1-def*)
apply(*rule conjI*)
apply(*rule impI*)
apply(*rule allI*)
apply(*rule impI*)
apply(*erule conjE*)
apply(*erule conjE*)
apply(*rule-tac x=y* **in** *exI*)
apply(*rule conjI*)
apply(*assumption*)
apply(*erule-tac x=xa* **in** *allE*)
apply(*drule mp*)
apply(*assumption*)
apply(*erule-tac a=x* **and** *b=xa* **in** *subst*)
apply(*assumption*)
apply(*rule impI*)
apply(*erule-tac x=x* **in** *allE*)
apply(*erule conjE*)
apply(*erule conjE*)
apply(*drule mp*)
apply(*assumption*)
apply(*erule exE*)
apply(*erule conjE*)
apply(*rotate-tac 3*)
apply(*erule-tac x=ya* **in** *allE*)
apply(*drule mp*)
apply(*assumption*)
apply(*drule-tac a=y* **and** *b=ya* **in** *sym*)
apply(*erule-tac a=ya* **and** *b=y* **in** *subst*)
apply(*assumption*)
done

theorem *ExtCo-and-ExtOb*: [|*ExtCo*(p,c,t);*ExtOb*(x,d,t)|] ==> ((*EX* y. (*Inst*(y,c,t) & *P*(y,x,t)))) <−> *UPt2*(c,d,t)

apply(*unfold iff-def*)
apply(*unfold UPt2-def*)
apply(*unfold ExtCo-def*)
apply(*unfold ExtOb-def*)
apply(*auto*)
done

theorem *ExtOb-and-ExtCo*: [|*ExtOb*(*x*,*c*,*t*);*ExtCo*(*p*,*d*,*t*)|] ==> ((*EX y*. (*Inst*(*y*,*d*,*t*) & *P*(*x*,*y*,*t*)))) <−> *UPt1*(*c*,*d*,*t*)
apply(*unfold iff-def*)
apply(*unfold UPt1-def*)
apply(*unfold ExtOb-def*)
apply(*unfold ExtCo-def*)
apply(*auto*)
done

theorem *UPt1-refl*: (*ALL c t*. *UPt1*(*c*,*c*,*t*))
apply(*unfold UPt1-def*)
apply(*auto*)
apply(*rule-tac x*=*x* **in** *exI*)
apply(*auto*)
apply(*insert Inst-E*)
apply(*unfold E-def*)
apply(*auto*)
done

theorem *UPt2-refl*: (*ALL c t*. *UPt2*(*c*,*c*,*t*))
apply(*unfold UPt2-def*)
apply(*auto*)
apply(*rule-tac x*=*y* **in** *exI*)
apply(*auto*)
apply(*insert Inst-E*)
apply(*unfold E-def*)
apply(*auto*)
done

theorem *UPt12-refl*: (*ALL c t*. *UPt12*(*c*,*c*,*t*))
apply(*unfold UPt12-def*)
apply(*insert UPt1-refl*)
apply(*insert UPt2-refl*)
apply(*auto*)
done

theorem *UPt1-trans*: [|*UPt1*(*c*,*d*,*t*);*UPt1*(*d*,*e*,*t*)|]==> *UPt1*(*c*,*e*,*t*)
apply(*unfold UPt1-def*)
apply(*auto*)

apply(*erule-tac x=x* **in** *allE*)
apply(*auto*)
apply(*erule-tac x=y* **in** *allE*)
apply(*auto*)
apply(*rule-tac x=ya* **in** *exI*)
apply(*drule-tac x=x* **and** *y=y* **and** *z=ya* **in** *P-trans-rule*)
apply(*auto*)
done

theorem *UPt2-trans*: [|*UPt2*(*c*,*d*,*t*);*UPt2*(*d*,*e*,*t*)|]==> *UPt2*(*c*,*e*,*t*)
apply(*unfold UPt2-def*)
apply(*auto*)
apply(*rotate-tac 1*)
apply(*erule-tac x=y* **in** *allE*)
apply(*auto*)
apply(*erule-tac x=x* **in** *allE*)
apply(*auto*)
apply(*rule-tac x=xa* **in** *exI*)
apply(*drule-tac x=xa* **and** *y=x* **and** *z=y* **in** *P-trans-rule*)
apply(*auto*)
done

theorem *UPt12-trans*: [|*UPt12*(*c*,*d*,*t*);*UPt12*(*d*,*e*,*t*)|]==> *UPt12*(*c*,*e*,*t*)
apply(*unfold UPt12-def*)
apply(*auto*)
apply(*drule UPt1-trans*)
apply(*auto*)
apply(*drule UPt2-trans*)
apply(*auto*)
done

theorem *UP1-iff*: (*ALL c d*. (*UP1*(*c*,*d*) <−> (*ALL t x*. (*Inst*(*x*,*c*,*t*) −−> (*EX y*. (*Inst*(*y*,*d*,*t*) & *P*(*x*,*y*,*t*)))))))
apply(*unfold UP1-def*,*unfold UPt1-def*)
apply(*auto*)
done

theorem *UP2-iff*: (*ALL c d*. (*UP2*(*c*,*d*) <−> (*ALL t y*. (*Inst*(*y*,*d*,*t*) −−> (*EX x*. (*Inst*(*x*,*c*,*t*) & *P*(*x*,*y*,*t*)))))))
apply(*unfold UP2-def*,*unfold UPt2-def*)
apply(*auto*)
done

theorem *UP1-refl*: (*ALL c*. *UP1*(*c*,*c*))
apply(*unfold UP1-def*)
apply(*insert UPt1-refl*)
apply(*auto*)

done

theorem *UP2-refl*: (*ALL c. UP2*(*c*,*c*))
apply(*unfold UP2-def*)
apply(*insert UPt2-refl*)
apply(*auto*)
done

theorem *UP12-refl*: (*ALL c. UP12*(*c*,*c*))
apply(*unfold UP12-def*)
apply(*insert UPt12-refl*)
apply(*auto*)
done

theorem *UP1-trans*: [|*UP1*(*c*,*d*);*UP1*(*d*,*e*)|]==> *UP1*(*c*,*e*)
apply(*unfold UP1-def*)
apply(*auto*)
apply(*drule-tac x*=*t* **in** *allE*)
apply(*drule-tac x*=*t* **in** *allE*)
apply(*drule UPt1-trans*)
apply(*auto*)
done

theorem *UP2-trans*: [|*UP2*(*c*,*d*);*UP2*(*d*,*e*)|]==> *UP2*(*c*,*e*)
apply(*unfold UP2-def*)
apply(*auto*)
apply(*drule-tac x*=*t* **in** *allE*)
apply(*drule-tac x*=*t* **in** *allE*)
apply(*drule UPt2-trans*)
apply(*auto*)
done

theorem *UP12-trans*: [|*UP12*(*c*,*d*);*UP12*(*d*,*e*)|]==> *UP12*(*c*,*e*)
apply(*unfold UP12-def*)
apply(*auto*)
apply(*drule-tac x*=*t* **in** *allE*)
apply(*drule-tac x*=*t* **in** *allE*)
apply(*drule UPt12-trans*)
apply(*auto*)
done

theorem *UP12-impl-UP1-and-UP2*: *UP12*(*c*,*d*) ==> (*UP1*(*c*,*d*) & *UP2*(*c*,*d*))
apply(*unfold UP12-def*,*unfold UP1-def*,*unfold UP2-def*)
apply(*rule conjI*)
apply(*rule allI*)
apply(*erule-tac x*=*t* **in** *allE*)
apply(*unfold UPt12-def*)

apply(*auto*)
done

theorem *IsA-and-UP1-impl-UP1-1*: [|*IsA*(*e*,*c*);*UP1*(*c*,*d*)|] ==> *UP1*(*e*,*d*)
apply(*unfold UP1-def*,*unfold UPt1-def*)
apply(*drule-tac c*=*e* **and** *d*=*c* **in** *Inst-IsA-rule*)
apply(*auto*)
done

theorem *IsA-and-UP1-impl-UP1-2*: [|*IsA*(*d*,*e*);*UP1*(*c*,*d*)|] ==> *UP1*(*c*,*e*)
apply(*unfold UP1-def*,*unfold UPt1-def*)
apply(*drule-tac c*=*d* **and** *d*=*e* **in** *Inst-IsA-rule*)
apply(*auto*)
apply(*erule-tac x*=*t* **in** *allE*)
apply(*rotate-tac 2*)
apply(*erule-tac x*=*x* **in** *allE*)
apply(*auto*)
done

theorem *IsA-and-UP2-impl-UP2-1*: [|*IsA*(*e*,*d*);*UP2*(*c*,*d*)|] ==> *UP2*(*c*,*e*)
apply(*unfold UP2-def*,*unfold UPt2-def*)
apply(*drule-tac c*=*e* **and** *d*=*d* **in** *Inst-IsA-rule*)
apply(*auto*)
done

theorem *IsA-and-UP2-impl-UP2-2*: [|*IsA*(*c*,*e*);*UP2*(*c*,*d*)|] ==> *UP2*(*e*,*d*)
apply(*unfold UP2-def*,*unfold UPt2-def*)
apply(*drule-tac c*=*c* **and** *d*=*e* **in** *Inst-IsA-rule*)
apply(*auto*)
apply(*erule-tac x*=*t* **in** *allE*)
apply(*rotate-tac 2*)
apply(*erule-tac x*=*y* **in** *allE*)
apply(*auto*)
done

theorem *IsA-and-UP12-impl-UP1-1*: [|*IsA*(*e*,*c*);*UP12*(*c*,*d*)|] ==> *UP1*(*e*,*d*)
apply(*drule UP12-impl-UP1-and-UP2*)
apply(*erule conjE*)
apply(*drule IsA-and-UP1-impl-UP1-1*)
apply(*auto*)
done

theorem *IsA-and-UP12-impl-UP2-2*: [|*IsA*(*c*,*e*);*UP12*(*c*,*d*)|] ==> *UP2*(*e*,*d*)
apply(*drule UP12-impl-UP1-and-UP2*)
apply(*erule conjE*)
apply(*drule IsA-and-UP2-impl-UP2-2*)

apply(*auto*)
done

theorem *IsA-and-UP12-impl-UP2-1*: [|*IsA*(*e*,*d*);*UP12*(*c*,*d*)|] ==> *UP2*(*c*,*e*)
apply(*drule UP12-impl-UP1-and-UP2*)
apply(*erule conjE*)
apply(*drule IsA-and-UP2-impl-UP2-1*)
apply(*auto*)
done

theorem *IsA-and-UP12-impl-UP1-2*: [|*IsA*(*d*,*e*);*UP12*(*c*,*d*)|] ==> *UP1*(*c*,*e*)
apply(*drule UP12-impl-UP1-and-UP2*)
apply(*erule conjE*)
apply(*drule IsA-and-UP1-impl-UP1-2*)
apply(*auto*)
done

theorem *IsA-imp-UP1*: *IsA*(*c*,*d*) ==> *UP1*(*c*,*d*)
apply(*unfold UP1-def*)
apply(*safe*)
apply(*unfold UPt1-def*)
apply(*safe*)
apply(*drule Inst-IsA-rule* [*of c d*])
apply(*erule-tac x*=*x* **in** *allE*)
apply(*erule-tac x*=*t* **in** *allE*)
apply(*safe*)
apply(*rule-tac x*=*x* **in** *exI*)
apply(*insert Inst-E*)
apply(*insert P-refl*)
apply(*auto*)
done

end

theory *BFO*

imports *QDistR Adjacency TMTL SumsPartitions UniversalParthood*

begin

end