Compactness Theorem for Propositional Logic and Combinatorial Applications

Fabián Fernando Serrano Suárez†, Thaynara Arielly de Lima⋆, Mauricio Ayala-Rincón‡

† Universidad Nacional de Colombia - Sede Manizales, Colombia
⋆Universidade Federal de Goiás, Goiânia, Brazil
‡Universidade de Brasília, Brasília D.F., Brazil

March 17, 2025

Abstract

This theory formalises the compactness theorem for propositional logic based on the model existence theorem approach. It also presents applications of the compactness theorem to formalize combinatorial theorems over countable structures: the de Bruijn-Erdős Graph coloring theorem for countable graphs, König's Lemma, and set- and graph-theoretical versions of Hall's Theorem for countable families of sets and graphs.

Contents

theory *Background-on-graphs*

imports *Main*

begin

1 Special Graph Theoretical Notions

This theory provides a background on specialized graph notions and properties. We follow the approach by L. Noschinski available in the AFPs. Since not all elements of Noschinski theory are required, we prefer not to import it.

The proof are desiccated in several steps since the focus is clarity instead proof automation.

record $('a,'b)$ *pre-digraph* =
  verts :: *'a set*
  arcs :: *'b set*
  tail :: *'b* ⇒ *'a*
  head :: *'b* ⇒ *'a*

definition *tails*:: $('a,'b)$ *pre-digraph* ⇒ *'a set* **where**
  tails G ≡ {*tail G e* |*e. e* ∈ *arcs G* }

definition *tails-set* :: $('a,'b)$ *pre-digraph* ⇒ *'b set* ⇒ *'a set* **where**
  tails-set G E ≡ { *tail G e* |*e. e* ∈ *E* ∧ *E* ⊆ *arcs G* }

definition *heads*:: $('a,'b)$ *pre-digraph* ⇒ *'a set* **where**
  heads G ≡ { *head G e* |*e. e* ∈ *arcs G* }

definition *heads-set*:: $('a,'b)$ *pre-digraph* ⇒ *'b set* ⇒ *'a set* **where**
  heads-set G E ≡ { *head G e* |*e. e* ∈ *E* ∧ *E* ⊆ *arcs G* }

definition *neighbour*:: $('a,'b)$ *pre-digraph* ⇒ *'a* ⇒ *'a* ⇒ *bool* **where**
  neighbour G v u ≡
  ∃ *e. e*∈ (*arcs G*) ∧ ((*head G e* = *v* ∧ *tail G e* = *u*) ∨
  (*head G e* = *u* ∧ *tail G e* = *v*))

definition *neighbourhood*:: $('a,'b)$ *pre-digraph* ⇒ *'a* ⇒ *'a set* **where**
  neighbourhood G v ≡ {*u* |*u. neighbour G u v*}

definition *bipartite-digraph*:: $('a,'b)$ *pre-digraph* ⇒ *'a set* ⇒ *'a set* ⇒ *bool* **where**
  bipartite-digraph G X Y ≡

$(X \cup Y = (verts\ G)) \wedge\ \ X \cap Y = \{\} \wedge$
$(\forall e \in (arcs\ G).(tail\ G\ e) \in X \longleftrightarrow (head\ G\ e) \in Y)$

definition *dir-bipartite-digraph*:: $('a,'b)$ *pre-digraph* $\Rightarrow$ *'a set* $\Rightarrow$ *'a set* $\Rightarrow$ *bool*
where
dir-bipartite-digraph $G\ X\ Y \equiv (bipartite\text{-}digraph\ G\ X\ Y) \wedge$
$((tails\ G = X) \wedge\ (\forall\ e1 \in arcs\ G.\ \forall\ e2 \in arcs\ G.\ e1 = e2 \longleftrightarrow head\ G\ e1 = head\ G\ e2 \wedge tail\ G\ e1 = tail\ G\ e2))$

definition *K-E-bipartite-digraph*:: $('a,'b)$ *pre-digraph* $\Rightarrow$ *'a set* $\Rightarrow$ *'a set* $\Rightarrow$ *bool*
where
K-E-bipartite-digraph $G\ X\ Y \equiv$
$(dir\text{-}bipartite\text{-}digraph\ G\ X\ Y) \wedge (\forall x{\in}X.\ \ finite\ (neighbourhood\ G\ x))$

definition *dirBD-matching*:: $('a,'b)$ *pre-digraph* $\Rightarrow$ *'a set* $\Rightarrow$ *'a set* $\Rightarrow$ *'b set* $\Rightarrow$ *bool*
where
dirBD-matching $G\ X\ Y\ E\ \equiv$
$dir\text{-}bipartite\text{-}digraph\ G\ X\ Y \wedge (E\ \subseteq\ (arcs\ G)) \wedge$
$(\forall\ e1{\in}E.\ (\forall\ e2{\in}\ E.\ \ e1 \neq e2 \longrightarrow$
$((head\ G\ e1) \neq (head\ G\ e2)) \wedge$
$((tail\ G\ e1) \neq (tail\ G\ e2))))$

lemma *tail-head*:
assumes *dir-bipartite-digraph* $G\ X\ Y$ **and** $e \in arcs\ G$
shows $(tail\ G\ e) \in X \wedge (head\ G\ e) \in Y$
$\langle proof \rangle$

lemma *tail-head1*:
assumes *dirBD-matching* $G\ X\ Y\ E$ **and** $e \in E$
shows $(tail\ G\ e) \in X \wedge (head\ G\ e) \in Y$
$\langle proof \rangle$

lemma *dirBD-matching-tail-edge-unicity*:
dirBD-matching $G\ X\ Y\ E \longrightarrow$
$(\forall e1 \in E.\ (\forall\ e2{\in}\ E.\ (tail\ G\ e1 = tail\ G\ e2) \longrightarrow e1 = e2))$

$\langle proof \rangle$

lemma *dirBD-matching-head-edge-unicity*:
dirBD-matching $G\ X\ Y\ E \longrightarrow$
$(\forall e1 \in E.\ (\forall\ e2{\in}\ E.\ (head\ G\ e1 = head\ G\ e2) \longrightarrow e1 = e2))$

$\langle proof \rangle$

definition *dirBD-perfect-matching*::

$('a,'b)$ *pre-digraph* $\Rightarrow$ *'a set* $\Rightarrow$ *'a set* $\Rightarrow$ *'b set* $\Rightarrow$ *bool*
where
dirBD-perfect-matching G X Y E $\equiv$
dirBD-matching G X Y E $\wedge$ (*tails-set G E* = *X*)

lemma *Tail-covering-edge-in-Pef-matching*:
$\forall x \in X$. *dirBD-perfect-matching G X Y E* $\longrightarrow$ ($\exists$ *e* $\in$ *E*. *tail G e* = *x*)
⟨*proof*⟩

lemma *Edge-unicity-in-dirBD-P-matching*:
$\forall x \in X$. *dirBD-perfect-matching G X Y E* $\longrightarrow$ ($\exists$!*e* $\in$ *E*. *tail G e* = *x*)

⟨*proof*⟩

definition *E-head* :: $('a,'b)$ *pre-digraph* $\Rightarrow$ *'b set* $\Rightarrow$ (*'a* $\Rightarrow$ *'a*)
where
E-head G E = (λx. (*THE y*. $\exists$ *e*. *e* $\in$ *E* $\wedge$ *tail G e* = *x* $\wedge$ *head G e* = *y*))

lemma *unicity-E-head1*:
assumes *dirBD-matching G X Y E* $\wedge$ *e* $\in$ *E* $\wedge$ *tail G e* = *x* $\wedge$ *head G e* = *y*
shows ($\forall$ *z*.($\exists$ *e*. *e* $\in$ *E* $\wedge$ *tail G e* = *x* $\wedge$ *head G e* = *z*)$\longrightarrow$ *z* = *y*)
⟨*proof*⟩

lemma *unicity-E-head2*:
assumes *dirBD-matching G X Y E* $\wedge$ *e* $\in$ *E* $\wedge$ *tail G e* = *x* $\wedge$ *head G e* = *y*
shows (*THE a*. $\exists$ *e*. *e* $\in$ *E* $\wedge$ *tail G e* = *x* $\wedge$ *head G e* = *a*) = *y*
⟨*proof*⟩

lemma *unicity-E-head*:
assumes *dirBD-matching G X Y E* $\wedge$ *e* $\in$ *E* $\wedge$ *tail G e* = *x* $\wedge$ *head G e* = *y*
shows (*E-head G E*) *x* = *y*
⟨*proof*⟩

lemma *E-head-image* :
dirBD-perfect-matching G X Y E $\longrightarrow$
(*e* $\in$ *E* $\wedge$ *tail G e* = *x* $\longrightarrow$ (*E-head G E*) *x* = *head G e*)

⟨*proof*⟩

lemma *E-head-in-neighbourhood*:
dirBD-matching G X Y E $\longrightarrow$ *e* $\in$ *E* $\longrightarrow$ *tail G e* = *x* $\longrightarrow$
(*E-head G E*) *x* $\in$ *neighbourhood G x*

⟨*proof*⟩

lemma *dirBD-matching-inj-on*:
dirBD-perfect-matching G X Y E $\longrightarrow$ *inj-on* (*E-head G E*) *X*

⟨*proof*⟩

end

datatype *'b formula* =
  FF
  | *TT*
  | *atom 'b*
  | *Negation 'b formula* (‹¬.(-)› [*110*] *110*)
  | *Conjunction 'b formula 'b formula* (**infixl** ‹∧.› *109*)
  | *Disjunction 'b formula 'b formula* (**infixl** ‹∨.› *108*)
  | *Implication 'b formula 'b formula* (**infixl** ‹→.› *100*)

lemma (¬.¬. *Atom P* →. *Atom Q* →. *Atom R*) =
  (((¬. (¬. *Atom P*)) →. *Atom Q*) →. *Atom R*)
⟨*proof*⟩

datatype *v-truth* = *Ttrue* | *Ffalse*

definition *v-negation* :: *v-truth* ⇒ *v-truth* **where**
v-negation x ≡ (*if x* = *Ttrue then Ffalse else Ttrue*)

definition *v-conjunction* :: *v-truth* ⇒ *v-truth* ⇒ *v-truth* **where**
v-conjunction x y ≡ (*if x* = *Ffalse then Ffalse else y*)

definition *v-disjunction* :: *v-truth* ⇒ *v-truth* ⇒ *v-truth* **where**
v-disjunction x y ≡ (*if x* = *Ttrue then Ttrue else y*)

definition *v-implication* :: *v-truth* ⇒ *v-truth* ⇒ *v-truth* **where**
v-implication x y ≡ (*if x* = *Ffalse then Ttrue else y*)

primrec *t-v-evaluation* :: (*'b* ⇒ *v-truth*) ⇒ *'b formula* ⇒ *v-truth*
where
  t-v-evaluation I FF = *Ffalse*
| *t-v-evaluation I TT* = *Ttrue*
| *t-v-evaluation I* (*atom p*) = *I p*
| *t-v-evaluation I* (¬. *F*) = (*v-negation* (*t-v-evaluation I F*))
| *t-v-evaluation I* (*F* ∧. *G*) = (*v-conjunction* (*t-v-evaluation I F*) (*t-v-evaluation I G*))
| *t-v-evaluation I* (*F* ∨. *G*) = (*v-disjunction* (*t-v-evaluation I F*) (*t-v-evaluation I G*))
| *t-v-evaluation I* (*F* →. *G*) = (*v-implication* (*t-v-evaluation I F*) (*t-v-evaluation I G*))

lemma *Bivaluation*:
shows *t-v-evaluation I F* = *Ttrue* ∨ *t-v-evaluation I F* = *Ffalse*⟨*proof*⟩

lemma *NegationValues1*:
assumes *t-v-evaluation I* (¬.*F*) = *Ffalse*
shows *t-v-evaluation I F* = *Ttrue*⟨*proof*⟩

lemma *NegationValues2*:
assumes *t-v-evaluation I* (¬.*F*) = *Ttrue*
shows *t-v-evaluation I F* = *Ffalse*⟨*proof*⟩
lemma *non-Ttrue*:
assumes *t-v-evaluation I F* ≠ *Ttrue* **shows** *t-v-evaluation I F* = *Ffalse*⟨*proof*⟩

lemma *ConjunctionValues*:
assumes *t-v-evaluation I* (*F* ∧. *G*) = *Ttrue*
shows *t-v-evaluation I F* = *Ttrue* ∧ *t-v-evaluation I G* = *Ttrue*⟨*proof*⟩

lemma *DisjunctionValues*:
assumes *t-v-evaluation I* (*F* ∨. *G*) = *Ttrue*
shows *t-v-evaluation I F* = *Ttrue* ∨ *t-v-evaluation I G* = *Ttrue*⟨*proof*⟩

lemma *ImplicationValues*:
assumes *t-v-evaluation I* (*F* →. *G*) = *Ttrue*
shows *t-v-evaluation I F* = *Ttrue* ⟶ *t-v-evaluation I G* = *Ttrue*⟨*proof*⟩⟨*proof*⟩

definition *model* :: (′*b* ⇒ *v-truth*) ⇒ ′*b formula set* ⇒ *bool* (‹- *model* -› [*80*,*80*] *80*) **where**
I model S ≡ (∀ *F* ∈ *S*. *t-v-evaluation I F* = *Ttrue*)

definition *satisfiable* :: ′*b formula set* ⇒ *bool* **where**
satisfiable S ≡ (∃ *v*. *v model S*)
⟨*proof*⟩

definition *consequence* :: ′*b formula set* ⇒ ′*b formula* ⇒ *bool* (‹- ⊨ -› [*80*,*80*] *80*) **where**
S ⊨ *F* ≡ (∀ *I*. *I model S* ⟶ *t-v-evaluation I F* = *Ttrue*)

⟨*proof*⟩⟨*proof*⟩

theorem *EquiConsSat*:
shows *S* ⊨ *F* = (¬ *satisfiable* (*S* ∪ {¬. *F*}))⟨*proof*⟩

definition *tautology* :: ′*b formula* ⇒ *bool* **where**
tautology F ≡ (∀ *I*. ((*t-v-evaluation I F*) = *Ttrue*))

lemma *tautology* (*F* →. (*G* →. *F*))
⟨*proof*⟩⟨*proof*⟩

theorem *CNS-tautology*: $tautology\ F = (\{\} \models F)\langle proof\rangle$

theorem *TautSatis*:
shows $tautology\ (F \to.\ G) = (\neg\ satisfiable\{F, \neg.G\})\langle proof\rangle\langle proof\rangle\langle proof\rangle\langle proof\rangle$

fun *FormulaLiteral* :: *'b formula* ⇒ *bool* **where**
FormulaLiteral FF = *True*
| *FormulaLiteral* (¬. *FF*) = *True*
| *FormulaLiteral TT* = *True*
| *FormulaLiteral* (¬. *TT*) = *True*
| *FormulaLiteral* (*atom P*) = *True*
| *FormulaLiteral* (¬.(*atom P*)) = *True*
| *FormulaLiteral F* = *False*

fun *FormulaNoNo* :: *'b formula* ⇒ *bool* **where**
FormulaNoNo (¬. (¬. *F*)) = *True*
| *FormulaNoNo F* = *False*

fun *FormulaAlfa* :: *'b formula* ⇒ *bool* **where**
FormulaAlfa (*F* ∧. *G*) = *True*
| *FormulaAlfa* (¬. (*F* ∨. *G*)) = *True*
| *FormulaAlfa* (¬. (*F* →. *G*)) = *True*
| *FormulaAlfa F* = *False*

fun *FormulaBeta* :: *'b formula* ⇒ *bool* **where**
FormulaBeta (*F* ∨. *G*) = *True*
| *FormulaBeta* (¬. (*F* ∧. *G*)) = *True*
| *FormulaBeta* (*F* →. *G*) = *True*
| *FormulaBeta F* = *False*
⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩
lemma *noLiteralNoNo*:
assumes *FormulaLiteral formula*
shows ¬(*FormulaNoNo formula*)
⟨*proof*⟩

lemma *noLiteralAlfa*:
assumes *FormulaLiteral formula*
shows ¬(*FormulaAlfa formula*)
⟨*proof*⟩

lemma *noLiteralBeta*:
  assumes *FormulaLiteral formula*
  shows ¬(*FormulaBeta formula*)
⟨*proof*⟩

lemma *noAlfaNoNo*:
  assumes *FormulaAlfa formula*
  shows ¬(*FormulaNoNo formula*)
⟨*proof*⟩

lemma *noBetaNoNo*:
  assumes *FormulaBeta formula*
  shows ¬(*FormulaNoNo formula*)
⟨*proof*⟩

lemma *noAlfaBeta*:
  assumes *FormulaAlfa formula*
  shows ¬(*FormulaBeta formula*)
⟨*proof*⟩

lemma *UniformNotation*:
 FormulaLiteral F ∨ *FormulaNoNo F* ∨ *FormulaAlfa F* ∨ *FormulaBeta F*⟨*proof*⟩

datatype *typeUniformNotation* = *Literal* | *NoNo* | *Alfa*| *Beta*

fun *typeFormula* :: *'b formula* ⇒ *typeUniformNotation* **where**
typeFormula F =
  (*if FormulaBeta F then Beta*
   else if FormulaNoNo F then NoNo
   else if FormulaAlfa F then Alfa
   else Literal)
⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩

fun *componentes* :: *'b formula* ⇒ *'b formula list* **where**
  componentes (¬. (¬. *G*)) = [*G*]
| *componentes* (*G* ∧. *H*) = [*G*, *H*]
| *componentes* (¬. (*G* ∨. *H*)) = [¬. *G*, ¬. *H*]
| *componentes* (¬. (*G* →. *H*)) = [*G*, ¬. *H*]
| *componentes* (*G* ∨. *H*) = [*G*, *H*]
| *componentes* (¬. (*G* ∧. *H*)) = [¬. *G*, ¬. *H*]
| *componentes* (*G* →. *H*) = [¬.*G*, *H*]

definition *Comp1* :: *'b formula* ⇒ *'b formula* **where**
Comp1 F = *hd* (*componentes F*)

definition *Comp2* :: *'b formula* ⇒ *'b formula* **where**
Comp2 F = *hd* (*tl* (*componentes F*))

primrec *t-v-evaluationDisyuncionG* :: (*'b* ⇒ *v-truth*) ⇒ (*'b formula list*) ⇒ *v-truth*
where
t-v-evaluationDisyuncionG I [] = *Ffalse*
| *t-v-evaluationDisyuncionG I* (*F#Fs*) = (*if t-v-evaluation I F* = *Ttrue then Ttrue*
else t-v-evaluationDisyuncionG I Fs)

primrec *t-v-evaluationConjuncionG* :: (*'b* ⇒ *v-truth*) ⇒ (*'b formula list*) *list* ⇒
v-truth **where**
t-v-evaluationConjuncionG I [] = *Ttrue*
| *t-v-evaluationConjuncionG I* (*D#Ds*) =
(*if t-v-evaluationDisyuncionG I D* = *Ffalse then Ffalse else t-v-evaluationConjuncionG*
I Ds)

definition *equivalentesG* :: (*'b formula list*) *list* ⇒ (*'b formula list*) *list* ⇒ *bool*
where
equivalentesG C1 C2 ≡ (∀ *I*. ((*t-v-evaluationConjuncionG I C1*) = (*t-v-evaluationConjuncionG*
I C2)))

⟨*proof*⟩

lemma *EquiNoNo*:
assumes *typeFormula F* = *NoNo*
shows *equivalentesG* [[*F*]] [[*Comp1 F*]]⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩

lemma *EquiAlfa*:
assumes *typeFormula F* = *Alfa*
shows *equivalentesG* [[*F*]] [[*Comp1 F*],[*Comp2 F*]]⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩

lemma *EquiBeta*:
assumes *typeFormula F* = *Beta*
shows *equivalentesG* [[*F*]] [[*Comp1 F*, *Comp2 F*]]⟨*proof*⟩

lemma *EquivNoNoComp*:
assumes *typeFormula F* = *NoNo*
shows *equivalent F* (*Comp1 F*)⟨*proof*⟩

lemma *EquivAlfaComp*:

assumes *typeFormula F* = *Alfa*
shows *equivalent F* (*Comp1 F* ∧. *Comp2 F*)⟨*proof*⟩

lemma *EquivBetaComp*:
assumes *typeFormula F* = *Beta*
shows *equivalent F* (*Comp1 F* ∨. *Comp2 F*)⟨*proof*⟩

definition *consistenceP* :: *'b formula set set* ⇒ *bool* **where**
consistenceP $\mathcal{C}$ =
(∀ *S*. *S* ∈ $\mathcal{C}$ ⟶ (∀ *P*. ¬ (*atom P* ∈ *S* ∧ (¬.*atom P*) ∈ *S*)) ∧
FF ∉ *S* ∧ (¬.*TT*) ∉ *S* ∧
(∀ *F*. (¬.¬.*F*) ∈ *S* ⟶ *S* ∪ {*F*} ∈ $\mathcal{C}$) ∧
(∀ *F*. ((*FormulaAlfa F*) ∧ *F*∈*S*) ⟶ (*S*∪{*Comp1 F*, *Comp2 F*}) ∈ $\mathcal{C}$) ∧
(∀ *F*. ((*FormulaBeta F*) ∧ *F*∈*S*) ⟶ (*S*∪{*Comp1 F*}∈$\mathcal{C}$) ∨ (*S*∪{*Comp2 F*}∈$\mathcal{C}$)))

definition *subset-closed* :: *'a set set* ⇒ *bool* **where**
subset-closed $\mathcal{C}$ = (∀ *S* ∈ $\mathcal{C}$. ∀ *S'*. *S'* ⊆ *S* ⟶ *S'* ∈ $\mathcal{C}$)

unbundle *no trancl-syntax*

definition *closure-subset* :: *'a set set* ⇒ *'a set set* (‹-$^+$›[*1000*] *1000*) **where**
$\mathcal{C}^+$ = {*S*. ∃ *S'* ∈ $\mathcal{C}$. *S* ⊆ *S'*}

lemma *closed-subset*: $\mathcal{C} \subseteq \mathcal{C}^+$
⟨*proof*⟩

lemma *closed-closed*: *subset-closed* ($\mathcal{C}^+$)
⟨*proof*⟩

lemma *cond-consistP1*:
assumes *consistenceP* $\mathcal{C}$ **and** *T* ∈ $\mathcal{C}$ **and** *S* ⊆ *T*
shows (∀ *P*. ¬(*atom P* ∈ *S* ∧ (¬.*atom P*) ∈ *S*))⟨*proof*⟩
lemma *cond-consistP2*:
assumes *consistenceP* $\mathcal{C}$ **and** *T* ∈ $\mathcal{C}$ **and** *S* ⊆ *T*
shows *FF* ∉ *S* ∧ (¬.*TT*)∉ *S*⟨*proof*⟩
lemma *cond-consistP3*:
assumes *consistenceP* $\mathcal{C}$ **and** *T* ∈ $\mathcal{C}$ **and** *S* ⊆ *T*
shows ∀ *F*. (¬.¬.*F*) ∈ *S* ⟶ *S* ∪ {*F*} ∈ $\mathcal{C}^+$
⟨*proof*⟩
lemma *cond-consistP4*:
assumes *consistenceP* $\mathcal{C}$ **and** *T* ∈ $\mathcal{C}$ **and** *S* ⊆ *T*

shows $\forall F.\ ((FormulaAlfa\ F) \wedge F \in S) \longrightarrow (S \cup \{Comp1\ F,\ Comp2\ F\}) \in \mathcal{C}^{+}$ ⟨*proof*⟩

lemma *cond-consistP5*:
assumes *consistenceP* $\mathcal{C}$ **and** $T \in \mathcal{C}$ **and** $S \subseteq T$
shows $(\forall F.\ ((FormulaBeta\ F) \wedge F \in S) \longrightarrow$
$(S \cup \{Comp1\ F\} \in \mathcal{C}^{+}) \vee (S \cup \{Comp2\ F\} \in \mathcal{C}^{+}))$⟨*proof*⟩
theorem *closed-consistenceP*:
assumes *hip1*: *consistenceP* $\mathcal{C}$
shows *consistenceP* $(\mathcal{C}^{+})$
⟨*proof*⟩

2 Finiteness Character Property

This theory formalises the theorem that states that subset closed propositional consistency properties can be extended to satisfy the finite character property.

The proof is by induction on the structure of propositional formulas based on the analysis of cases for the possible different types of formula in the sets of the collection of sets that hold the propositional consistency property.

definition *finite-character* :: *'a set set* ⇒ *bool* **where**
finite-character $\mathcal{C} = (\forall S.\ S \in \mathcal{C} = (\forall S'.\ finite\ S' \longrightarrow S' \subseteq S \longrightarrow S' \in \mathcal{C}))$

theorem *finite-character-closed*:
assumes *finite-character* $\mathcal{C}$
shows *subset-closed* $\mathcal{C}$
⟨*proof*⟩

definition *closure-cfinite* :: *'a set set* ⇒ *'a set set* (‹-⁻› [*1000*] *999*) **where**
$\mathcal{C}^{-} = \{S.\ \forall S'.\ S' \subseteq S \longrightarrow finite\ S' \longrightarrow S' \in \mathcal{C}\}$

lemma *finite-character-subset*:
assumes *subset-closed* $\mathcal{C}$
shows $\mathcal{C} \subseteq \mathcal{C}^{-}$
⟨*proof*⟩

lemma *finite-character*: *finite-character* $(\mathcal{C}^{-})$

⟨*proof*⟩

lemma *cond-characterP1*:
  assumes *consistenceP* $\mathcal{C}$
  and *subset-closed* $\mathcal{C}$
  and *hip*: $\forall S' \subseteq S.$ *finite* $S' \longrightarrow S' \in \mathcal{C}$
  shows $(\forall P. \neg(atom\ P \in S \wedge (\neg.atom\ P) \in S))$⟨*proof*⟩
lemma *cond-characterP2*:
  assumes *consistenceP* $\mathcal{C}$
  and *subset-closed* $\mathcal{C}$
  and *hip*: $\forall S' \subseteq S.$ *finite* $S' \longrightarrow S' \in \mathcal{C}$
  shows $FF \notin S \wedge (\neg.TT) \notin S$⟨*proof*⟩

lemma *cond-characterP3*:
  assumes *consistenceP* $\mathcal{C}$
  and *subset-closed* $\mathcal{C}$
  and *hip*: $\forall S' \subseteq S.$ *finite* $S' \longrightarrow S' \in \mathcal{C}$
  shows $\forall F. (\neg.\neg.F) \in S \longrightarrow S \cup \{F\} \in \mathcal{C}^-$⟨*proof*⟩
lemma *cond-characterP4*:
  assumes *consistenceP* $\mathcal{C}$
  and *subset-closed* $\mathcal{C}$
  and *hip*: $\forall S' \subseteq S.$ *finite* $S' \longrightarrow S' \in \mathcal{C}$
  shows $(\forall F. ((FormulaAlfa\ F) \wedge F \in S) \longrightarrow (S \cup \{Comp1\ F,\ Comp2\ F\}) \in \mathcal{C}^-)$⟨*proof*⟩
lemma *cond-characterP5*:
  assumes *consistenceP* $\mathcal{C}$
  and *subset-closed* $\mathcal{C}$
  and *hip*: $\forall S' \subseteq S.$ *finite* $S' \longrightarrow S' \in \mathcal{C}$
  shows $\forall F.\ FormulaBeta\ F \wedge F \in S \longrightarrow S \cup \{Comp1\ F\} \in \mathcal{C}^- \vee S \cup \{Comp2\ F\} \in \mathcal{C}^-$⟨*proof*⟩

theorem *cfinite-consistenceP*:
  assumes *hip1*: *consistenceP* $\mathcal{C}$ **and** *hip2*: *subset-closed* $\mathcal{C}$
  shows *consistenceP* $(\mathcal{C}^-)$
⟨*proof*⟩

definition *maximal* :: *'a set* ⇒ *'a set set* ⇒ *bool* **where**
  $maximal\ S\ \mathcal{C} = (\forall S' \in \mathcal{C}.\ S \subseteq S' \longrightarrow S = S')$

primrec *sucP* :: *'b formula set* ⇒ *'b formula set set* ⇒ (*nat* ⇒ *'b formula*) ⇒ *nat* ⇒ *'b formula set*
where
  $sucP\ S\ \mathcal{C}\ f\ 0 = S$
| $sucP\ S\ \mathcal{C}\ f\ (Suc\ n) =$
    $(if\ sucP\ S\ \mathcal{C}\ f\ n \cup \{f\ n\} \in \mathcal{C}$
     $then\ sucP\ S\ \mathcal{C}\ f\ n \cup \{f\ n\}$

else sucP S $\mathcal{C}$ *f n*)

definition *MsucP* :: *'b formula set* ⇒ *'b formula set set* ⇒ (*nat* ⇒ *'b formula*) ⇒ *'b formula set*
where
MsucP S $\mathcal{C}$ *f* = ($\bigcup$ *n. sucP S* $\mathcal{C}$ *f n*)

theorem *Max-subsetuntoP*: *S* ⊆ *MsucP S* $\mathcal{C}$ *f*⟨*proof*⟩

definition *chain* :: (*nat* ⇒ *'a set*) ⇒ *bool* **where**
chain S = (∀ *n. S n* ⊆ *S* (*Suc n*))

⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩

theorem *chain-union-closed*:
assumes *hip1*: *finite-character* $\mathcal{C}$
and *hip2*:*chain S*
and *hip3*: ∀ *n. S n* ∈ $\mathcal{C}$
shows ($\bigcup$ *n. S n*) ∈ $\mathcal{C}$⟨*proof*⟩

lemma *chain-suc*: *chain* (*sucP S* $\mathcal{C}$ *f*)
⟨*proof*⟩

theorem *MaxP-in-C*:
assumes *hip1*: *finite-character* $\mathcal{C}$ **and** *hip2*: *S* ∈ $\mathcal{C}$
shows *MsucP S* $\mathcal{C}$ *f* ∈ $\mathcal{C}$
⟨*proof*⟩

definition *enumeration* :: (*nat* ⇒*'b*) ⇒ *bool* **where**
enumeration f = (∀ *y*.∃ *n. y* = (*f n*))

lemma *enum-nat*: ∃ *g. enumeration* (*g*:: *nat* ⇒ *nat*)
⟨*proof*⟩

theorem *suc-maximalP*:
assumes *hip1*: *enumeration f* **and** *hip2*: *subset-closed* $\mathcal{C}$
shows *maximal* (*MsucP S* $\mathcal{C}$ *f*) $\mathcal{C}$
⟨*proof*⟩

corollary *ConsistentExtensionP*:
assumes *hip1*: *finite-character* $\mathcal{C}$
and *hip2*: $S \in \mathcal{C}$
and *hip3*: *enumeration f*
shows $S \subseteq MsucP\ S\ \mathcal{C}\ f$
and $MsucP\ S\ \mathcal{C}\ f \in \mathcal{C}$
and *maximal* $(MsucP\ S\ \mathcal{C}\ f)\ \mathcal{C}$
⟨*proof*⟩

3 Hintikka Theorem

The formalization of Hintikka's lemma is by induction on the structure of the formulas in a Hintikka set H by applying the technical theorem `hintikkaP_model_aux`. This theorem applies a series of lemmas to address the evaluation of all possible cases of formulas in H. Indeed, considering the Boolean evaluation IH that maps all propositional letters in H to true and all other letters to false, the most interesting cases of the inductive proof are those related to implicational formulas in H and the negation of arbitrary formulas in H. These cases are not straightforward since implicational and negation formulas are not considered in the definition of Hintikka sets. For an implicational formula, say $F_1 \longrightarrow F_2$, it is necessary to prove that if it belongs to H, its evaluation by IH is true. Also, whenever $\neg(F_1 \longrightarrow F_2)$ belongs to H its evaluation is false. The proof is obtained by relating such formulas, respectively, with β and α formulas (case P6). The second interesting case is the one related to arbitrary negations. In this case, it is proved that if $\neg F$ belongs to H, its evaluation by IH is true, and in the case that $\neg\neg F$ belongs to H, its evaluation by IH is also true (Case P7).

definition *hintikkaP* :: *'b formula set* ⇒ *bool* **where**
hintikkaP H = ((∀ *P*. ¬ (*atom P* ∈ *H* ∧ (¬.*atom P*) ∈ *H*)) ∧
FF ∉ *H* ∧ (¬.*TT*) ∉ *H* ∧
(∀ *F*. (¬.¬.*F*) ∈ *H* ⟶ *F* ∈ *H*) ∧
(∀ *F*. ((*FormulaAlfa F*) ∧ *F* ∈ *H*) ⟶
((*Comp1 F*) ∈ *H* ∧ (*Comp2 F*) ∈ *H*)) ∧
(∀ *F*. ((*FormulaBeta F*) ∧ *F* ∈ *H*) ⟶
((*Comp1 F*) ∈ *H* ∨ (*Comp2 F*) ∈ *H*)))

fun *IH* :: *'b formula set* ⇒ *'b* ⇒ *v-truth* **where**
IH H P = (*if atom P* ∈ *H then Ttrue else Ffalse*)

⟨*proof*⟩
lemma *case-P1*:
assumes *hip1*: *hintikkaP H* **and**

hip2: ∀ *G*. (*G*, *FF*) ∈ *measure f-size* ⟶
(*G* ∈ *H* ⟶ *t-v-evaluation* (*IH H*) *G* = *Ttrue*) ∧ ((¬.*G*) ∈ *H* ⟶ *t-v-evaluation* (*IH H*) (¬.*G*)= *Ttrue*)
shows (*FF* ∈ *H* ⟶ *t-v-evaluation* (*IH H*) *FF* = *Ttrue*) ∧ ((¬.*FF*) ∈ *H* ⟶ *t-v-evaluation* (*IH H*) (¬.*FF*)=*Ttrue*)⟨*proof*⟩

lemma *case-P2*:
assumes *hip1*: *hintikkaP H* **and**
hip2: ∀ *G*. (*G*, *TT*) ∈ *measure f-size* ⟶
(*G* ∈ *H* ⟶ *t-v-evaluation* (*IH H*) *G* = *Ttrue*) ∧ ((¬.*G*) ∈ *H* ⟶ *t-v-evaluation* (*IH H*) (¬.*G*)= *Ttrue*)
shows
(*TT* ∈ *H* ⟶ *t-v-evaluation* (*IH H*) *TT* = *Ttrue*) ∧ ((¬.*TT*) ∈ *H* ⟶ *t-v-evaluation* (*IH H*) (¬.*TT*)=*Ttrue*)⟨*proof*⟩

lemma *case-P3*:
assumes *hip1*: *hintikkaP H* **and**
hip2: ∀ *G*. (*G*, *atom P*) ∈ *measure f-size* ⟶
(*G* ∈ *H* ⟶ *t-v-evaluation* (*IH H*) *G* = *Ttrue*) ∧ ((¬.*G*) ∈ *H* ⟶ *t-v-evaluation* (*IH H*) (¬.*G*)= *Ttrue*)
shows (*atom P* ∈ *H* ⟶ *t-v-evaluation* (*IH H*) (*atom P*) = *Ttrue*) ∧
((¬.*atom P*) ∈ *H* ⟶ *t-v-evaluation* (*IH H*) (¬.*atom P*) = *Ttrue*)⟨*proof*⟩

lemma *case-P4*:
assumes *hip1*: *hintikkaP H* **and**
hip2: ∀ *G*. (*G*, *F1* ∧. *F2*) ∈ *measure f-size* ⟶
(*G* ∈ *H* ⟶ *t-v-evaluation* (*IH H*) *G* = *Ttrue*) ∧ ((¬.*G*) ∈ *H* ⟶ *t-v-evaluation* (*IH H*) (¬.*G*) = *Ttrue*)
shows ((*F1* ∧. *F2*) ∈ *H* ⟶ *t-v-evaluation* (*IH H*) (*F1* ∧. *F2*) = *Ttrue*) ∧
((¬.(*F1* ∧. *F2*)) ∈ *H* ⟶ *t-v-evaluation* (*IH H*) (¬.(*F1* ∧. *F2*)) = *Ttrue*)⟨*proof*⟩

lemma *case-P5*:
assumes *hip1*: *hintikkaP H* **and**
hip2: ∀ *G*. (*G*, *F1* ∨. *F2*) ∈ *measure f-size* ⟶
(*G* ∈ *H* ⟶ *t-v-evaluation* (*IH H*) *G* = *Ttrue*) ∧ ((¬.*G*) ∈ *H* ⟶ *t-v-evaluation* (*IH H*) (¬.*G*) = *Ttrue*)
shows ((*F1* ∨. *F2*) ∈ *H* ⟶ *t-v-evaluation* (*IH H*) (*F1* ∨. *F2*) = *Ttrue*) ∧
((¬.(*F1* ∨. *F2*)) ∈ *H* ⟶ *t-v-evaluation* (*IH H*) (¬.(*F1* ∨. *F2*)) = *Ttrue*)⟨*proof*⟩

lemma *case-P6*:
assumes *hip1*: *hintikkaP H* **and**
hip2: ∀ *G*. (*G*, *F1* →. *F2*) ∈ *measure f-size* ⟶
(*G* ∈ *H* ⟶ *t-v-evaluation* (*IH H*) *G* = *Ttrue*) ∧ ((¬.*G*) ∈ *H* ⟶ *t-v-evaluation* (*IH H*) (¬.*G*) = *Ttrue*)
shows ((*F1* →. *F2*) ∈ *H* ⟶ *t-v-evaluation* (*IH H*) (*F1* →. *F2*) = *Ttrue*) ∧
((¬.(*F1* →. *F2*)) ∈ *H* ⟶ *t-v-evaluation* (*IH H*) (¬.(*F1* →. *F2*)) = *Ttrue*)⟨*proof*⟩

lemma *case-P7*:
assumes *hip1*: *hintikkaP H* **and**
hip2: ∀ *G*. (*G*, (¬.*form*)) ∈ *measure f-size* ⟶

$(G \in H \longrightarrow$ *t-v-evaluation* $(IH\ H)\ G = Ttrue) \wedge ((\neg.G) \in H \longrightarrow$ *t-v-evaluation* $(IH\ H)\ (\neg.G) = Ttrue)$
shows $((\neg.form) \in H \longrightarrow$ *t-v-evaluation* $(IH\ H)\ (\neg.form) = Ttrue) \wedge$
$((\neg.(\neg.form)) \in H \longrightarrow$ *t-v-evaluation* $(IH\ H)\ (\neg.(\neg.form)) = Ttrue)\langle proof\rangle$
theorem *hintikkaP-model-aux*:
assumes *hip*: *hintikkaP H*
shows $(F \in H \longrightarrow$ *t-v-evaluation* $(IH\ H)\ F = Ttrue) \wedge$
$((\neg.F) \in H \longrightarrow$ *t-v-evaluation* $(IH\ H)\ (\neg.F) = Ttrue)$
$\langle proof\rangle$

corollary *ModeloHintikkaPa*:
assumes *hintikkaP H* **and** $F \in H$
shows *t-v-evaluation* $(IH\ H)\ F = Ttrue$
$\langle proof\rangle$

corollary *ModeloHintikkaP*:
assumes *hintikkaP H*
shows $(IH\ H)$ *model H*
$\langle proof\rangle$

corollary *Hintikkasatisfiable*:
assumes *hintikkaP H*
shows *satisfiable H*
$\langle proof\rangle$

4 Maximal Hintikka

This theory formalises maximality of Hintikka sets according to Smullyan's textbook [3]. Specifically, following [1] (page 55) this theory formalises the fact that if $\mathcal{C}$ is a propositional consistence property closed by subsets, and M a maximal set belonging to $\mathcal{C}$ then M is a Hintikka set.

lemma *ext-hintikkaP1*:
assumes *hip1*: *consistenceP* $\mathcal{C}$ **and** *hip2*: $M \in \mathcal{C}$
shows $\forall p.\ \neg\ (atom\ p \in M \wedge (\neg.atom\ p) \in M)\langle proof\rangle$

lemma *ext-hintikkaP2*:
assumes *hip1*: *consistenceP* $\mathcal{C}$ **and** *hip2*: $M \in \mathcal{C}$
shows $FF \notin M\langle proof\rangle$

lemma *ext-hintikkaP3*:
assumes *hip1*: *consistenceP* $\mathcal{C}$ **and** *hip2*: $M \in \mathcal{C}$
shows $(\neg.TT) \notin M\langle proof\rangle$

lemma *ext-hintikkaP4*:
assumes *hip1*: *consistenceP* $\mathcal{C}$ **and** *hip2*: *maximal M* $\mathcal{C}$ **and** *hip3*: $M \in \mathcal{C}$
shows $\forall F.\ (\neg.\neg.F) \in M \longrightarrow F \in M$⟨*proof*⟩

lemma *ext-hintikkaP5*:
assumes *hip1*: *consistenceP* $\mathcal{C}$ **and** *hip2*: *maximal M* $\mathcal{C}$ **and** *hip3*: $M \in \mathcal{C}$
shows $\forall F.$ (*FormulaAlfa F*) $\wedge\ F \in M \longrightarrow$ (*Comp1 F* $\in M\ \wedge$ *Comp2 F* $\in M$)⟨*proof*⟩

lemma *ext-hintikkaP6*:
assumes *hip1*: *consistenceP* $\mathcal{C}$ **and** *hip2*: *maximal M* $\mathcal{C}$ **and** *hip3*: $M \in \mathcal{C}$
shows $\forall F.$ (*FormulaBeta F*) $\wedge\ F \in M \longrightarrow$ *Comp1 F* $\in M\ \vee$ *Comp2 F* $\in M$⟨*proof*⟩

theorem *MaximalHintikkaP*:
assumes *hip1*: *consistenceP* $\mathcal{C}$ **and** *hip2*: *maximal M* $\mathcal{C}$ **and** *hip3*: $M \in \mathcal{C}$
shows *hintikkaP M*
⟨*proof*⟩

lemma *enumeration*: *enumeration* $f = (\exists g.\ \forall y.\ f(g\ y) = y)$
⟨*proof*⟩

datatype *tree-b* = *Leaf nat* | *Tree tree-b tree-b*

```
primrec diag :: nat ⇒ (nat × nat) where
  diag 0 = (0, 0)
| diag (Suc n) =
     (let (x, y) = diag n
      in case y of
          0 ⇒ (0, Suc x)
        | Suc y ⇒ (Suc x, y))
```

```
function undiag :: nat × nat ⇒ nat where
  undiag (0, 0) = 0
| undiag (0, Suc y) = Suc (undiag (y, 0))
| undiag (Suc x, y) = Suc (undiag (x, Suc y))
⟨proof⟩
```

termination
⟨*proof*⟩

lemma *diag-undiag* [*simp*]: *diag* (*undiag* (*x*, *y*)) = (*x*, *y*)
⟨*proof*⟩

lemma *enumeration-natxnat*: *enumeration* (*diag*::*nat* ⇒ (*nat* × *nat*))

⟨*proof*⟩

function *diag-tree-b* :: *nat* ⇒ *tree-b* **where**
diag-tree-b n = (*case fst* (*diag n*) *of*
  0 ⇒ *Leaf* (*snd* (*diag n*))
  | *Suc z* ⇒ *Tree* (*diag-tree-b z*) (*diag-tree-b* (*snd* (*diag n*))))
⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩

primrec *undiag-tree-b* :: *tree-b* ⇒ *nat* **where**
undiag-tree-b (*Leaf n*) = *undiag* (*0*, *n*)
| *undiag-tree-b* (*Tree t1 t2*) =
undiag (*Suc* (*undiag-tree-b t1*), *undiag-tree-b t2*)

lemma *diag-undiag-tree-b* [*simp*]: *diag-tree-b* (*undiag-tree-b t*) = *t*
⟨*proof*⟩

lemma *enumeration-tree-b*: *enumeration* (*diag-tree-b* :: *nat* ⇒ *tree-b*)
⟨*proof*⟩

fun *formulaP-from-tree-b* :: (*nat* ⇒ *'b*) ⇒ *tree-b* ⇒ *'b formula* **where**
formulaP-from-tree-b g (*Leaf 0*) = *FF*
| *formulaP-from-tree-b g* (*Leaf* (*Suc 0*)) = *TT*
| *formulaP-from-tree-b g* (*Leaf* (*Suc* (*Suc n*))) = (*atom* (*g n*))
| *formulaP-from-tree-b g* (*Tree* (*Leaf* (*Suc 0*)) (*Tree T1 T2*)) =
((*formulaP-from-tree-b g T1*) ∧. (*formulaP-from-tree-b g T2*))
| *formulaP-from-tree-b g* (*Tree* (*Leaf* (*Suc* (*Suc 0*))) (*Tree T1 T2*)) =
((*formulaP-from-tree-b g T1*) ∨. (*formulaP-from-tree-b g T2*))
| *formulaP-from-tree-b g* (*Tree* (*Leaf* (*Suc* (*Suc* (*Suc 0*)))) (*Tree T1 T2*)) =
((*formulaP-from-tree-b g T1*) →. (*formulaP-from-tree-b g T2*))
| *formulaP-from-tree-b g* (*Tree* (*Leaf* (*Suc* (*Suc* (*Suc* (*Suc 0*))))) *T*) =
(¬. (*formulaP-from-tree-b g T*))⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩

primrec *tree-b-from-formulaP* :: (*'b* ⇒ *nat*) ⇒ *'b formula* ⇒ *tree-b* **where**
tree-b-from-formulaP g FF = *Leaf 0*
| *tree-b-from-formulaP g TT* = *Leaf* (*Suc 0*)
| *tree-b-from-formulaP g* (*atom P*) = *Leaf* (*Suc* (*Suc* (*g P*)))
| *tree-b-from-formulaP g* (*F* ∧. *G*) = *Tree* (*Leaf* (*Suc 0*))
(*Tree* (*tree-b-from-formulaP g F*) (*tree-b-from-formulaP g G*))
| *tree-b-from-formulaP g* (*F* ∨. *G*) = *Tree* (*Leaf* (*Suc* (*Suc 0*)))
(*Tree* (*tree-b-from-formulaP g F*) (*tree-b-from-formulaP g G*))
| *tree-b-from-formulaP g* (*F* →. *G*) = *Tree* (*Leaf* (*Suc* (*Suc* (*Suc 0*))))
(*Tree* (*tree-b-from-formulaP g F*) (*tree-b-from-formulaP g G*))

| *tree-b-from-formulaP* g ($\neg$. F) = *Tree* (*Leaf* (*Suc* (*Suc* (*Suc* (*Suc* 0)))))
  (*tree-b-from-formulaP* g F)

definition ΔP :: ($nat \Rightarrow$ $'b$) $\Rightarrow$ $nat \Rightarrow$ $'b$ *formula* **where**
  ΔP g n = *formulaP-from-tree-b* g (*diag-tree-b* n)

definition $\Delta P'$ :: ($'b \Rightarrow nat$) $\Rightarrow$ $'b$ *formula* $\Rightarrow$ *nat* **where**
  $\Delta P'$ g' F = *undiag-tree-b* (*tree-b-from-formulaP* g' F)

theorem *enumerationformulasP*[*simp*]:
  assumes $\forall x.\ g(g'\ x) = x$
  shows $\Delta P\ g\ (\Delta P'\ g'\ F) = F$
⟨*proof*⟩

corollary *EnumerationFormulasP*:
  assumes $\forall P.\ \exists\ n.\ P = g\ n$
  shows $\forall F.\ \exists n.\ F = \Delta P\ g\ n$
⟨*proof*⟩

corollary *EnumerationFormulasP1*:
  assumes *enumeration* (g:: *nat* $\Rightarrow$ $'b$)
  shows *enumeration* ((ΔP g):: *nat* $\Rightarrow$ $'b$ *formula*)
⟨*proof*⟩

corollary *EnumeracionFormulasNat*:
  shows $\exists\ f$. *enumeration* (f:: *nat* $\Rightarrow$ *nat formula*)
  ⟨*proof*⟩

5 Model Existence Theorem

This theory formalises the Model Existence Theorem according to Smullyan's textbook [3] as presented by Fitting in [1].

theorem *ExtensionCharacterFinitoP*:
  shows $\mathcal{C} \subseteq \mathcal{C}^{+-}$
  and *finite-character* ($\mathcal{C}^{+-}$)
  and *consistenceP* $\mathcal{C} \longrightarrow$ *consistenceP* ($\mathcal{C}^{+-}$)
⟨*proof*⟩

lemma *ExtensionConsistenteP1*:
  assumes h: *enumeration* g

and *h1*: *consistenceP* $\mathcal{C}$
and *h2*: $S \in \mathcal{C}$
shows $S \subseteq$ *MsucP* S $(\mathcal{C}^{+-})$ g
and *maximal* (*MsucP* S $(\mathcal{C}^{+-})$ g) $(\mathcal{C}^{+-})$
and *MsucP* S $(\mathcal{C}^{+-})$ $g \in \mathcal{C}^{+-}$

⟨*proof*⟩

theorem *HintikkaP*:
assumes *h0*:*enumeration* g **and** *h1*: *consistenceP* $\mathcal{C}$ **and** *h2*: $S \in \mathcal{C}$
shows *hintikkaP* (*MsucP* S $(\mathcal{C}^{+-})$ g)
⟨*proof*⟩

theorem *ExistenceModelP*:
assumes *h0*: *enumeration* g
and *h1*: *consistenceP* $\mathcal{C}$
and *h2*: $S \in \mathcal{C}$
and *h3*: $F \in S$
shows *t-v-evaluation* (*IH* (*MsucP* S $(\mathcal{C}^{+-})$ g)) F = *Ttrue*
⟨*proof*⟩

theorem *Theo-ExistenceModels*:
assumes *h1*: $\exists g$. *enumeration* (g:: *nat* $\Rightarrow$ *'b formula*)
and *h2*: *consistenceP* $\mathcal{C}$
and *h3*: (S:: *'b formula set*) $\in \mathcal{C}$
shows *satisfiable* S
⟨*proof*⟩

corollary *Satisfiable-SetP1*:
assumes *h0*: $\exists g$. *enumeration* (g:: *nat* $\Rightarrow$ *'b*)
and *h1*: *consistenceP* $\mathcal{C}$
and *h2*: (S:: *'b formula set*) $\in \mathcal{C}$
shows *satisfiable* S
⟨*proof*⟩

corollary *Satisfiable-SetP2*:
assumes *consistenceP* $\mathcal{C}$ **and** (S:: *nat formula set*) $\in \mathcal{C}$
shows *satisfiable* S
⟨*proof*⟩

theory *PropCompactness*

imports *Main*
HOL−Library.Countable-Set
ModelExistence

begin

6 Compactness Theorem for Propositional Logic

This theory formalises the compactness theorem based on the existence model theorem. The formalisation, initially published as [2] in Spanish, was adapted to extend several combinatorial theorems over finite structures to the infinite case (e.g., see Serrano, Ayala-Rincón, and de Lima formalizations of Hall's Theorem for infinite families of sets and infinite graphs [4, 5].)

The formalization shows first Hintikka's Lemma: Hintikka sets of propositional formulas are satisfiable. Such a set is defined as a set of propositional formulas that does neither include both A and $\neg A$ for a propositional letter nor $\bot$, or $\neg\top$. Additionally, if it includes $\neg\neg F$, F is included; if it includes a conjunctive formula, which is an α formula, then the two components of the conjunction are included; and finally, if it includes a disjunction, which is a β formula, at least one of the components of the disjunction is included.

The satisfiability of any Hintikka set is proved by assuming a valuation that maps all propositional letters in the set to true and all other propositional letters to false. The second step consists in proving that families of sets of propositional formulas, which hold the so-called "propositional consistency property," consist of satisfiable sets. The last is indeed the model existence theorem. The model existence theorem compiles the essence of completeness: a family of sets of propositional formulas that holds the propositional consistency property can be extended, preserving this property to a set collection that is closed for subsets and satisfies the finite character property. The finite character property states that a set belongs to the family if and only if each of its finite subsets belongs to the family. With the model existence theorem in hands, the compactness theorem is obtained easily: given a set of propositional formulas S such that all its finite subsets are satisfiable, one considers the family $\mathcal{C}$ of subsets in S such that all their finite subsets are satisfiable. S belongs to the family $\mathcal{C}$ and the latter holds the propositional consistence property.

The auxiliary lemma of Consistence Compactness is required to apply the Model Existence Theorem to obtain the compactness theorem. This lemma states the general fact that the collection $\mathcal{C}$ of all sets of propositional formulas such that all their subsets are satisfiable is a propositional consistency property.

lemma *UnsatisfiableAtom*:

shows ¬ (*satisfiable* {*F*, ¬.*F*})
⟨*proof*⟩

lemma *consistenceP-Prop1*:
assumes ∀ (*A*::′*b formula set*). (*A*⊆ *W* ∧ *finite A*) ⟶ *satisfiable A*
shows (∀ *P*. ¬ (*Atom P* ∈ *W* ∧ (¬. *Atom P*) ∈ *W*))
⟨*proof*⟩

lemma *UnsatisfiableFF*:
shows ¬ (*satisfiable* {*FF*})
⟨*proof*⟩

lemma *consistenceP-Prop2*:
assumes ∀ (*A*::′*b formula set*). (*A*⊆ *W* ∧ *finite A*) ⟶ *satisfiable A*
shows *FF* ∉ *W*
⟨*proof*⟩

lemma *UnsatisfiableFFa*:
shows ¬ (*satisfiable* {¬.*TT*})
⟨*proof*⟩

lemma *consistenceP-Prop3*:
assumes ∀ (*A*::′*b formula set*). (*A*⊆ *W* ∧ *finite A*) ⟶ *satisfiable A*
shows ¬.*TT* ∉ *W*
⟨*proof*⟩

lemma *Subset-Sat*:
assumes *hip1*: *satisfiable S* **and** *hip2*: *S*′⊆ *S*
shows *satisfiable S*′
⟨*proof*⟩

lemma *satisfiableUnion1*:
assumes *satisfiable* (*A* ∪ {¬.¬.*F*})
shows *satisfiable* (*A* ∪ {*F*})
⟨*proof*⟩

lemma *consistenceP-Prop4*:
assumes *hip1*: ∀ (*A*::′*b formula set*). (*A*⊆ *W* ∧ *finite A*) ⟶ *satisfiable A*
and *hip2*: ¬.¬.*F* ∈ *W*
shows ∀ (*A*::′*b formula set*). (*A*⊆ *W* ∪ {*F*} ∧ *finite A*) ⟶ *satisfiable A*
⟨*proof*⟩

lemma *satisfiableUnion2*:
assumes *hip1*: *FormulaAlfa F* **and** *hip2*: *satisfiable* (*A* ∪ {*F*})
shows *satisfiable* (*A* ∪ {*Comp1 F*,*Comp2 F*})
⟨*proof*⟩

lemma *consistenceP-Prop5*:
assumes *hip0*: *FormulaAlfa F*
and *hip1*: ∀ (*A*::′*b formula set*). (*A*⊆ *W* ∧ *finite A*) ⟶ *satisfiable A*
and *hip2*: *F* ∈ *W*
shows ∀ (*A*::′*b formula set*). (*A*⊆ *W* ∪ {*Comp1 F*, *Comp2 F*} ∧ *finite A*) ⟶
satisfiable A
⟨*proof*⟩

lemma *satisfiableUnion3*:
assumes *hip1*: *FormulaBeta F* **and** *hip2*: *satisfiable* (*A* ∪ {*F*})
shows *satisfiable* (*A* ∪ {*Comp1 F*}) ∨ *satisfiable* (*A* ∪ {*Comp2 F*})
⟨*proof*⟩

lemma *consistenceP-Prop6*:
assumes *hip0*: *FormulaBeta F*
and *hip1*: ∀ (*A*::′*b formula set*). (*A*⊆ *W* ∧ *finite A*) ⟶ *satisfiable A*
and *hip2*: *F* ∈ *W*
shows (∀ (*A*::′*b formula set*). (*A*⊆ *W* ∪ {*Comp1 F*} ∧ *finite A*) ⟶
satisfiable A) ∨
(∀ (*A*::′*b formula set*). (*A*⊆ *W* ∪ {*Comp2 F*} ∧ *finite A*) ⟶
satisfiable A)
⟨*proof*⟩

lemma *ConsistenceCompactness*:
shows *consistenceP*{*W*::′*b formula set*. ∀ *A*. (*A*⊆ *W* ∧ *finite A*) ⟶
satisfiable A}
⟨*proof*⟩

lemma *countable-enumeration-formula*:
shows ∃ *f*. *enumeration* (*f*:: *nat* ⇒′*a*::*countable formula*)
⟨*proof*⟩

theorem *Compactness-Theorem*:
assumes ∀ *A*. (*A* ⊆ (*S*:: ′*a*::*countable formula set*) ∧ *finite A*) ⟶ *satisfiable A*
shows *satisfiable S*
⟨*proof*⟩

end

theory *Hall-Theorem*
imports
PropCompactness
Marriage.Marriage
begin

7 Hall Theorem for countable (infinite) families of sets

Hall's Theorem for countable families of sets is proved as a consequence of compactness theorem for propositional calculus ([4]). The theory imports Marriage theory from the AFP, which proves marriage theorem for the finite case. The proof also uses an updated version of Serrano's formalization of the compactness theorem for propositional logic.

definition *system-representatives* :: $('a \Rightarrow 'b\ set) \Rightarrow 'a\ set \Rightarrow ('a \Rightarrow 'b) \Rightarrow bool$
where
system-representatives $S\ I\ R \equiv (\forall i{\in}I.\ (R\ i) \in (S\ i)) \wedge (inj\text{-}on\ R\ I)$

definition *set-to-list* :: $'a\ set \Rightarrow 'a\ list$
where *set-to-list* $s = (SOME\ l.\ set\ l = s)$

lemma *set-set-to-list*:
finite $s \Longrightarrow set\ (set\text{-}to\text{-}list\ s) = s$
⟨*proof*⟩

lemma *list-to-set*:
assumes *finite* $(S\ i)$
shows $set\ (set\text{-}to\text{-}list\ (S\ i)) = (S\ i)$
⟨*proof*⟩

primrec *disjunction-atomic* :: $'b\ list \Rightarrow 'a \Rightarrow ('a \times 'b) formula$ **where**
disjunction-atomic $[]\ i = FF$
$|$ *disjunction-atomic* $(x\#D)\ i = (atom\ (i,\ x))\ \vee.\ (disjunction\text{-}atomic\ D\ i)$

lemma *t-v-evaluation-disjunctions1*:
assumes *t-v-evaluation* $I\ (disjunction\text{-}atomic\ (a\ \#\ l)\ i) = Ttrue$
shows *t-v-evaluation* $I\ (atom\ (i,a)) = Ttrue \vee t\text{-}v\text{-}evaluation\ I\ (disjunction\text{-}atomic\ l\ i) = Ttrue$
⟨*proof*⟩

lemma *t-v-evaluation-atom*:
assumes *t-v-evaluation* $I\ (disjunction\text{-}atomic\ l\ i) = Ttrue$
shows $\exists x.\ x \in set\ l \wedge (t\text{-}v\text{-}evaluation\ I\ (atom\ (i,x)) = Ttrue)$
⟨*proof*⟩

definition $\mathcal{F}$ :: $('a \Rightarrow 'b\ set) \Rightarrow 'a\ set \Rightarrow (('a \times 'b) formula)\ set$ **where**
$\mathcal{F}\ S\ I \equiv (\bigcup i{\in}I.\ \{\ disjunction\text{-}atomic\ (set\text{-}to\text{-}list\ (S\ i))\ i\ \})$

definition $\mathcal{G}$ :: $('a \Rightarrow 'b\ set) \Rightarrow 'a\ set \Rightarrow ('a \times 'b) formula\ set$ **where**
$\mathcal{G}\ S\ I \equiv \{\neg.(atom\ (i,x)\ \wedge.\ atom(i,y))$
$|x\ y\ i\ .\ x{\in}(S\ i) \wedge y{\in}(S\ i) \wedge\ x{\neq}y \wedge i{\in}I\}$

definition $\mathcal{H}$ :: $('a \Rightarrow 'b\ set) \Rightarrow 'a\ set \Rightarrow ('a \times 'b) formula\ set$ **where**
$\mathcal{H}\ S\ I \equiv \{\neg.(atom\ (i,x)\ \wedge.\ atom(j,x))$

$| \; x \; i \; j. \; x \in (S \; i) \cap (S \; j) \wedge (i{\in}I \wedge j{\in}I \wedge i{\neq}j)\}$

definition $\mathcal{T}$ :: $('a \Rightarrow 'b \; set) \Rightarrow 'a \; set \Rightarrow ('a \times 'b)formula \; set$ **where**
$\mathcal{T} \; S \; I \equiv (\mathcal{F} \; S \; I) \cup (\mathcal{G} \; S \; I) \cup (\mathcal{H} \; S \; I)$

primrec *indices-formula* :: $('a \times 'b)formula \Rightarrow 'a \; set$ **where**
indices-formula $FF = \{\}$
| *indices-formula* $TT = \{\}$
| *indices-formula* $(atom \; P) = \{fst \; P\}$
| *indices-formula* $(\neg. \; F) =$ *indices-formula* F
| *indices-formula* $(F \wedge. \; G) =$ *indices-formula* $F \cup$ *indices-formula* G
| *indices-formula* $(F \vee. \; G) =$ *indices-formula* $F \cup$ *indices-formula* G
| *indices-formula* $(F \rightarrow. \; G) =$ *indices-formula* $F \cup$ *indices-formula* G

definition *indices-set-formulas* :: $('a \times 'b)formula \; set \Rightarrow 'a \; set$ **where**
indices-set-formulas $S = (\bigcup F \in S.$ *indices-formula* $F)$

lemma *finite-indices-formulas*:
shows *finite* (*indices-formula* F)
⟨*proof*⟩

lemma *finite-set-indices*:
assumes *finite* S
shows *finite* (*indices-set-formulas* S)
⟨*proof*⟩

lemma *indices-disjunction*:
assumes $F =$ *disjunction-atomic* $L \; i$ **and** $L \neq []$
shows *indices-formula* $F = \{i\}$
⟨*proof*⟩

lemma *nonempty-set-list*:
assumes $\forall \, i{\in}I. \; (S \; i){\neq}\{\}$ **and** $\forall \, i{\in}I.$ *finite* $(S \; i)$
shows $\forall \, i{\in}I.$ *set-to-list* $(S \; i){\neq}[]$
⟨*proof*⟩

lemma *at-least-subset-indices*:
assumes $\forall \, i{\in}I. \; (S \; i){\neq}\{\}$ **and** $\forall \, i{\in}I.$ *finite* $(S \; i)$
shows *indices-set-formulas* $(\mathcal{F} \; S \; I) \subseteq I$
⟨*proof*⟩

lemma *at-most-subset-indices*:
shows *indices-set-formulas* $(\mathcal{G} \; S \; I) \subseteq I$
⟨*proof*⟩

lemma *different-subset-indices*:
shows *indices-set-formulas* $(\mathcal{H} \; S \; I) \subseteq I$
⟨*proof*⟩

lemma *indices-union-sets*:
shows *indices-set-formulas*$(A \cup B) = ($*indices-set-formulas* $A) \cup ($*indices-set-formulas* $B)$
⟨*proof*⟩

lemma *at-least-subset-subset-indices1*:
assumes $F \in (\mathcal{F}\ S\ I)$
shows (*indices-formula* F) $\subseteq$ (*indices-set-formulas* $(\mathcal{F}\ S\ I)$)
⟨*proof*⟩

lemma *at-most-subset-subset-indices1*:
assumes $F \in (\mathcal{G}\ S\ I)$
shows (*indices-formula* F) $\subseteq$ (*indices-set-formulas* $(\mathcal{G}\ S\ I)$)
⟨*proof*⟩

lemma *different-subset-indices1*:
assumes $F \in (\mathcal{H}\ S\ I)$
shows (*indices-formula* F) $\subseteq$ (*indices-set-formulas* $(\mathcal{H}\ S\ I)$)
⟨*proof*⟩

lemma *all-subset-indices*:
assumes $\forall i \in I.(S\ i) \neq \{\}$ **and** $\forall i \in I.$ *finite*$(S\ i)$
shows *indices-set-formulas* $(\mathcal{T}\ S\ I) \subseteq I$
⟨*proof*⟩

lemma *inclusion-indices*:
assumes $S \subseteq H$
shows *indices-set-formulas* $S \subseteq$ *indices-set-formulas* H
⟨*proof*⟩

lemma *indices-subset-formulas*:
assumes $\forall i \in I.(S\ i) \neq \{\}$ **and** $\forall i \in I.$ *finite*$(S\ i)$ **and** $A \subseteq (\mathcal{T}\ S\ I)$
shows (*indices-set-formulas* A) $\subseteq I$
⟨*proof*⟩

lemma *To-subset-all-its-indices*:
assumes $\forall i \in I.\ (S\ i) \neq \{\}$ **and** $\forall i \in I.$ *finite* $(S\ i)$ **and** *To* $\subseteq (\mathcal{T}\ S\ I)$
shows *To* $\subseteq (\mathcal{T}\ S$ (*indices-set-formulas* *To*))
⟨*proof*⟩

lemma *all-nonempty-sets*:
assumes $\forall i \in I.\ (S\ i) \neq \{\}$ **and** $\forall i \in I.$ *finite* $(S\ i)$ **and** $A \subseteq (\mathcal{T}\ S\ I)$
shows $\forall i \in$(*indices-set-formulas* A). $(S\ i) \neq \{\}$
⟨*proof*⟩

lemma *all-finite-sets*:
assumes $\forall i \in I.\ (S\ i) \neq \{\}$ **and** $\forall i \in I.$ *finite* $(S\ i)$ **and** $A \subseteq (\mathcal{T}\ S\ I)$
shows $\forall i \in$(*indices-set-formulas* A). *finite* $(S\ i)$
⟨*proof*⟩

lemma *all-nonempty-sets1*:
  assumes $\forall J \subseteq I.\ finite\ J \longrightarrow\ card\ J \le card\ (\bigcup\ (S\ `\ J))$
  shows $\forall i \in I.\ (S\ i) \neq \{\}$ ⟨*proof*⟩

lemma *system-distinct-representatives-finite*:
  assumes
  $\forall i \in I.\ (S\ i) \neq \{\}$ **and** $\forall i \in I.\ finite\ (S\ i)$ **and** $To \subseteq (\mathcal{T}\ S\ I)$ **and** $finite\ To$
  and $\forall J \subseteq (indices\text{-}set\text{-}formulas\ To).\ card\ J \le card\ (\bigcup\ (S\ `\ J))$
 shows $\exists R.\ system\text{-}representatives\ S\ (indices\text{-}set\text{-}formulas\ To)\ R$
⟨*proof*⟩

fun *Hall-interpretation* :: $('a \Rightarrow 'b\ set) \Rightarrow 'a\ set \Rightarrow ('a \Rightarrow 'b) \Rightarrow (('a \times 'b) \Rightarrow v\text{-}truth)$ **where**
$Hall\text{-}interpretation\ A\ \mathcal{I}\ R = (\lambda(i,x).(if\ \ i \in \mathcal{I} \wedge x \in (A\ i) \wedge (R\ i) = x\ \ then\ Ttrue\ else\ Ffalse))$

lemma *t-v-evaluation-index*:
  assumes $t\text{-}v\text{-}evaluation\ (Hall\text{-}interpretation\ S\ I\ R)\ (atom\ (i,x)) = Ttrue$
  shows $(R\ i) = x$
⟨*proof*⟩

lemma *distinct-elements-distinct-indices*:
  assumes $F = \neg.(atom\ (i,x) \wedge.\ atom(i,y))$ **and** $x \neq y$
  shows $t\text{-}v\text{-}evaluation\ (Hall\text{-}interpretation\ S\ I\ R)\ F = Ttrue$
⟨*proof*⟩

lemma *same-element-same-index*:
  assumes
  $F = \neg.(atom\ (i,x) \wedge.\ atom(j,x))$ **and** $i \in I \wedge j \in I$ **and** $i \neq j$ **and** $inj\text{-}on\ R\ I$
  shows $t\text{-}v\text{-}evaluation\ (Hall\text{-}interpretation\ S\ I\ R)\ F = Ttrue$
⟨*proof*⟩

lemma *disjunctor-Ttrue-in-atomic-disjunctions*:
  assumes $x \in set\ l$ **and** $t\text{-}v\text{-}evaluation\ I\ (atom\ (i,x)) = Ttrue$
  shows $t\text{-}v\text{-}evaluation\ I\ (disjunction\text{-}atomic\ l\ i) = Ttrue$
⟨*proof*⟩

lemma *t-v-evaluation-disjunctions*:
  assumes $finite\ (S\ i)$
  and $x \in (S\ i)\ \wedge\ t\text{-}v\text{-}evaluation\ I\ (atom\ (i,x)) = Ttrue$
  and $F = disjunction\text{-}atomic\ (set\text{-}to\text{-}list\ (S\ i))\ i$
  shows $t\text{-}v\text{-}evaluation\ I\ F = Ttrue$
⟨*proof*⟩

theorem *SDR-satisfiable*:
  assumes $\forall i \in \mathcal{I}.\ (A\ i) \neq \{\}$ **and** $\forall i \in \mathcal{I}.\ finite\ (A\ i)$ **and** $X \subseteq (\mathcal{T}\ A\ \mathcal{I})$
  and $system\text{-}representatives\ A\ \mathcal{I}\ R$
shows $satisfiable\ X$

⟨*proof*⟩

lemma *finite-is-satisfiable*:
assumes
∀ *i*∈*I*. (*S i*)≠{} **and** ∀ *i*∈*I*. *finite* (*S i*) **and** *To* ⊆ (𝒯 *S I*) **and** *finite To*
and ∀ *J*⊆(*indices-set-formulas To*). *card J* ≤ *card* (⋃ (*S* ' *J*))
shows *satisfiable To*
⟨*proof*⟩

lemma *diag-nat*:
shows ∀ *y z*.∃ *x*. (*y*,*z*) = *diag x*
⟨*proof*⟩

lemma *EnumFormulasHall*:
assumes ∃ *g*. *enumeration* (*g*:: *nat* ⇒'*a*) **and** ∃ *h*. *enumeration* (*h*:: *nat* ⇒'*b*)
shows ∃ *f*. *enumeration* (*f*:: *nat* ⇒('*a* ×'*b*)*formula*)
⟨*proof*⟩

theorem *all-formulas-satisfiable*:
fixes *S* :: ('*a*::*countable* ⇒ '*b*::*countable set*) **and** *I* :: '*a set*
assumes ∀ *i*∈(*I*::'*a set*). *finite* (*S i*) **and** ∀ *J*⊆*I*. *finite J* ⟶ *card J* ≤ *card* (⋃ (*S* ' *J*))
shows *satisfiable* (𝒯 *S I*)
⟨*proof*⟩

fun *SDR* :: (('*a* × '*b*) ⇒ *v-truth*) ⇒ ('*a* ⇒ '*b set*) ⇒ '*a set* ⇒ ('*a* ⇒'*b*)
where
SDR M S I = (λ*i*. (*THE x*. (*t-v-evaluation M* (*atom* (*i*,*x*)) = *Ttrue*) ∧ *x*∈(*S i*)))

lemma *existence-representants*:
assumes *i* ∈ *I* **and** *M model* (ℱ *S I*) **and** *finite*(*S i*)
shows ∃ *x*. (*t-v-evaluation M* (*atom* (*i*,*x*)) = *Ttrue*) ∧ *x* ∈ (*S i*)
⟨*proof*⟩

lemma *unicity-representants*:
shows ∀ *y*.(*x*∈(*S i*) ∧ *y*∈(*S i*) ∧ *x*≠*y* ∧ *i*∈*I*) ⟶
(¬.(*atom* (*i*,*x*) ∧. *atom*(*i*,*y*))∈ (𝒢 *S I*))
⟨*proof*⟩

lemma *unicity-selection-representants*:
assumes *i* ∈ *I* **and** *M model* (𝒢 *S I*)
shows ∀ *y*.(*x*∈(*S i*) ∧ *y*∈(*S i*) ∧ *x*≠*y* ∧ *i*∈*I*) ⟶
(*t-v-evaluation M* (¬.(*atom* (*i*,*x*) ∧. *atom*(*i*,*y*))) = *Ttrue*)
⟨*proof*⟩

lemma *uniqueness-satisfaction*:
assumes *t-v-evaluation M* (*atom* (*i*,*x*)) = *Ttrue* ∧ *x*∈(*S i*) **and**
∀ *y*. *y* ∈ (*S i*) ∧ *x*≠*y* ⟶ *t-v-evaluation M* (*atom* (*i*, *y*)) = *Ffalse*
shows ∀ *z*. *t-v-evaluation M* (*atom* (*i*, *z*)) = *Ttrue* ∧ *z*∈(*S i*) ⟶ *z* = *x*

⟨*proof*⟩

lemma *uniqueness-satisfaction-in-Si*:
assumes *t-v-evaluation M* (*atom* (*i*,*x*)) = *Ttrue* ∧ *x*∈(*S i*) **and**
∀ *y*. *y* ∈ (*S i*) ∧ *x*≠*y* ⟶ (*t-v-evaluation M* (¬.(*atom* (*i*,*x*) ∧. *atom*(*i*,*y*))) = *Ttrue*)
shows ∀ *y*. *y* ∈ (*S i*) ∧ *x*≠*y* ⟶ *t-v-evaluation M* (*atom* (*i*, *y*)) = *Ffalse*
⟨*proof*⟩

lemma *uniqueness-aux1*:
assumes *t-v-evaluation M* (*atom* (*i*,*x*)) = *Ttrue* ∧ *x*∈(*S i*)
and ∀ *y*. *y* ∈ (*S i*) ∧ *x*≠*y* ⟶ (*t-v-evaluation M* (¬.(*atom* (*i*,*x*) ∧. *atom*(*i*,*y*))) = *Ttrue*)
shows ∀ *z*. *t-v-evaluation M* (*atom* (*i*, *z*)) = *Ttrue* ∧ *z*∈(*S i*) ⟶ *z* = *x*
⟨*proof*⟩

lemma *uniqueness-aux2*:
assumes *t-v-evaluation M* (*atom* (*i*,*x*)) = *Ttrue* ∧ *x*∈(*S i*) **and**
(⋀*z*.(*t-v-evaluation M* (*atom* (*i*, *z*)) = *Ttrue* ∧ *z*∈(*S i*)) ⟹ *z* = *x*)
shows (*THE a*. (*t-v-evaluation M* (*atom* (*i*,*a*)) = *Ttrue*) ∧ *a*∈(*S i*)) = *x*
⟨*proof*⟩

lemma *uniqueness-aux*:
assumes *t-v-evaluation M* (*atom* (*i*,*x*)) = *Ttrue* ∧ *x*∈(*S i*) **and**
∀ *y*. *y* ∈ (*S i*) ∧ *x*≠*y* ⟶ (*t-v-evaluation M* (¬.(*atom* (*i*,*x*) ∧. *atom*(*i*,*y*))) = *Ttrue*)
shows (*THE a*. (*t-v-evaluation M* (*atom* (*i*,*a*)) = *Ttrue*) ∧ *a*∈(*S i*)) = *x*
⟨*proof*⟩

lemma *function-SDR*:
assumes *i* ∈ *I* **and** *M model* (𝓕 *S I*) **and** *M model* (𝓖 *S I*) **and** *finite*(*S i*)
shows ∃!*x*. (*t-v-evaluation M* (*atom* (*i*,*x*)) = *Ttrue*) ∧ *x* ∈ (*S i*) ∧ (*SDR M S I i*) = *x*
⟨*proof*⟩

lemma *aux-for-𝓗-formulas*:
assumes
(*t-v-evaluation M* (*atom* (*i*,*a*)) = *Ttrue*) ∧ *a* ∈ (*S i*)
and (*t-v-evaluation M* (*atom* (*j*,*b*)) = *Ttrue*) ∧ *b* ∈ (*S j*)
and *i*∈*I* ∧ *j*∈*I* ∧ *i*≠*j*
and (*a* ∈ (*S i*) ∩ (*S j*) ∧ *i*∈*I* ∧ *j*∈*I* ∧ *i*≠*j* ⟶
(*t-v-evaluation M* (¬.(*atom* (*i*,*a*) ∧. *atom*(*j*,*a*))) = *Ttrue*))
shows *a* ≠ *b*
⟨*proof*⟩

lemma *model-of-all*:
assumes *M model* (𝓣 *S I*)
shows *M model* (𝓕 *S I*) **and** *M model* (𝓖 *S I*) **and** *M model* (𝓗 *S I*)
⟨*proof*⟩

lemma *sets-have-distinct-representants*:
assumes
hip1: $i \in I$ **and** *hip2*: $j \in I$ **and** *hip3*: $i \neq j$ **and** *hip4*: *M model* ($\mathcal{T}$ *S I*)
and *hip5*: *finite*(*S i*) **and** *hip6*: *finite*(*S j*)
shows *SDR M S I i* $\neq$ *SDR M S I j*
⟨*proof*⟩

lemma *satisfiable-representant*:
assumes *satisfiable* ($\mathcal{T}$ *S I*) **and** $\forall i \in I$. *finite* (*S i*)
shows $\exists R$. *system-representatives S I R*
⟨*proof*⟩

theorem *Hall*:
fixes *S* :: ('*a*::*countable* ⇒ '*b*::*countable set*) **and** *I* :: '*a set*
assumes *Finite*: $\forall i \in I$. *finite* (*S i*)
and *Marriage*: $\forall J \subseteq I$. *finite J* $\longrightarrow$ *card J* $\leq$ *card* ($\bigcup$ (*S* ` *J*))
shows $\exists R$. *system-representatives S I R*
⟨*proof*⟩

theorem *marriage-necessity*:
fixes *A* :: '*a* ⇒ '*b set* **and** *I* :: '*a set*
assumes $\forall i \in I$. *finite* (*A i*)
and $\exists R$. ($\forall i \in I$. *R i* $\in$ *A i*) $\wedge$ *inj-on R I* (**is** $\exists R$. *?R R A* & *?inj R A*)
shows $\forall J \subseteq I$. *finite J* $\longrightarrow$ *card J* $\leq$ *card* ($\bigcup$(*A* ` *J*))
⟨*proof*⟩

end

theory *Hall-Theorem-Graphs*
imports
Background-on-graphs
HOL−Library.Countable-Set
Hall-Theorem

begin

8 Hall Theorem for countable (infinite) Graphs

This section formalizes Hall Theorem for countable infinite Graphs ([5]). The proof applied a proof of Hall's theorem for countable infinite families of sets, obtained by the authors directly from the compactness theorem for propositional logic. The proof is based on Smullyan's approach given in the third chapter of his influential textbook on mathematical logic [3], based on Henkin's model existence theorem. It follows the impeccable presentation in Fitting's textbook [1].

definition *dirBD-to-Hall*::
('a,'b) *pre-digraph* ⇒ 'a *set* ⇒ 'a *set* ⇒ 'a *set* ⇒ ('a ⇒ 'a *set*) ⇒ *bool*
where
dirBD-to-Hall G X Y I S ≡
dir-bipartite-digraph G X Y ∧ *I* = *X* ∧ (∀ *v*∈*I*. (*S v*) = (*neighbourhood G v*))

theorem *dir-BD-to-Hall*:
dirBD-perfect-matching G X Y E ⟶
system-representatives (*neighbourhood G*) *X* (*E-head G E*)
⟨*proof*⟩

lemma *marriage-necessary-graph*:
assumes (*dirBD-perfect-matching G X Y E*) **and** ∀ *i*∈*X*. *finite* (*neighbourhood G i*)
shows ∀ *J*⊆*X*. *finite J* ⟶ (*card J*) ≤ *card* (⋃ (*neighbourhood G* ‘ *J*))
⟨*proof*⟩

lemma *neighbour3*:
fixes *G* :: ('a, 'b) *pre-digraph* **and** *X*:: 'a *set*
assumes *dir-bipartite-digraph G X Y* **and** *x*∈*X*
shows *neighbourhood G x* = {*y* |*y*. ∃ *e*. *e* ∈ *arcs G* ∧ ((*x* = *tail G e*) ∧ (*y* = *head G e*))}
⟨*proof*⟩

lemma *perfect*:
fixes *G* :: ('a, 'b) *pre-digraph* **and** *X*:: 'a *set*
assumes *dir-bipartite-digraph G X Y* **and** *system-representatives* (*neighbourhood G*) *X R*
shows *tails-set G* {*e* |*e*. *e* ∈ (*arcs G*) ∧ ((*tail G e*) ∈ *X* ∧ (*head G e*) = *R*(*tail G e*))} = *X*
⟨*proof*⟩

lemma *dirBD-matching*:
fixes *G* :: ('a, 'b) *pre-digraph* **and** *X*:: 'a *set*
assumes *dir-bipartite-digraph G X Y* **and** *R*: *system-representatives* (*neighbourhood G*) *X R*
and *e1* ∈ *arcs G* ∧ *tail G e1* ∈ *X* **and** *e2* ∈ *arcs G* ∧ *tail G e2* ∈ *X*
and *R*(*tail G e1*) = *head G e1*
and *R*(*tail G e2*) = *head G e2*
shows *e1* ≠ *e2* ⟶ *head G e1* ≠ *head G e2* ∧ *tail G e1* ≠ *tail G e2*
⟨*proof*⟩

lemma *marriage-sufficiency-graph*:
fixes *G* :: ('a::*countable*, 'b::*countable*) *pre-digraph* **and** *X*:: 'a *set*
assumes *dir-bipartite-digraph G X Y* **and** ∀ *i*∈*X*. *finite* (*neighbourhood G i*)
shows
(∀ *J*⊆*X*. *finite J* ⟶ (*card J*) ≤ *card* (⋃ (*neighbourhood G* ‘ *J*))) ⟶

($\exists E$. *dirBD-perfect-matching G X Y E*)
⟨*proof*⟩

theorem *Hall-digraph*:
fixes G :: ('*a*::*countable*, '*b*::*countable*) *pre-digraph* **and** X:: '*a set*
assumes *dir-bipartite-digraph G X Y* **and** $\forall i \in X$. *finite* (*neighbourhood G i*)
shows ($\exists E$. *dirBD-perfect-matching G X Y E*) $\longleftrightarrow$
($\forall J \subseteq X$. *finite J* $\longrightarrow$ (*card J*) $\leq$ *card* ($\bigcup$ (*neighbourhood G* ‘ *J*)))
⟨*proof*⟩

locale *set-family* =
fixes I :: '*a set* **and** X :: '*a* $\Rightarrow$ '*b set*

locale *sdr* = *set-family* +
fixes *repr* :: '*a* $\Rightarrow$ '*b*
assumes *inj-repr*: *inj-on repr I* **and** *repr-X*: $x \in I \Longrightarrow$ *repr* $x \in X\ x$

locale *bipartite-digraph* =
fixes X :: '*a set* **and** Y :: '*b set* **and** E :: ('*a* $\times$ '*b*) *set*
assumes *E-subset*: $E \subseteq X \times Y$

locale *Count-Nbhdfin-bipartite-digraph* =
fixes X :: '*a*:: *countable set* **and** Y :: '*b*:: *countable set*
and E :: ('*a* $\times$ '*b*) *set*
assumes *E-subset*: $E \subseteq X \times Y$

assumes *Nbhd-Tail-finite*: $\forall x \in X$. *finite* $\{y.\ (x, y) \in E\}$

locale *matching* = *bipartite-digraph* +
fixes M :: ('*a* $\times$ '*b*) *set*
assumes *M-subset*: $M \subseteq E$
assumes *M-right-unique*: $(x, y) \in M \Longrightarrow (x, y') \in M \Longrightarrow y = y'$
assumes *M-left-unique*: $(x, y) \in M \Longrightarrow (x', y) \in M \Longrightarrow x = x'$

locale *perfect-matching* = *matching* +
  assumes *M-perfect*: *fst* ‘ *M* = *X*

lemma (**in** *sdr*) *perfect-matching*:
    perfect-matching I ($\bigcup$ *i*∈*I*. *X i*) (*Sigma I X*) {(*x*, *repr x*)|*x*. *x* ∈ *I*}
⟨*proof*⟩

lemma (**in** *perfect-matching*) *sdr*: *sdr X* (λ*x*. {*y*. (*x*,*y*) ∈ *E*}) (λ*x*. *the-elem* {*y*. (*x*,*y*) ∈ *M*})
⟨*proof*⟩

From these transformations, the formalization of the countable version of Hall's Theorem for Graphs (more specifically, its sufficiency) can be stated as below; in words "if for any finite $X_s \subseteq X$ the subgraph induced by X_s has a perfect matching then the whole graph has a perfect matching"

theorem (**in** *Count-Nbhdfin-bipartite-digraph*) *Hall-Graph*:
 assumes ∃ *g*. *enumeration* (*g*:: *nat* ⇒′*a*) **and** ∃ *h*. *enumeration* (*h*:: *nat* ⇒′*b*)
 shows (∀ *Xs* ⊆ *X*. (*finite Xs*) ⟶
      (∃ *Ms*. *perfect-matching Xs*
          {*y*. *x* ∈ *Xs* ∧ (*x*,*y*) ∈ *E*}
          {(*x*,*y*). *x* ∈ *Xs* ∧ (*x*,*y*) ∈ *E*}
          Ms))
    ⟶ (∃ *M*. *perfect-matching X Y E M*)
⟨*proof*⟩

end

9 de Bruijn-Erdős k-coloring theorem for countable infinite graphs

This section formalizes de Bruijn-Erdős k-coloring theorem for countable infinite graphs. The construction applies the compactness theorem for propositional logic directly.

type-synonym ′*v digraph* = (′*v set*) × ((′*v* × ′*v*) *set*)

abbreviation *vert* :: ′*v digraph* ⇒ ′*v set* (‹*V*[-]› [*80*] *80*) **where**
  V[*G*] ≡ *fst G*

abbreviation *edge* :: ′*v digraph* ⇒ (′*v* × ′*v*) *set* (‹*E*[-]› [*80*] *80*) **where**
  E[*G*] ≡ *snd G*

definition *is-graph* :: *'v digraph* ⇒ *bool* **where**
$\mathit{is\text{-}graph}\ G \equiv \forall\ u\ v.\ (u,v) \in E[G] \longrightarrow u \in V[G] \land v \in V[G] \land u \neq v$

definition *is-induced-subgraph* :: *'v digraph* ⇒ *'v digraph* ⇒ *bool* **where**
$\mathit{is\text{-}induced\text{-}subgraph}\ H\ G \equiv$
$(V[H] \subseteq V[G]) \land E[H] = E[G] \cap ((V[H]) \times (V[H]))$

lemma
assumes *is-graph G* **and** *is-induced-subgraph H G*
shows *is-graph H*⟨*proof*⟩

definition *coloring* :: (*'v* ⇒ *nat*) ⇒ *nat* ⇒ *'v digraph* ⇒ *bool* **where**
$\mathit{coloring}\ c\ k\ G \equiv$
$(\forall u.\ u \in V[G] \longrightarrow c(u) \leq k) \land (\forall u\ v.(u,v) \in E[G] \longrightarrow c(u) \neq c(v))$

definition *colorable* :: *'v digraph* ⇒ *nat* ⇒ *bool* **where**
$\mathit{colorable}\ G\ k \equiv \exists c.\ \mathit{coloring}\ c\ k\ G$

primrec *atomic-disjunctions* :: *'v* ⇒ *nat* ⇒ (*'v* × *nat*)*formula* **where**
atomic-disjunctions v 0 = *atom* (*v*, *0*)
| *atomic-disjunctions v* (*Suc k*) =
(*atom* (*v*, *Suc k*)) ∨. (*atomic-disjunctions v k*)

definition $\mathcal{F}$ :: *'v digraph* ⇒ *nat* ⇒ ((*'v* × *nat*)*formula*) *set* **where**
$\mathcal{F}\ G\ k \equiv (\bigcup v \in V[G].\ \{\mathit{atomic\text{-}disjunctions}\ v\ k\})$

definition $\mathcal{G}$ :: *'v digraph* ⇒ *nat* ⇒ (*'v* × *nat*)*formula set* **where**
$\mathcal{G}\ G\ k \equiv \{\neg.(\mathit{atom}\ (v,\ i) \land.\ \mathit{atom}(v,j))$
$\mid v\ i\ j.\ (v \in V[G]) \land (0 \leq i \land 0 \leq j \land i \leq k \land j \leq k \land i \neq j)\}$

definition $\mathcal{H}$ :: *'v digraph* ⇒ *nat* ⇒ (*'v* × *nat*)*formula set* **where**
$\mathcal{H}\ G\ k \equiv \{\neg.(\mathit{atom}\ (u,\ i) \land.\ \mathit{atom}(v,i))$
$\mid u\ v\ i\ .\ (u \in V[G] \land v \in V[G] \land (u,v) \in E[G]) \land (0 \leq i \land i \leq k)\}$

definition $\mathcal{T}$ :: *'v digraph* ⇒ *nat* ⇒ (*'v* × *nat*)*formula set* **where**
$\mathcal{T}\ G\ k \equiv (\mathcal{F}\ G\ k) \cup (\mathcal{G}\ G\ k) \cup (\mathcal{H}\ G\ k)$

primrec *vertices-formula* :: (*'v* × *nat*)*formula* ⇒ *'v set* **where**
vertices-formula FF = {}
| *vertices-formula TT* = {}
| *vertices-formula* (*atom P*) = {*fst P*}
| *vertices-formula* (¬. *F*) = *vertices-formula F*
| *vertices-formula* (*F* ∧. *G*) = *vertices-formula F* ∪ *vertices-formula G*
| *vertices-formula* (*F* ∨. *G*) = *vertices-formula F* ∪ *vertices-formula G*

| *vertices-formula* $(F \rightarrow. G)$ = *vertices-formula* $F \cup$ *vertices-formula* G

definition *vertices-set-formulas* :: $('v \times nat)$*formula set* $\Rightarrow$ $'v$ *set* **where**
vertices-set-formulas $S = (\bigcup F \in S.$ *vertices-formula* $F)$

lemma *finite-vertices*:
shows *finite* (*vertices-formula* F)
⟨*proof*⟩

lemma *vertices-disjunction*:
assumes F = *atomic-disjunctions* v k **shows** *vertices-formula* $F = \{v\}$
⟨*proof*⟩

lemma *all-vertices-colored*:
shows *vertices-set-formulas* $(\mathcal{F}\ G\ k) \subseteq V[G]$
⟨*proof*⟩

lemma *vertices-maximumC*:
shows *vertices-set-formulas*$(\mathcal{G}\ G\ k) \subseteq V[G]$
⟨*proof*⟩

lemma *distinct-verticesC*:
shows *vertices-set-formulas*$(\mathcal{H}\ G\ k) \subseteq V[G]$
⟨*proof*⟩

lemma *vv*:
shows *vertices-set-formulas* $(A \cup B)$ = (*vertices-set-formulas* A) $\cup$ (*vertices-set-formulas* B)
⟨*proof*⟩

lemma *vv1*:
assumes $F \in (\mathcal{F}\ G\ k)$
shows (*vertices-formula* F) $\subseteq$ (*vertices-set-formulas* $(\mathcal{F}\ G\ k)$)
⟨*proof*⟩

lemma *vv2*:
assumes $F \in (\mathcal{G}\ G\ k)$
shows (*vertices-formula* F) $\subseteq$ (*vertices-set-formulas* $(\mathcal{G}\ G\ k)$)
⟨*proof*⟩

lemma *vv3*:
assumes $F \in (\mathcal{H}\ G\ k)$
shows (*vertices-formula* F) $\subseteq$ (*vertices-set-formulas* $(\mathcal{H}\ G\ k)$)
⟨*proof*⟩

lemma *vertex-set-inclusion*:

shows *vertices-set-formulas* $(\mathcal{T}\ G\ k) \subseteq V[G]$
⟨*proof*⟩

lemma *vsf*:
assumes $G \subseteq H$
shows *vertices-set-formulas* $G \subseteq$ *vertices-set-formulas* H
⟨*proof*⟩

lemma *vertices-subset-formulas*:
assumes $S \subseteq (\mathcal{T}\ G\ k)$
shows *vertices-set-formulas* $S \subseteq V[G]$
⟨*proof*⟩

definition *subgraph-aux* :: *'v digraph* ⇒ *'v set* ⇒ *'v digraph* **where**
subgraph-aux $G\ V \equiv (V,\ E[G] \cap (V \times V))$

lemma *induced-subgraph*:
assumes *is-graph* G **and** $S \subseteq (\mathcal{T}\ G\ k)$
shows *is-induced-subgraph* (*subgraph-aux* G (*vertices-set-formulas* S)) G
⟨*proof*⟩

lemma *finite-subgraph*:
assumes *is-graph* G **and** $S \subseteq (\mathcal{T}\ G\ k)$ **and** *finite* S
shows *finite-graph* (*subgraph-aux* G (*vertices-set-formulas* S))
⟨*proof*⟩

fun *graph-interpretation* :: *'v digraph* ⇒ (*'v* ⇒ *nat*) ⇒ ((*'v* × *nat*) ⇒ *v-truth*)
where
graph-interpretation $G\ f = (\lambda(v,i).($*if* $v \in V[G] \wedge f(v) = i$ *then Ttrue else Ffalse*))

lemma *value1*:
assumes $v \in V[G]$ **and** $f(v) \leq k$ **and** $F =$ *atomic-disjunctions* $v\ k$
shows *t-v-evaluation* (*graph-interpretation* $G\ f$) $F =$ *Ttrue*
⟨*proof*⟩

lemma *t-value-vertex*:
assumes *t-v-evaluation* (*graph-interpretation* $G\ f$) (*atom* (v, i)) $=$ *Ttrue*
shows $f(v)=i$
⟨*proof*⟩

lemma *value2*:
assumes $i \neq j$ **and** $F = \neg.($*atom* $(v, i) \wedge.$ *atom* $(v, j))$

shows *t-v-evaluation* (*graph-interpretation* *G f*) *F* = *Ttrue*
⟨*proof*⟩

lemma *value3*:
assumes $f(u) \neq f(v)$ **and** *F* =¬.(*atom* (*u*, *i*) ∧. *atom* (*v*, *i*))
shows *t-v-evaluation* (*graph-interpretation* *G f*) *F* = *Ttrue*
⟨*proof*⟩

theorem *coloring-satisfiable*:
assumes *is-graph G* **and** *S* ⊆ (𝒯 *G k*) **and**
coloring f k (*subgraph-aux G* (*vertices-set-formulas S*))
shows *satisfiable S*
⟨*proof*⟩

fun *graph-coloring* :: ((′*v* × *nat*) ⇒ *v-truth*) ⇒ *nat* ⇒ (′*v* ⇒ *nat*)
where
graph-coloring I k = (λ*v*.(*THE i*. (*t-v-evaluation I* (*atom* (*v*,*i*)) = *Ttrue*) ∧ *0*≤*i* ∧ *i*≤*k*))

lemma *unicity*:
assumes (*t-v-evaluation I* (*atom* (*v*, *i*)) = *Ttrue* ∧ *0*≤*i* ∧ *i* ≤ *k*)
and ∀ *j*. (*0*≤*j* ∧ *j*≤*k* ∧ *i*≠*j*) ⟶ (*t-v-evaluation I* (¬.(*atom* (*v*, *i*) ∧. *atom*(*v*,*j*))) = *Ttrue*)
shows ∀ *j*. (*0*≤*j* ∧ *j*≤*k* ∧ *i*≠*j*) ⟶ *t-v-evaluation I* (*atom* (*v*, *j*)) = *Ffalse*
⟨*proof*⟩

lemma *existence*:
assumes (*t-v-evaluation I* (*atom* (*v*, *i*)) = *Ttrue* ∧ *0*≤*i* ∧ *i* ≤ *k*)
and ∀ *j*. (*0*≤*j* ∧ *j*≤*k* ∧ *i*≠*j*) ⟶ *t-v-evaluation I* (*atom* (*v*, *j*)) = *Ffalse*
shows (∀ *x*. (*t-v-evaluation I* (*atom* (*v*, *x*)) = *Ttrue* ∧ *0*≤*x* ∧ *x* ≤ *k*) ⟶ *x* = *i*)
⟨*proof*⟩

lemma *exist-unicity1*:
assumes (*t-v-evaluation I* (*atom* (*v*, *i*)) = *Ttrue* ∧ *0*≤*i* ∧ *i* ≤ *k*)
and ∀ *j*. (*0*≤*j* ∧ *j*≤*k* ∧ *i*≠*j*) ⟶ (*t-v-evaluation I* (¬.(*atom* (*v*, *i*) ∧. *atom*(*v*,*j*))) = *Ttrue*)
shows (∀ *x*. (*t-v-evaluation I* (*atom* (*v*, *x*)) = *Ttrue* ∧ *0*≤*x* ∧ *x* ≤ *k*) ⟶ *x* = *i*)
⟨*proof*⟩

lemma *exist-unicity2*:
assumes (*t-v-evaluation I* (*atom* (*v*, *i*)) = *Ttrue* ∧ *0*≤*i* ∧ *i* ≤ *k*) **and**
(⋀*x*. (*t-v-evaluation I* (*atom* (*v*, *x*)) = *Ttrue* ∧ *0*≤*x* ∧ *x* ≤ *k*) ⟹ *x* = *i*)
shows (*THE a*. (*t-v-evaluation I* (*atom* (*v*,*a*)) = *Ttrue* ∧ *0*≤*a* ∧ *a* ≤ *k*)) = *i*
⟨*proof*⟩

lemma *exist-unicity*:

assumes (*t-v-evaluation I* (*atom* (*v*, *i*)) = *Ttrue* ∧ *0*≤*i* ∧ *i*≤*k*) **and**
∀ *j*. (*0*≤*j* ∧ *j*≤*k* ∧ *i*≠*j*) ⟶ (*t-v-evaluation I* (¬.(*atom* (*v*, *i*) ∧. *atom*(*v*,*j*))) = *Ttrue*)
shows (*THE a*. (*t-v-evaluation I* (*atom* (*v*,*a*)) = *Ttrue* ∧ *0*≤*a* ∧ *a* ≤ *k*)) = *i*
⟨*proof*⟩

lemma *unique-color*:
assumes *v* ∈ *V*[*G*]
shows ∀ *i j*.(*0*≤*i* ∧ *0*≤*j* ∧ *i*≤*k* ∧ *j*≤*k* ∧ *i*≠*j*) ⟶ (¬.(*atom* (*v*, *i*) ∧. *atom*(*v*,*j*))∈ (𝒢 *G k*))
⟨*proof*⟩

lemma *different-colors*:
assumes *u* ∈ *V*[*G*] **and** *v*∈*V*[*G*] **and** (*u*,*v*)∈*E*[*G*]
shows ∀ *i*.(*0*≤*i* ∧ *i*≤*k*) ⟶ (¬.(*atom* (*u*, *i*) ∧. *atom*(*v*,*i*))∈ (ℋ *G k*))
⟨*proof*⟩

lemma *atom-value*:
assumes (*t-v-evaluation I* (*atomic-disjunctions u k*)) = *Ttrue*
shows ∃ *i*.(*t-v-evaluation I* (*atom* (*u*,*i*)) = *Ttrue*) ∧ *0*≤*i* ∧ *i*≤*k*
⟨*proof*⟩

lemma *coloring-function*:
assumes *u* ∈ *V*[*G*] **and** *I model* (𝒯 *G k*)
shows ∃!*i*. (*t-v-evaluation I* (*atom* (*u*,*i*)) = *Ttrue* ∧ *0*≤*i* ∧ *i*≤*k*) ∧ *graph-coloring I k u* = *i*
⟨*proof*⟩

lemma ℋ*1*:
assumes (*t-v-evaluation I* (*atom* (*u*, *a*)) = *Ttrue* ∧ *0*≤*a* ∧ *a*≤*k*) **and** (*t-v-evaluation I* (*atom* (*v*, *b*)) = *Ttrue* ∧ *0*≤*b* ∧ *b*≤*k*)
and ∀ *i*.(*0*≤*i* ∧ *i*≤*k*) ⟶ (*t-v-evaluation I* (¬.(*atom* (*u*, *i*) ∧. *atom*(*v*,*i*))) = *Ttrue*)
shows *a*≠*b*
⟨*proof*⟩

lemma *distinct-colors*:
assumes *is-graph G* **and** (*u*,*v*) ∈ *E*[*G*] **and** *I*: *I model* (𝒯 *G k*)
shows *graph-coloring I k u* ≠ *graph-coloring I k v*
⟨*proof*⟩

theorem *satisfiable-coloring*:
assumes *is-graph G* **and** *satisfiable* (𝒯 *G k*)
shows *colorable G k*
⟨*proof*⟩

theorem *deBruijn-Erdos-coloring*:
assumes *is-graph* (*G*::(*'vertices*:: *countable*) *set* × (*'vertices* × *'vertices*) *set*)
and ∀ *H*. (*is-induced-subgraph H G* ∧ *finite-graph H* ⟶ *colorable H k*)
shows *colorable G k*
⟨*proof*⟩

end

10 König Lemma

This section formalizes König Lemma from the compactness theorem for propositional logic directly.

type-synonym *'a rel* = (*'a* × *'a*) *set*

definition *irreflexive-on* :: *'a set* ⇒ *'a rel* ⇒ *bool*
where *irreflexive-on A r* ≡ (∀ *x*∈*A*. (*x*, *x*) ∉ *r*)

definition *transitive-on* :: *'a set* ⇒ *'a rel* ⇒ *bool*
where *transitive-on A r* ≡
(∀ *x*∈*A*. ∀ *y*∈*A*. ∀ *z*∈*A*. (*x*, *y*) ∈ *r* ∧ (*y*, *z*) ∈ *r* ⟶ (*x*, *z*) ∈ *r*)

definition *total-on* :: *'a set* ⇒ *'a rel* ⇒ *bool*
where *total-on A r* ≡ (∀ *x*∈*A*. ∀ *y*∈*A*. *x* ≠ *y* ⟶ (*x*, *y*) ∈ *r* ∨ (*y*, *x*) ∈ *r*)

definition *minimum* :: *'a set* ⇒ *'a* ⇒*'a rel* ⇒ *bool*
where *minimum A a r* ≡ (*a*∈*A* ∧ (∀ *x*∈*A*. *x* ≠ *a* ⟶ (*a*,*x*) ∈ *r*))

definition *predecessors* :: *'a set* ⇒*'a* ⇒*'a rel* ⇒ *'a set*
where *predecessors A a r* ≡ {*x*∈*A*.(*x*, *a*) ∈ *r*}

definition *height* :: *'a set* ⇒*'a* ⇒ *'a rel* ⇒ *nat*
where *height A a r* ≡ *card* (*predecessors A a r*)

definition *level* :: *'a set* ⇒ *'a rel* ⇒ *nat* ⇒*'a set*
where *level A r n* ≡ {*x*∈*A*. *height A x r* = *n*}

definition *imm-successors* :: *'a set* ⇒ *'a* ⇒ *'a rel* ⇒ *'a set*
where *imm-successors A a r* ≡
{*x*∈*A*. (*a*,*x*)∈ *r* ∧ *height A x r* = (*height A a r*)+*1*}

definition *strict-part-order* :: *'a set* ⇒ *'a rel* ⇒ *bool*
where *strict-part-order A r* ≡ *irreflexive-on A r* ∧ *transitive-on A r*

lemma *minimum-element*:
assumes *strict-part-order A r* **and** *minimum A a r* **and** *r*={}

shows *A*={*a*}
⟨*proof*⟩

lemma *spo-uniqueness-min*:
assumes *strict-part-order A r* **and** *minimum A a r* **and** *minimum A b r*
shows *a*=*b*
⟨*proof*⟩

lemma *emptyness-pred-min-spo*:
assumes *minimum A a r* **and** *strict-part-order A r*
shows *predecessors A a r* = {}
⟨*proof*⟩

lemma *emptyness-pred-min-spo2*:
assumes *strict-part-order A r* **and** *minimum A a r*
shows ∀ *x*∈*A*.(*predecessors A x r* = {}) ⟷ (*x*=*a*)
⟨*proof*⟩

lemma *height-minimum*:
assumes *strict-part-order A r* **and** *minimum A a r*
shows *height A a r* = *0*
⟨*proof*⟩

lemma *zero-level*:
assumes *strict-part-order A r*
and *minimum A a r* **and** ∀ *x*∈*A*. *finite* (*predecessors A x r*)
shows (*level A r 0*) = {*a*}
⟨*proof*⟩

lemma *min-predecessor*:
assumes *minimum A a r*
shows ∀ *x*∈*A*. *x*≠*a* ⟶ *a*∈*predecessors A x r*
⟨*proof*⟩

lemma *spo-subset-preservation*:
assumes *strict-part-order A r* **and** *B*⊆*A*
shows *strict-part-order B r*
⟨*proof*⟩

lemma *total-ord-subset-preservation*:
assumes *total-on A r* **and** *B*⊆*A*
shows *total-on B r*
⟨*proof*⟩

definition *maximum* :: *'a set* ⇒ *'a* ⇒*'a rel* ⇒ *bool*
where *maximum A a r* ≡ (*a*∈*A* ∧ (∀ *x*∈*A*. *x* ≠ *a* ⟶ (*x*,*a*) ∈ *r*))

lemma *maximum-strict-part-order*:
assumes *strict-part-order A r* **and** *A*≠{} **and** *total-on A r*

and *finite A*
shows (∃ *a*. *maximum A a r*)
⟨*proof*⟩

lemma *finiteness-union-finite-sets*:
fixes *S* :: *'a* ⇒ *'a set*
assumes ∀ *x*. *finite* (*S x*) **and** *finite A*
shows *finite* (⋃ *a*∈*A*. (*S a*)) ⟨*proof*⟩

lemma *uniqueness-level-aux*:
assumes *k*>*0*
shows (*level A r n*) ∩ (*level A r* (*n*+*k*)) = {}
⟨*proof*⟩

lemma *uniqueness-level*:
assumes *n*≠*m*
shows (*level A r n*) ∩ (*level A r m*) = {}
⟨*proof*⟩

definition *tree* :: *'a set* ⇒ *'a rel* ⇒ *bool*
where *tree A r* ≡
r ⊆ *A* × *A* ∧ *r*≠{} ∧ (*strict-part-order A r*) ∧ (∃ *a*. *minimum A a r*) ∧
(∀ *a*∈*A*. *finite* (*predecessors A a r*) ∧ (*total-on* (*predecessors A a r*) *r*))

definition *finite-tree*:: *'a set* ⇒ *'a rel* ⇒ *bool*
where
finite-tree A r ≡ *tree A r* ∧ *finite A*

abbreviation *infinite-tree*:: *'a set* ⇒ *'a rel* ⇒ *bool*
where
infinite-tree A r ≡ *tree A r* ∧ ¬ *finite A*

definition *enumerable-tree* :: *'a set* ⇒ *'a rel* ⇒ *bool* **where**
enumerable-tree A r ≡ ∃ *g*. *enumeration* (*g*:: *nat* ⇒*'a*)

definition *finitely-branching* :: *'a set* ⇒ *'a rel* ⇒ *bool*
where *finitely-branching A r* ≡ (∀ *x*∈*A*. *finite* (*imm-successors A x r*))

definition *sub-linear-order* :: *'a set* ⇒ *'a set* ⇒ *'a rel* ⇒ *bool*
where *sub-linear-order B A r* ≡ *B*⊆*A* ∧ (*strict-part-order A r*) ∧ (*total-on B r*)

definition *path* :: *'a set* ⇒ *'a set* ⇒ *'a rel* ⇒ *bool*
where *path B A r* ≡
(*sub-linear-order B A r*) ∧
(∀ *C*. *B* ⊆ *C* ∧ *sub-linear-order C A r* ⟶ *B* = *C*)

definition *finite-path*:: *'a set* ⇒ *'a set* ⇒ *'a rel* ⇒ *bool*
where *finite-path B A r* ≡ *path B A r* ∧ *finite B*

definition *infinite-path*:: *'a set* ⇒ *'a set* ⇒ *'a rel* ⇒ *bool*
where *infinite-path B A r* ≡ *path B A r* ∧ ¬ *finite B*

lemma *tree*:
assumes *tree A r*
shows
r ⊆ *A* × *A* **and** *r*≠{}
and *strict-part-order A r*
and ∃ *a*. *minimum A a r*
and (∀ *a*∈*A*. *finite* (*predecessors A a r*) ∧ (*total-on* (*predecessors A a r*) *r*))
⟨*proof*⟩

lemma *non-empty*:
assumes *tree A r* **shows** *A*≠{}
⟨*proof*⟩

lemma *predecessors-spo*:
assumes *tree A r*
shows ∀ *x*∈*A*. *strict-part-order* (*predecessors A x r*) *r*
⟨*proof*⟩

lemma *predecessors-maximum*:
assumes *tree A r* **and** *minimum A a r*
shows ∀ *x*∈*A*. *x*≠*a* ⟶ (∃ *b*. *maximum* (*predecessors A x r*) *b r*)
⟨*proof*⟩

lemma *non-empty-preds-in-tree*:
assumes *tree A r* **and** *card* (*predecessors A x r*) = *n*+*1*
shows *x*∈*A*
⟨*proof*⟩

lemma *imm-predecessor*:
assumes *tree A r*
and *card* (*predecessors A x r*) = *n*+*1* **and**
maximum (*predecessors A x r*) *b r*
shows *height A b r* = *n*
⟨*proof*⟩

lemma *height*:
assumes *tree A r* **and** *height A x r* = *n*+*1*
shows ∃ *y*. (*y*,*x*)∈*r* ∧ *height A y r* = *n*
⟨*proof*⟩

lemma *level*:
assumes *tree A r* **and** *x* ∈ (*level A r* (*n*+*1*))
shows ∃ *y*. (*y*,*x*)∈*r* ∧ *y* ∈ (*level A r n*)
⟨*proof*⟩

primrec *set-nodes-at-level* :: *'a set* ⇒ *'a rel* ⇒ *nat* ⇒*'a set* **where**
set-nodes-at-level A r 0 = {*a*. (*minimum A a r*)}
| *set-nodes-at-level A r* (*Suc n*) = (⋃ *a*∈ (*set-nodes-at-level A r n*). *imm-successors A a r*)

lemma *set-nodes-at-level-zero-spo*:
  assumes *strict-part-order A r* **and** *minimum A a r*
  shows (*set-nodes-at-level A r 0*) = {*a*}
⟨*proof*⟩

lemma *height-level*:
  assumes *strict-part-order A r* **and** *minimum A a r*
  and *x* ∈ *set-nodes-at-level A r n*
  shows *height A x r* = *n*
⟨*proof*⟩

lemma *level-func-vs-level-def*:
  assumes *tree A r*
  shows *set-nodes-at-level A r n* = *level A r n*
⟨*proof*⟩

lemma *pertenece-level*:
  assumes *x* ∈ *set-nodes-at-level A r n*
  shows *x*∈*A*
⟨*proof*⟩

lemma *finiteness-set-nodes-at-levela*:
  assumes ∀ *x*∈*A*. *finite* (*imm-successors A x r*) **and** *finite* (*set-nodes-at-level A r n*)
  shows *finite* (⋃ *a*∈ (*set-nodes-at-level A r n*). *imm-successors A a r*)
⟨*proof*⟩

lemma *finiteness-set-nodes-at-level*:
  assumes *finite* (*set-nodes-at-level A r 0*) **and** *finitely-branching A r*
  shows *finite* (*set-nodes-at-level A r n*)
⟨*proof*⟩

lemma *finite-level*:
  assumes *tree A r* **and** *finitely-branching A r*
  shows *finite* (*level A r n*)
⟨*proof*⟩

lemma *finite-level-a*:
  assumes *tree A r* **and** ∀ *n*. *finite* (*level A r n*)
  shows *finitely-branching A r*
⟨*proof*⟩

lemma *empty-predec*:
  assumes ∀ *x*∈*A*. (*x*,*y*)∉*r*

shows *predecessors A y r* ={}
⟨*proof*⟩

lemma *level-element*:
$\forall x \in A. \exists n.\ x \in level\ A\ r\ n$
⟨*proof*⟩

lemma *union-levels*:
shows $A = (\bigcup n.\ level\ A\ r\ n)$
⟨*proof*⟩

lemma *path-to-node*:
assumes *tree A r* **and** $x \in (level\ A\ r\ (n+1))$
shows $\forall k.(0 \le k \wedge k \le n) \longrightarrow (\exists y.\ (y,x) \in r \wedge y \in (level\ A\ r\ k))$
⟨*proof*⟩

lemma *set-nodes-at-level*:
assumes *tree A r*
shows $(level\ A\ r\ (n+1)) \ne \{\} \longrightarrow (\forall k.(0 \le k \wedge k \le n) \longrightarrow (level\ A\ r\ k) \ne \{\})$
⟨*proof*⟩

lemma *emptyness-below-height*:
assumes *tree A r*
shows $((level\ A\ r\ (n+1)) = \{\}) \longrightarrow (\forall k.\ k > (n+1) \longrightarrow (level\ A\ r\ k) = \{\})$
⟨*proof*⟩

lemma *characterization-nodes-tree-finite-height*:
assumes *tree A r* **and** $\forall k.\ k > m \longrightarrow (level\ A\ r\ k) = \{\}$
shows $A = (\bigcup n \in \{0..m\}.\ level\ A\ r\ n)$
⟨*proof*⟩

lemma *finite-tree-if-fin-branches-and-fin-height*:
assumes *tree A r* **and** *finitely-branching A r*
and $\exists n.\ (\forall k.\ k > n \longrightarrow (level\ A\ r\ k) = \{\})$
shows *finite A*
⟨*proof*⟩

lemma *all-levels-non-empty*:
assumes *infinite-tree A r* **and** *finitely-branching A r*
shows $\forall n.\ level\ A\ r\ n \ne \{\}$
⟨*proof*⟩

lemma *simple-cyclefree*:
assumes *tree A r* **and** $(x,z) \in r$ **and** $(y,z) \in r$ **and** $x \ne y$
shows $(x,y) \in r \vee (y,x) \in r$
⟨*proof*⟩

lemma *inclusion-predecessors*:
assumes $r \subseteq A \times A$ **and** *strict-part-order A r* **and** $(x,y) \in r$

shows (*predecessors A x r*) ⊂ (*predecessors A y r*)
⟨*proof*⟩

lemma *different-height-finite-pred*:
assumes *r* ⊆ *A* × *A* **and** *strict-part-order A r* **and** (*x*,*y*)∈*r*
and *finite* (*predecessors A y r*)
shows *height A x r* < *height A y r*
⟨*proof*⟩

lemma *different-levels-finite-pred*:
assumes *r* ⊆ *A* × *A* **and** *strict-part-order A r* **and** (*x*,*y*)∈*r*
and *x* ∈ (*level A r n*) **and** *y* ∈ (*level A r m*)
and *finite* (*predecessors A y r*)
shows *level A r n* ≠ *level A r m*
⟨*proof*⟩

lemma *less-level-pred-in-fin-pred*:
assumes *r* ⊆ *A* × *A* **and** *strict-part-order A r*
and *x* ∈ *predecessors A y r* **and** *y* ∈ (*level A r n*)
and *x* ∈ (*level A r m*)
and *finite* (*predecessors A y r*)
shows *m*<*n*
⟨*proof*⟩

lemma *emptyness-inter-diff-levels-aux*:
assumes *tree A r* **and** *x*∈(*predecessors A z r*)
and *y*∈(*predecessors A z r*)
and *x*≠*y* **and** *x* ∈ (*level A r n*) **and** *y* ∈ (*level A r m*)
shows *level A r n* ∩ *level A r m* = {}
⟨*proof*⟩

lemma *emptyness-inter-diff-levels*:
assumes *tree A r* **and** (*x*,*z*)∈ *r* **and** (*y*,*z*)∈ *r*
and *x*≠*y* **and** *x* ∈ (*level A r n*) **and** *y* ∈ (*level A r m*)
shows *level A r n* ∩ *level A r m* = {}
⟨*proof*⟩

primrec *disjunction-nodes* :: *'a list* ⇒ *'a formula* **where**
disjunction-nodes [] = *FF*
| *disjunction-nodes* (*v*#*D*) = (*atom v*) ∨. (*disjunction-nodes D*)

lemma *truth-value-disjunction-nodes*:
assumes *v*∈ *set l* **and** *t-v-evaluation I* (*atom v*) = *Ttrue*
shows *t-v-evaluation I* (*disjunction-nodes l*) = *Ttrue*
⟨*proof*⟩

lemma *set-set-to-list1*:
assumes *tree A r* **and** *finitely-branching A r*
shows *set* (*set-to-list* (*level A r n*)) = (*level A r n*)

⟨*proof*⟩

lemma *truth-value-disjunction-formulas*:
assumes *tree A r* **and** *finitely-branching A r*
and $v \in (level\ A\ r\ n) \wedge t\text{-}v\text{-}evaluation\ I\ (atom\ v) = Ttrue$
and $F = disjunction\text{-}nodes(set\text{-}to\text{-}list\ (level\ A\ r\ n))$
shows $t\text{-}v\text{-}evaluation\ I\ \ F = Ttrue$
⟨*proof*⟩

definition $\mathcal{F}$ :: $'a\ set \Rightarrow\ 'a\ rel \Rightarrow ('a\ formula)\ set$ **where**
$\mathcal{F}\ A\ r\ \equiv (\bigcup n.\ \{disjunction\text{-}nodes(set\text{-}to\text{-}list\ (level\ A\ r\ n))\})$

definition $\mathcal{G}$ :: $'a\ set \Rightarrow\ 'a\ rel \Rightarrow ('a\ formula)\ set$ **where**
$\mathcal{G}\ A\ r \equiv \{(atom\ u) \rightarrow.\ (atom\ v)\ |u\ v.\ u{\in}A \wedge v{\in}A \wedge (v,u){\in}\ r\}$

definition $\mathcal{H}n$ :: $'a\ set \Rightarrow\ 'a\ rel \Rightarrow nat \Rightarrow ('a\ formula)\ set$ **where**
$\mathcal{H}n\ A\ r\ n \equiv \{\neg.((atom\ u) \wedge.\ (atom\ v))$
$|u\ v\ .\ u{\in}(level\ A\ r\ n) \wedge v{\in}(level\ A\ r\ n) \wedge u{\neq}v\ \}$
definition $\mathcal{H}$ :: $'a\ set \Rightarrow\ 'a\ rel \Rightarrow ('a\ formula)\ set$ **where**
$\mathcal{H}\ A\ r\ \equiv \bigcup n.\ \mathcal{H}n\ A\ r\ n$

definition $\mathcal{T}$ :: $'a\ set \Rightarrow\ 'a\ rel \Rightarrow ('a\ formula)\ set$ **where**
$\mathcal{T}\ A\ r\ \equiv (\mathcal{F}\ A\ r) \cup (\mathcal{G}\ A\ r) \cup (\mathcal{H}\ A\ r)$

primrec *nodes-formula* :: $'v\ formula\ \Rightarrow\ 'v\ set$ **where**
$nodes\text{-}formula\ FF = \{\}$
$|\ nodes\text{-}formula\ TT = \{\}$
$|\ nodes\text{-}formula\ (atom\ P) =\ \{P\}$
$|\ nodes\text{-}formula\ (\neg.\ F) = nodes\text{-}formula\ F$
$|\ nodes\text{-}formula\ (F\ \wedge.\ G) = nodes\text{-}formula\ F \cup nodes\text{-}formula\ G$
$|\ nodes\text{-}formula\ (F\ \vee.\ G) = nodes\text{-}formula\ F \cup nodes\text{-}formula\ G$
$|\ nodes\text{-}formula\ (F\ \rightarrow.G) = nodes\text{-}formula\ F \cup nodes\text{-}formula\ G$

definition *nodes-set-formulas* :: $'v\ formula\ set\ \Rightarrow\ 'v\ set$ **where**
$nodes\text{-}set\text{-}formulas\ S = (\bigcup F{\in}\ S.\ nodes\text{-}formula\ F)$

definition *maximum-height*:: $'v\ set \Rightarrow 'v\ rel \Rightarrow\ 'v\ \ formula\ \ set\ \Rightarrow\ nat$ **where**
$maximum\text{-}height\ A\ r\ S =\ Max\ (\bigcup x{\in}nodes\text{-}set\text{-}formulas\ S.\ \{height\ A\ x\ r\})$

lemma *node-formula*:
assumes $v \in set\ l$
shows $v \in nodes\text{-}formula\ (disjunction\text{-}nodes\ l)$
⟨*proof*⟩

lemma *node-disjunction-formulas*:
assumes *tree A r* **and** *finitely-branching A r* **and** $v{\in}(level\ A\ r\ n)$
and $F = disjunction\text{-}nodes(set\text{-}to\text{-}list\ (level\ A\ r\ n))$
shows $v \in nodes\text{-}formula\ F$
⟨*proof*⟩

fun *node-sig-level-max*:: $'v$ *set* ⇒ $'v$ *rel* ⇒ $'v$ *formula set* ⇒ $'v$
  where *node-sig-level-max A r S* =
  (*SOME u. u* ∈ (*level A r* ((*maximum-height A r S*)+*1*)))

lemma *node-level-maximum*:
  assumes *infinite-tree A r* **and** *finitely-branching A r*
  shows (*node-sig-level-max A r S*) ∈ (*level A r* ((*maximum-height A r S*)+*1*))
⟨*proof*⟩

fun *path-interpretation* :: $'v$ *set* ⇒$'v$ *rel* ⇒ $'v$ ⇒ ($'v$ ⇒ *v-truth*) **where**
path-interpretation A r u = (λ*v*. (*if* (*v*,*u*)∈*r then Ttrue else Ffalse*))

lemma *finiteness-nodes-formula*:
finite (*nodes-formula F*) ⟨*proof*⟩

lemma *finiteness-set-nodes*:
  assumes *finite S*
  shows *finite* (*nodes-set-formulas S*)
  ⟨*proof*⟩

lemma *maximum1*:
  assumes *finite S* **and** *u* ∈ *nodes-set-formulas S*
  shows (*height A u r*) ≤ (*maximum-height A r S*)
⟨*proof*⟩

lemma *value-path-interpretation*:
  assumes *t-v-evaluation* (*path-interpretation A r v*) (*atom u*) = *Ttrue*
  shows (*u*,*v*)∈*r*
⟨*proof*⟩

lemma *satisfiable-path*:
  assumes *infinite-tree A r*
  and *finitely-branching A r* **and** *S* ⊆ (𝒯 *A r*)
  and *finite S*
shows *satisfiable S*
⟨*proof*⟩

definition ℬ:: $'a$ *set* ⇒ ($'a$ ⇒ *v-truth*) ⇒ $'a$ *set* **where**
ℬ *A I* ≡ {*u*|*u*. *u*∈*A* ∧ *t-v-evaluation I* (*atom u*) = *Ttrue*}

lemma *value-disjunction-list1*:
  assumes *t-v-evaluation I* (*disjunction-nodes* (*a* # *l*)) = *Ttrue*
  shows *t-v-evaluation I* (*atom a*) = *Ttrue* ∨ *t-v-evaluation I* (*disjunction-nodes l*) = *Ttrue*
⟨*proof*⟩

lemma *value-disjunction-list*:
  assumes *t-v-evaluation I* (*disjunction-nodes l*) = *Ttrue*

shows $\exists x.\ x \in set\ l \wedge t\text{-}v\text{-}evaluation\ I\ (atom\ x) = Ttrue$
⟨*proof*⟩

lemma *intersection-branch-set-nodes-at-level*:
assumes *infinite-tree A r* **and** *finitely-branching A r*
and I: $\forall F \in (\mathcal{F}\ A\ r).\ t\text{-}v\text{-}evaluation\ I\ F = Ttrue$
shows $\forall n.\ \exists x.\ x \in level\ A\ r\ n \wedge x \in (\mathcal{B}\ A\ I)$ ⟨*proof*⟩

lemma *intersection-branch-emptyness-below-height*:
assumes I: $\forall F \in (\mathcal{H}\ A\ r).\ t\text{-}v\text{-}evaluation\ I\ F = Ttrue$
and $x \in (\mathcal{B}\ A\ I)$ **and** $y \in (\mathcal{B}\ A\ I)$ **and** $x \neq y$ **and** n: $x \in level\ A\ r\ n$
and m: $y \in level\ A\ r\ m$
shows $n \neq m$
⟨*proof*⟩

lemma *intersection-branch-level*:
assumes *infinite-tree A r* **and** *finitely-branching A r*
and I: $\forall F \in (\mathcal{F}\ A\ r) \cup (\mathcal{H}\ A\ r).\ t\text{-}v\text{-}evaluation\ I\ F = Ttrue$
shows $\forall n.\ \exists u.\ (\mathcal{B}\ A\ I) \cap\ level\ A\ r\ n = \{u\}$
⟨*proof*⟩

lemma *predecessor-in-branch*:
assumes I: $\forall F \in (\mathcal{G}\ A\ r).\ t\text{-}v\text{-}evaluation\ I\ F = Ttrue$
and $y \in (\mathcal{B}\ A\ I)$ **and** $(x,y) \in r$ **and** $x \in A$ **and** $y \in A$
shows $x \in (\mathcal{B}\ A\ I)$
⟨*proof*⟩

lemma *is-path*:
assumes *infinite-tree A r* **and** *finitely-branching A r*
and I: $\forall F \in (\mathcal{T}\ A\ r).\ t\text{-}v\text{-}evaluation\ I\ F = Ttrue$
shows *path* $(\mathcal{B}\ A\ I)\ A\ r$
⟨*proof*⟩

lemma *surjective-infinite*:
assumes $\exists f{::}\ 'a \Rightarrow nat.\ \forall n.\ \exists x \in A.\ n = f(x)$
shows *infinite A*
⟨*proof*⟩

lemma *family-intersection-infinita*:
fixes $P ::\ nat \Rightarrow 'a\ set$
assumes $\forall n.\ \forall m.\ n \neq m \longrightarrow P\ n \cap P\ m = \{\}$
and $\forall n.\ (A \cap (P\ n)) \neq \{\}$
shows *infinite* $(\bigcup n.\ (A \cap (P\ n)))$
⟨*proof*⟩

lemma *infinite-path*:
assumes *infinite-tree A r* **and** *finitely-branching A r*
and I: $\forall F \in (\mathcal{F}\ A\ r).\ t\text{-}v\text{-}evaluation\ I\ F = Ttrue$
shows *infinite* $(\mathcal{B}\ A\ I)$

⟨*proof*⟩

theorem *Koenig-Lemma*:
assumes *infinite-tree* (*A*::′*nodes*:: *countable set*) *r*
and *finitely-branching A r*
shows ∃ *B*. *infinite-path B A r*
⟨*proof*⟩

end

References

[1] M. Fitting. *First-Order Logic and Automated Theorem Proving*. Springer-Verlag, second edition, 1996.

[2] F. F. Serrano Suárez. *Formalización en Isar de la Meta-Lógica de Primer Orden*. PhD thesis, Departamento de Ciencias de la Computación e Inteligencia Artificial, Universidad de Sevilla, Spain, 2012. https://idus.us.es/handle/11441/57780. In Spanish.

[3] R. M. Smullyan. *First-Order Logic*, volume 43 of *Ergebnisse der Mathematik und ihrer Grenzgebiete. 2. Folge*. Springer-Verlag, Berlin, 1968. Also available as a Dover Publications Inc., 1994.

[4] F. F. S. Suárez, M. Ayala-Rincón, and T. A. de Lima. Hall's Theorem for Enumerable Families of Finite Sets. In *Proceedings 15th International Conference on Intelligent Computer Mathematics, CICM*, volume 13467 of *Lecture Notes in Computer Science*, pages 107–121. Springer, 2022.

[5] F. F. S. Suárez, M. Ayala-Rincón, and T. A. de Lima. Formalisation of Hall's Theorem for Countable Infinite Graphs. In *Proceedings 18th Colombian Conference on Computing, CCC*. Springer, 2024.