Clique is not solvable by monotone circuits of polynomial size*

René Thiemann
University of Innsbruck

March 17, 2025

Abstract

Given a graph G with n vertices and a number s, the decision problem Clique asks whether G contains a fully connected subgraph with s vertices. For this NP-complete problem there exists a non-trivial lower bound: no monotone circuit of a size that is polynomial in n can solve Clique.

This entry provides an Isabelle/HOL formalization of a concrete lower bound (the bound is $\sqrt[7]{n}^{\sqrt[8]{n}}$ for the fixed choice of $s = \sqrt[4]{n}$), following a proof by Gordeev.

Contents

*We thank Lev Gordeev for several clarification regarding his proof, for his explanation of the history of the underlying proof idea, and for a lively and ongoing interesting discussion on how his draft can be repaired.

1 Introduction

In this AFP submission we verify the result, that no polynomial-sized circuit can implement the Clique problem.

We arrived at this formalization by trying to verify an unpublished draft of Gordeev [4], which tries to show that Clique cannot be solved by any polynomial-sized circuit, including non-monotone ones, where the concrete exponential lower bound is $\sqrt[7]{n}^{\sqrt[8]{n}}$ for graphs with n vertices and cliques of size $s = \sqrt[4]{n}$.

Although there are some flaws in that draft, all of these disappear if one restricts to monotone circuits. Consequently, the claimed lower bound is valid for monotone circuits.

We verify a simplified version of Gordeev's proof, where those parts that deal with negations in circuits have been eliminated from definitions and proofs.

Gordeev's work itself was inspired by "Razborov's theorem" in a textbook by Papadimitriou [5], which states that Clique cannot be encoded with a monotone circuit of polynomial size. However the proof in the draft uses a construction based on the sunflower lemma of Erds and Rado [3], following a proof in Boppana and Sipser [2]. There are further proofs on lower bounds of monotone circuits for Clique. For instance, an early result is due to Alon and Boppana [1], where they show a slightly different lower bound (using a differently structured proof without the construction based on sunflowers.)

2 Preliminaries

theory *Preliminaries*
  imports
    Main
    HOL.Real
    HOL−Library.FuncSet
begin

lemma *fact-approx-add*: *fact* (*l* + *n*) ≤ *fact l* ∗ (*real l* + *real n*) ^ *n*
proof (*induct n arbitrary*: *l*)
  case (*Suc n l*)
  have *fact* (*l* + *Suc n*) = (*real l* + *Suc n*) ∗ *fact* (*l* + *n*) **by** *simp*
  also have . . . ≤ (*real l* + *Suc n*) ∗ (*fact l* ∗ (*real l* + *real n*) ^ *n*)
    by (*intro mult-left-mono*[*OF Suc*], *auto*)
  also have . . . = *fact l* ∗ ((*real l* + *Suc n*) ∗ (*real l* + *real n*) ^ *n*) **by** *simp*
  also have . . . ≤ *fact l* ∗ ((*real l* + *Suc n*) ∗ (*real l* + *real* (*Suc n*)) ^ *n*)
    by (*rule mult-left-mono*, *rule mult-left-mono*, *rule power-mono*, *auto*)
  finally show *?case* **by** *simp*
qed *simp*

lemma *fact-approx-minus*: **assumes** $k \geq n$
  shows *fact* $k \leq$ *fact* $(k - n) * ($*real* k ^ $n)$
proof −
  define l **where** $l = k - n$
  from *assms* **have** k: $k = l + n$ **unfolding** *l-def* **by** *auto*
  show *?thesis* **unfolding** k **using** *fact-approx-add*[*of* l n] **by** *simp*
qed

lemma *fact-approx-upper-add*: **assumes** *al*: $a \leq$ *Suc* l **shows** *fact* l ∗ *real* a ^ n
$\leq$ *fact* $(l + n)$
proof (*induct* n)
  case (*Suc* n)
  have *fact* l ∗ *real* a ^ (*Suc* n) = (*fact* l ∗ *real* a ^ n) ∗ *real* a **by** *simp*
  also have ... ≤ *fact* $(l + n)$ ∗ *real* a
    by (*rule mult-right-mono*[*OF Suc*], *auto*)
  also have ... ≤ *fact* $(l + n)$ ∗ *real* (*Suc* $(l + n)$)
    by (*intro mult-left-mono*, *insert al*, *auto*)
  also have ... = *fact* (*Suc* $(l + n)$) **by** *simp*
  finally show *?case* **by** *simp*
qed *simp*

lemma *fact-approx-upper-minus*: **assumes** $n \leq k$ **and** $n + a \leq$ *Suc* k
  shows *fact* $(k - n)$ ∗ *real* a ^ $n \leq$ *fact* k
proof −
  define l **where** $l = k - n$
  from *assms* **have** k: $k = l + n$ **unfolding** *l-def* **by** *auto*
  show *?thesis* **using** *assms* **unfolding** k
    apply *simp*
    apply (*rule fact-approx-upper-add*, *insert assms*, *auto simp*: *l-def*)
    done
qed

lemma *choose-mono*: $n \leq m \Longrightarrow n$ *choose* $k \leq m$ *choose* k
  unfolding *binomial-def*
  by (*rule card-mono*, *auto*)

lemma *div-mult-le*: (a *div* b) ∗ c ≤ (a ∗ c) *div* (b :: *nat*)
 by (*metis div-mult2-eq div-mult-mult2 mult.commute mult-0-right times-div-less-eq-dividend*)

lemma *div-mult-pow-le*: (a *div* b)^n ≤ a^n *div* (b :: *nat*)^n
proof (*cases* $b = 0$)
  case *True*
  thus *?thesis* **by** (*cases* n, *auto*)
next
  case b: *False*
  then obtain c d **where** a: $a = b * c + d$ **and** *id*: $c = a$ *div* b $d = a$ *mod* b **by**
auto
  have (a *div* b)^n = c^n **unfolding** *id* **by** *simp*
  also have ... = (b ∗ c)^n *div* b^n **using** b

```
    by (metis div-power dvd-triv-left nonzero-mult-div-cancel-left)
  also have ... ≤ (b * c + d)^n div b^n
    by (rule div-le-mono, rule power-mono, auto)
  also have ... = a^n div b^n  unfolding a by simp
  finally show ?thesis .
qed

lemma choose-inj-right:
  assumes id: (n choose l) = (k choose l)
    and n0: n choose l ≠ 0
    and l0:  l ≠ 0
  shows n = k
proof (rule ccontr)
  assume nk: n ≠ k
  define m where m = min n k
  define M where M = max n k
  from nk have mM: m < M unfolding m-def M-def by auto
  let ?new = insert (M − 1) {0..< l − 1}
  let ?m = {K ∈ Pow {0..<m}. card K = l}
  let ?M = {K ∈ Pow {0..<M}. card K = l}
  from id n0 have lM :l ≤ M unfolding m-def M-def by auto
  from id have id: (m choose l) = (M choose l)
    unfolding m-def M-def by auto
  from this[unfolded binomial-def]
  have card ?M < Suc (card ?m)
    by auto
  also have ... = card (insert ?new ?m)
    by (rule sym, rule card-insert-disjoint, force, insert mM, auto)
  also have ... ≤ card (insert ?new ?M)
    by (rule card-mono, insert mM, auto)
  also have insert ?new ?M = ?M
    by (insert mM lM l0, auto)
  finally show False by simp
qed


end
```

3 Monotone Formulas

We define monotone formulas, i.e., without negation, and show that usually the constant TRUE is not required.

```
theory Monotone-Formula
  imports Main
begin
```

3.1 Definition

datatype $'a$ *mformula* =
  TRUE | *FALSE* | — True and False
  Var $'a$ | — propositional variables
  Conj $'a$ *mformula* $'a$ *mformula* | — conjunction
  Disj $'a$ *mformula* $'a$ *mformula* — disjunction

the set of subformulas of a mformula

fun *SUB* :: $'a$ *mformula* ⇒ $'a$ *mformula set* **where**
  $SUB\ (Conj\ \varphi\ \psi) = \{Conj\ \varphi\ \psi\} \cup SUB\ \varphi \cup SUB\ \psi$
| $SUB\ (Disj\ \varphi\ \psi) = \{Disj\ \varphi\ \psi\} \cup SUB\ \varphi \cup SUB\ \psi$
| $SUB\ (Var\ x) = \{Var\ x\}$
| $SUB\ FALSE = \{FALSE\}$
| $SUB\ TRUE = \{TRUE\}$

the variables of a mformula

fun *vars* :: $'a$ *mformula* ⇒ $'a$ *set* **where**
  $vars\ (Var\ x) = \{x\}$
| $vars\ (Conj\ \varphi\ \psi) = vars\ \varphi \cup vars\ \psi$
| $vars\ (Disj\ \varphi\ \psi) = vars\ \varphi \cup vars\ \psi$
| $vars\ FALSE = \{\}$
| $vars\ TRUE = \{\}$

lemma *finite-SUB*[*simp*, *intro*]: *finite* (*SUB* φ)
  by (*induct* φ, *auto*)

The circuit-size of a mformula: number of subformulas

definition *cs* :: $'a$ *mformula* ⇒ *nat* **where**
  $cs\ \varphi = card\ (SUB\ \varphi)$

variable assignments

type-synonym $'a$ *VAS* = $'a$ ⇒ *bool*

evaluation of mformulas

fun *eval* :: $'a$ *VAS* ⇒ $'a$ *mformula* ⇒ *bool* **where**
  $eval\ \vartheta\ FALSE = False$
| $eval\ \vartheta\ TRUE = True$
| $eval\ \vartheta\ (Var\ x) = \vartheta\ x$
| $eval\ \vartheta\ (Disj\ \varphi\ \psi) = (eval\ \vartheta\ \varphi \vee eval\ \vartheta\ \psi)$
| $eval\ \vartheta\ (Conj\ \varphi\ \psi) = (eval\ \vartheta\ \varphi \wedge eval\ \vartheta\ \psi)$

lemma *eval-vars*: **assumes** $\bigwedge x.\ x \in vars\ \varphi \Longrightarrow \vartheta 1\ x = \vartheta 2\ x$
  shows $eval\ \vartheta 1\ \varphi = eval\ \vartheta 2\ \varphi$
  using *assms* **by** (*induct* φ, *auto*)

3.2 Conversion of mformulas to true-free mformulas

inductive-set *tf-mformula* :: $'a$ *mformula set* **where**

tf-False: *FALSE* ∈ *tf-mformula*
| *tf-Var*: *Var x* ∈ *tf-mformula*
| *tf-Disj*: φ ∈ *tf-mformula* ⟹ ψ ∈ *tf-mformula* ⟹ *Disj* φ ψ ∈ *tf-mformula*
| *tf-Conj*: φ ∈ *tf-mformula* ⟹ ψ ∈ *tf-mformula* ⟹ *Conj* φ ψ ∈ *tf-mformula*

fun *to-tf-formula* **where**
to-tf-formula (*Disj phi psi*) = (*let phi′* = *to-tf-formula phi*; *psi′* = *to-tf-formula psi*
in (*if phi′* = *TRUE* ∨ *psi′* = *TRUE then TRUE else Disj phi′ psi′*))
| *to-tf-formula* (*Conj phi psi*) = (*let phi′* = *to-tf-formula phi*; *psi′* = *to-tf-formula psi*
in (*if phi′* = *TRUE then psi′ else if psi′* = *TRUE then phi′ else Conj phi′ psi′*))

| *to-tf-formula phi* = *phi*

lemma *eval-to-tf-formula*: *eval* ϑ (*to-tf-formula* φ) = *eval* ϑ φ
by (*induct* φ *rule*: *to-tf-formula.induct*, *auto simp*: *Let-def*)

lemma *to-tf-formula*: *to-tf-formula* φ ≠ *TRUE* ⟹ *to-tf-formula* φ ∈ *tf-mformula*

by (*induct* φ, *auto simp*: *Let-def intro*: *tf-mformula.intros*)

lemma *vars-to-tf-formula*: *vars* (*to-tf-formula* φ) ⊆ *vars* φ
by (*induct* φ *rule*: *to-tf-formula.induct*, *auto simp*: *Let-def*)

lemma *SUB-to-tf-formula*: *SUB* (*to-tf-formula* φ) ⊆ *to-tf-formula* ‘ *SUB* φ
by (*induct* φ *rule*: *to-tf-formula.induct*, *auto simp*: *Let-def*)

lemma *cs-to-tf-formula*: *cs* (*to-tf-formula* φ) ≤ *cs* φ
proof −
have *cs* (*to-tf-formula* φ) ≤ *card* (*to-tf-formula* ‘ *SUB* φ)
unfolding *cs-def* **by** (*rule card-mono*[*OF finite-imageI*[*OF finite-SUB*] *SUB-to-tf-formula*])
also have . . . ≤ *cs* φ **unfolding** *cs-def*
by (*rule card-image-le*[*OF finite-SUB*])
finally show *cs* (*to-tf-formula* φ) ≤ *cs* φ **.**
qed

lemma *to-tf-mformula*: **assumes** ¬ *eval* ϑ φ
shows ∃ ψ ∈ *tf-mformula*. (∀ ϑ. *eval* ϑ φ = *eval* ϑ ψ) ∧ *vars* ψ ⊆ *vars* φ ∧ *cs* ψ ≤ *cs* φ
proof (*intro bexI*[*of - to-tf-formula* φ] *conjI allI eval-to-tf-formula*[*symmetric*] *vars-to-tf-formula to-tf-formula*)
from *assms* **have** ¬ *eval* ϑ (*to-tf-formula* φ) **by** (*simp add*: *eval-to-tf-formula*)
thus *to-tf-formula* φ ≠ *TRUE* **by** *auto*
show *cs* (*to-tf-formula* φ) ≤ *cs* φ **by** (*rule cs-to-tf-formula*)
qed

end

4 Simplied Version of Gordeev's Proof for Monotone Circuits

4.1 Setup of Global Assumptions and Proofs of Approximations

```
theory Assumptions-and-Approximations
imports
  HOL−Real-Asymp.Real-Asymp
  Stirling-Formula.Stirling-Formula
  Preliminaries
begin

locale first-assumptions =
  fixes l p k :: nat
  assumes l2: l > 2
  and pl: p > l
  and kp: k > p
begin

lemma k2: k > 2 using pl l2 kp by auto
lemma p: p > 2 using pl l2 kp by auto
lemma k: k > l using pl l2 kp by auto

definition m = k^4

lemma km: k < m
  using power-strict-increasing-iff[of k 1 4] k2 unfolding m-def by auto

lemma lm: l + 1 < m using km k by simp

lemma m2: m > 2 using k2 km by auto

lemma mp: m > p using km k kp by simp

definition L = fact l * (p − 1) ^ l

lemma kml: k ≤ m − l
proof −
  have k ≤ k * k − k using k2 by (cases k, auto)
  also have ... ≤ (k * k) * 1 − l using k by simp
  also have ... ≤ (k * k) * (k * k) − l
    by (intro diff-le-mono mult-left-mono, insert k2, auto)
  also have (k * k) * (k * k) = m unfolding m-def by algebra
  finally show ?thesis .
qed
end

locale second-assumptions = first-assumptions +
```

assumes *kl2*: $k = l\hat{}2$
and *l8*: $l \geq 8$
begin

lemma *Lm*: $L \geq m$
proof −
have $m \leq l \hat{} l$
unfolding *L-def m-def*
unfolding *kl2 power-mult*[*symmetric*]
by (*intro power-increasing*, *insert l8*, *auto*)
also have $\ldots \leq (p - 1) \hat{} l$
by (*rule power-mono*, *insert pl*, *auto*)
also have $\ldots \leq fact\ l * (p - 1) \hat{} l$ **by** *simp*
also have $\ldots \leq L$ **unfolding** *L-def* **by** *simp*
finally show *?thesis* **.**
qed

lemma *Lp*: $L > p$ **using** *Lm mp* **by** *auto*

lemma *L3*: $L > 3$ **using** *p Lp* **by** *auto*
end

definition $eps = 1/(1000 :: real)$
lemma *eps*: $eps > 0$ **unfolding** *eps-def* **by** *simp*

definition *L0* :: *nat* **where**
$L0 = (SOME\ l0.\ \forall l \geq l0.\ 1\ /\ 3 < (1 + - 1\ /\ real\ l)\ \hat{}\ l)$

definition *M0* :: *nat* **where**
$M0 = (SOME\ y.\ \forall\ x.\ x \geq y \longrightarrow (root\ 8\ (real\ x) * log\ 2\ (real\ x) + 1)\ /\ real\ x\ powr\ (1\ /\ 8 + eps) \leq 1)$

definition *L0'* :: *nat* **where**
$L0' = (SOME\ l0.\ \forall\ n \geq l0.\ 6 * (real\ n)\hat{}16 * fact\ n < real\ (n^2\ \hat{}\ 4)\ powr\ (1\ /\ 8 * real\ (n^2\ \hat{}\ 4)\ powr\ (1\ /\ 8)))$

definition *L0''* :: *nat* **where** $L0'' = (SOME\ l0.\ \forall\ l \geq l0.\ real\ l * log\ 2\ (real\ (l^2\ \hat{}\ 4)) + 1 < real\ (l^2))$

lemma *L0''*: **assumes** $l \geq L0''$ **shows** $real\ l * log\ 2\ (real\ (l^2\ \hat{}\ 4)) + 1 < real\ (l^2)$
proof −
have $(\lambda\ l :: nat.\ (real\ l * log\ 2\ (real\ (l^2\ \hat{}\ 4)) + 1)\ /\ real\ (l^2)) \longlongrightarrow 0$ **by** *real-asymp*
from *LIMSEQ-D*[*OF this*, *of 1*] **obtain** *l0*
where $\forall l \geq l0.\ |1 + real\ l * log\ 2\ (real\ l\ \hat{}\ 8)|\ /\ (real\ l)^2 < 1$ **by** (*auto simp*: *field-simps*)
hence $\forall\ l \geq max\ 1\ l0.\ real\ l * log\ 2\ (real\ (l^2\ \hat{}\ 4)) + 1 < real\ (l^2)$
by (*auto simp*: *field-simps*)

```
  hence ∃ l0. ∀ l ≥ l0. real l ∗ log 2 (real (l^2 ^ 4)) + 1 < real (l^2) by blast
  from someI-ex[OF this, folded L0''-def, rule-format, OF assms]
  show ?thesis .
qed

definition M0' :: nat where
  M0' = (SOME x0. ∀ x ≥ x0. real x powr (2 / 3) ≤ x powr (3 / 4) − 1)

locale third-assumptions = second-assumptions +
  assumes pllog: l ∗ log 2 m ≤ p real p ≤ l ∗ log 2 m + 1
    and L0: l ≥ L0
    and L0': l ≥ L0'
    and M0': m ≥ M0'
    and M0: m ≥ M0
begin


lemma approximation1:
  (real (k − 1)) ^ (m − l) ∗ prod (λ i. real (k − 1 − i)) {0..<l}
   > (real (k − 1)) ^ m / 3
proof −
have real (k − 1) ^ (m − l) ∗ (∏ i = 0..<l. real (k − 1 − i)) =
    real (k − 1) ^ m ∗
    (inverse (real (k − 1)) ^ l ∗ (∏ i = 0..<l. real (k − 1 − i)))
    by (subst power-diff-conv-inverse, insert k2 lm, auto)
  also have … > (real (k − 1)) ^ m ∗ (1/3)
  proof (rule mult-strict-left-mono)
    define f where f l = (1 + (−1) / real l) ^ l for l
    define e1 :: real where e1 = exp (− 1)
    define lim :: real where lim = 1 / 3
    from tendsto-exp-limit-sequentially[of −1, folded f-def]
    have f: f ⟶ e1 by (simp add: e1-def)
    have lim < (1 − 1 / real 6) ^ 6 unfolding lim-def by code-simp
    also have  … ≤ exp (− 1)
      by (rule exp-ge-one-minus-x-over-n-power-n, auto)
    finally have lim < e1 unfolding e1-def by auto
    with f have ∃ l0. ∀ l. l ≥ l0 ⟶ f l > lim
      by (metis eventually-sequentially order-tendstoD(1))
    from someI-ex[OF this[unfolded f-def lim-def], folded L0-def] L0
    have fl: f l > 1/3 unfolding f-def by auto
    define start where start = inverse (real (k − 1)) ^ l ∗ (∏ i = 0..<l. real (k
− 1 − i))
    have uminus start
      = uminus (prod (λ -. inverse (real (k − 1))) {0..<l} ∗ prod (λ i. real (k − 1
− i)) {0 ..< l})
      by (simp add: start-def)
   also have … = uminus (prod (λ i. inverse (real (k − 1)) ∗ real (k − 1 − i))
{0..<l})
      by (subst prod.distrib, simp)
    also have … ≤ uminus (prod (λ i. inverse (real (k − 1)) ∗ real (k − 1 − (l
```

```
− 1))) {0..<l})
      unfolding neg-le-iff-le
    by (intro prod-mono conjI mult-left-mono, insert k2 l2, auto intro!: diff-le-mono2)
    also have ... = uminus ((inverse (real (k − 1)) ∗ real (k − l)) ^ l) by simp
   also have inverse (real (k − 1)) ∗ real (k − l) = inverse (real (k − 1)) ∗ ((real
(k − 1)) − (real l − 1))
      using l2 k2 k by simp
    also have ... = 1 − (real l − 1) / (real (k − 1)) using l2 k2 k
      by (simp add: field-simps)
    also have real (k − 1) = real k − 1 using k2 by simp
    also have ... = (real l − 1) ∗ (real l + 1) unfolding kl2 of-nat-power
      by (simp add: field-simps power2-eq-square)
    also have (real l − 1) / ... = inverse (real l + 1)
     using l2 by (smt (verit, best) divide-divide-eq-left′ divide-inverse nat-1-add-1
nat-less-real-le nonzero-mult-div-cancel-left of-nat-1 of-nat-add)
    also have − ((1 − inverse (real l + 1)) ^ l) ≤ − ((1 − inverse (real l)) ^ l)
      unfolding neg-le-iff-le
      by (intro power-mono, insert l2, auto simp: field-simps)
   also have ... < − (1/3) using fl unfolding f-def by (auto simp: field-simps)
    finally have start: start > 1 / 3 by simp
    thus inverse (real (k − 1)) ^ l ∗ (∏ i = 0..<l. real (k − 1 − i)) > 1/3
      unfolding start-def by simp
  qed (insert k2, auto)
  finally show ?thesis by simp
qed

lemma approximation2: fixes s :: nat
  assumes m choose k ≤ s ∗ L² ∗ (m − l − 1 choose (k − l − 1))
  shows ((m − l) / k)^l / (6 ∗ L^2) < s
proof −
  let ?r = real
  define q where q = (?r (L²) ∗ ?r (m − l − 1 choose (k − l − 1)))
  have q: q > 0 unfolding q-def
    by (insert L3 km, auto)
  have ?r (m choose k) ≤ ?r (s ∗ L² ∗ (m − l − 1 choose (k − l − 1)))
    unfolding of-nat-le-iff using assms by simp
  hence m choose k ≤ s ∗ q unfolding q-def by simp
  hence ∗: s ≥ (m choose k) / q using q by (metis mult-imp-div-pos-le)
  have (((m − l) / k)^l / (L^2)) / 6 < ((m − l) / k)^l / (L^2) / 1
    by (rule divide-strict-left-mono, insert m2 L3 lm k, auto intro!: mult-pos-pos
divide-pos-pos zero-less-power)
  also have ... = ((m − l) / k)^l / (L^2) by simp
  also have ... ≤ ((m choose k) / (m − l − 1 choose (k − l − 1))) / (L^2)
  proof (rule divide-right-mono)
    define b where b = ?r (m − l − 1 choose (k − l − 1))
    define c where c = (?r k)^l
    have b0: b > 0 unfolding b-def using km l2 by simp
    have c0: c > 0 unfolding c-def using k by auto
   define aim where aim = (((m − l) / k)^l ≤ (m choose k) / (m − l − 1 choose
```

$(k - l - 1)))$

have *aim* $\longleftrightarrow ((m - l) / k)$^*l* $\leq$ (*m choose k*) / *b* **unfolding** *b-def aim-def* **by** *simp*

also have ... $\longleftrightarrow b * ((m - l) / k)$^*l* $\leq$ (*m choose k*) **using** *b0*
by (*simp add*: *mult.commute pos-le-divide-eq*)

also have ... $\longleftrightarrow b * (m - l)$^*l* / *c* $\leq$ (*m choose k*)
by (*simp add*: *power-divide c-def*)

also have ... $\longleftrightarrow b * (m - l)$^*l* $\leq$ (*m choose k*) $*$ *c* **using** *c0 b0*
by (*auto simp add*: *mult.commute pos-divide-le-eq*)

also have (*m choose k*) = *fact m* / (*fact k* $*$ *fact* $(m - k)$)
by (*rule binomial-fact*, *insert km*, *auto*)

also have *b* = *fact* $(m - l - 1)$ / (*fact* $(k - l - 1)$ $*$ *fact* $(m - l - 1 - (k - l - 1)))$ **unfolding** *b-def*
by (*rule binomial-fact*, *insert k km*, *auto*)

finally have *aim* $\longleftrightarrow$
fact $(m - l - 1)$ / *fact* $(k - l - 1)$ $*$ $(m - l)$ ^ *l* / *fact* $(m - l - 1 - (k - l - 1))$
$\leq$ (*fact m* / *fact k*) $*$ (*?r k*)^*l* / *fact* $(m - k)$ **unfolding** *c-def* **by** *simp*

also have $m - l - 1 - (k - l - 1) = m - k$ **using** *l2 k km* **by** *simp*

finally have *aim* $\longleftrightarrow$
fact $(m - l - 1)$ / *fact* $(k - l - 1)$ $*$ *?r* $(m - l)$ ^ *l*
$\leq$ *fact m* / *fact k* $*$ *?r k* ^ *l* **unfolding** *divide-le-cancel* **using** *km* **by** *simp*

also have ... $\longleftrightarrow$ (*fact* $(m - (l + 1))$ $*$ *?r* $(m - l)$ ^ *l*) $*$ *fact k*
$\leq$ (*fact m* / *k*) $*$ (*fact* $(k - (l + 1))$ $*$ (*?r k* $*$ *?r k* ^ *l*))
using *k2*
by (*simp add*: *field-simps*)

also have ...

proof (*intro mult-mono*)

have *fact k* $\leq$ *fact* $(k - (l + 1))$ $*$ (*?r k* ^ $(l + 1)$)
by (*rule fact-approx-minus*, *insert k*, *auto*)

also have ... = (*fact* $(k - (l + 1))$ $*$ *?r k* ^ *l*) $*$ *?r k* **by** *simp*

finally show *fact k* $\leq$ *fact* $(k - (l + 1))$ $*$ (*?r k* $*$ *?r k* ^ *l*) **by** (*simp add*: *field-simps*)

have *fact* $(m - (l + 1))$ $*$ *real* $(m - l)$ ^ *l* $\leq$ *fact m* / *k* $\longleftrightarrow$
(*fact* $(m - (l + 1))$ $*$ *?r k*) $*$ *real* $(m - l)$ ^ *l* $\leq$ *fact m* **using** *k2* **by** (*simp add*: *field-simps*)

also have ...

proof −

have (*fact* $(m - (l + 1))$ $*$ *?r k*) $*$ *?r* $(m - l)$ ^ *l* $\leq$
(*fact* $(m - (l + 1))$ $*$ *?r* $(m - l)$) $*$ *?r* $(m - l)$ ^ *l*
by (*intro mult-mono*, *insert kml*, *auto*)

also have ((*fact* $(m - (l + 1))$ $*$ *?r* $(m - l)$) $*$ *?r* $(m - l)$ ^ *l*) =
(*fact* $(m - (l + 1))$ $*$ *?r* $(m - l)$ ^ $(l + 1)$) **by** *simp*

also have ... $\leq$ *fact m*
by (*rule fact-approx-upper-minus*, *insert km k*, *auto*)

finally show *fact* $(m - (l + 1))$ $*$ *real k* $*$ *real* $(m - l)$ ^ *l* $\leq$ *fact m* **.**

qed

finally show *fact* $(m - (l + 1))$ $*$ *real* $(m - l)$ ^ *l* $\leq$ *fact m* / *k* **.**

qed *auto*

```
    finally show ((m − l) / k)^l ≤ (m choose k) / (m − l − 1 choose (k − l −
1))
      unfolding aim-def .
  qed simp
  also have ... = (m choose k) / q
    unfolding q-def by simp
  also have ... ≤ s using q ∗ by metis
  finally show ((m − l) / k)^l / (6 ∗ L^2) < s by simp
qed

lemma approximation3: fixes s :: nat
  assumes (k − 1)^m / 3 < (s ∗ (L^2 ∗ (k − 1) ^ m)) / 2 ^ (p − 1)
  shows ((m − l) / k)^l / (6 ∗ L^2) < s
proof −
  define A where A = real (L^2 ∗ (k − 1) ^ m)
  have A0: A > 0 unfolding A-def using L3 k2 m2 by simp
  from mult-strict-left-mono[OF assms, of 2 ^ (p − 1)]
  have 2^(p − 1) ∗ (k − 1)^m / 3 < s ∗ A
    by (simp add: A-def)
  from divide-strict-right-mono[OF this, of A] A0
  have 2^(p − 1) ∗ (k − 1)^m / 3 / A < s
    by simp
  also have 2^(p − 1) ∗ (k − 1)^m / 3 / A = 2^(p − 1) / (3 ∗ L^2)
    unfolding A-def using k2 by simp
  also have ... = 2^p / (6 ∗ L^2) using p by (cases p, auto)
  also have 2^p = 2 powr p
    by (simp add: powr-realpow)
  finally have ∗: 2 powr p / (6 ∗ L^2) < s .
  have m ^ l = m powr l using m2 l2 powr-realpow by auto
  also have ... = 2 powr (log 2 m ∗ l)
    unfolding powr-powr[symmetric]
    by (subst powr-log-cancel, insert m2, auto)
  also have ... = 2 powr (l ∗ log 2 m) by (simp add: ac-simps)
  also have ... ≤ 2 powr p
    by (rule powr-mono, insert pllog, auto)
  finally have m ^ l ≤ 2 powr p .
  from divide-right-mono[OF this, of 6 ∗ L^2] ∗
  have m ^ l / (6 ∗ L^2) < s by simp
  moreover have ((m − l) / k)^l / (6 ∗ L^2) ≤ m^l / (6 ∗ L^2)
  proof (rule divide-right-mono, unfold of-nat-power, rule power-mono)
    have real (m − l) / real k ≤ real (m − l) / 1
      using k2 lm by (intro divide-left-mono, auto)
    also have ... ≤ m by simp
    finally show (m − l) / k ≤ m by simp
  qed auto
  ultimately show ?thesis by simp
qed

lemma identities: k = root 4 m l = root 8 m
```

```
proof -
  let ?r = real
  have ?r k ^ 4 = ?r m unfolding m-def by simp
  from arg-cong[OF this, of root 4]
  show km-id: k = root 4 m by (simp add: real-root-pos2)
  have ?r l ^ 8 = ?r m unfolding m-def using kl2 by simp
  from arg-cong[OF this, of root 8]
  show lm-id: l = root 8 m by (simp add: real-root-pos2)
qed

lemma identities2: root 4 m = m powr (1/4) root 8 m = m powr (1/8)
  by (subst root-powr-inverse, insert m2, auto)+


lemma appendix-A-1: assumes x ≥ M0' shows x powr (2/3) ≤ x powr (3/4)
- 1
proof -
  have (λ x. x powr (2/3) / (x powr (3/4) - 1)) ⟶ 0
    by real-asymp
  from LIMSEQ-D[OF this, of 1, simplified] obtain x0 :: nat where
    sub: x ≥ x0 ⟹ x powr (2 / 3) / |x powr (3/4) - 1| < 1 for x
    by (auto simp: field-simps)
  have (λ x :: real. 2 / (x powr (3/4))) ⟶ 0
    by real-asymp
  from LIMSEQ-D[OF this, of 1, simplified] obtain x1 :: nat where
    sub2: x ≥ x1 ⟹ 2 / x powr (3 / 4) < 1 for x by auto
  {
    fix x
    assume x: x ≥ x0 x ≥ x1 x ≥ 1
    define a where a = x powr (3/4) - 1
    from sub[OF x(1)] have small: x powr (2 / 3) / |a| ≤ 1
      by (simp add: a-def)
    have 2: 2 ≤ x powr (3/4) using sub2[OF x(2)] x(3) by simp
    hence a: a > 0 by (simp add: a-def)
    from mult-left-mono[OF small, of a] a
    have x powr (2 / 3) ≤ a
      by (simp add: field-simps)
    hence x powr (2 / 3) ≤ x powr (3 / 4) - 1 unfolding a-def by simp
  }
  hence ∃ x0 :: nat. ∀ x ≥ x0. x powr (2 / 3) ≤ x powr (3 / 4) - 1
    by (intro exI[of - max x0 (max x1 1)], auto)
  from someI-ex[OF this, folded M0'-def, rule-format, OF assms]
  show ?thesis .
qed


lemma appendix-A-2: (p - 1)^l < m powr ((1 / 8 + eps) * l)
proof -
  define f where f (x :: nat) = (root 8 x * log 2 x + 1) / (x powr (1/8 + eps))
```

for *x*
have $f \longrightarrow 0$ **using** *eps* **unfolding** *f-def* **by** *real-asymp*
from *LIMSEQ-D*[*OF this*, *of 1*]
have *ex*: $\exists\ x.\ \forall\ y.\ y \geq x \longrightarrow f\ y \leq 1$ **by** *fastforce*
have *lim*: $root\ 8\ m * log\ 2\ m + 1 \leq m\ powr\ (1\ /\ 8 + eps)$
using *someI-ex*[*OF ex*[*unfolded f-def*], *folded M0-def*, *rule-format*, *OF M0*] *m2*
by (*simp add*: *field-simps*)
define *start* **where** $start = real\ (p - 1)\hat{}l$
have $(p - 1)\hat{}l < p\ \hat{}\ l$
by (*rule power-strict-mono*, *insert p l2*, *auto*)
hence $start < real\ (p\ \hat{}\ l)$
using *start-def of-nat-less-of-nat-power-cancel-iff* **by** *blast*
also have $\ldots = p\ powr\ l$
by (*subst powr-realpow*, *insert p*, *auto*)
also have $\ldots \leq (l * log\ 2\ m + 1)\ powr\ l$
by (*rule powr-mono2*, *insert pllog*, *auto*)
also have $l = root\ 8\ m$ **unfolding** *identities* **by** *simp*
finally have $start < (root\ 8\ m * log\ 2\ m + 1)\ powr\ root\ 8\ m$
by (*simp add*: *identities2*)
also have $\ldots \leq (m\ powr\ (1\ /\ 8 + eps))\ powr\ root\ 8\ m$
by (*rule powr-mono2*[*OF - - lim*], *insert m2*, *auto*)
also have $\ldots = m\ powr\ ((1\ /\ 8 + eps) * l)$ **unfolding** *powr-powr identities* **..**
finally show *?thesis* **unfolding** *start-def* **by** *simp*
qed

lemma *appendix-A-3*: $6 * real\ l\hat{}16 * fact\ l < m\ powr\ (1\ /\ 8 * l)$
proof −
define *f* **where** $f = (\lambda n.\ 6 * (real\ n)\hat{}16 * (sqrt\ (2 * pi * real\ n) * (real\ n\ /\ exp\ 1)\ \hat{}\ n))$
define *g* **where** $g = (\lambda\ n.\ 6 * (real\ n)\hat{}16 * (sqrt\ (2 * 4 * real\ n) * (real\ n\ /\ 2)\ \hat{}\ n))$
define *h* **where** $h = (\lambda\ n.\ ((real\ (n^2\ \hat{}\ 4)\ powr\ (1\ /\ 8 * (real\ (n^2\ \hat{}\ 4))\ powr\ (1/8)))))$
have *e*: $2 \leq (exp\ 1 :: real)$ **using** *exp-ge-add-one-self*[*of 1*] **by** *simp*
from *fact-asymp-equiv*
have *1*: $(\lambda\ n.\ 6 * (real\ n)\hat{}16 * fact\ n\ /\ h\ n) \sim[sequentially]\ (\lambda\ n.\ f\ n\ /\ h\ n)$ **unfolding** *f-def*
by (*intro asymp-equiv-intros*)
have *2*: $f\ n \leq g\ n$ **for** *n* **unfolding** *f-def g-def*
by (*intro mult-mono power-mono divide-left-mono real-sqrt-le-mono*, *insert pi-less-4 e*, *auto*)
have *2*: $abs\ (f\ n\ /\ h\ n) \leq abs\ (g\ n\ /\ h\ n)$ **for** *n*
unfolding *abs-le-square-iff power2-eq-square*
by (*intro mult-mono divide-right-mono 2*, *auto simp*: *h-def f-def g-def*)
have *2*: $abs\ (g\ n\ /\ h\ n) < e \Longrightarrow abs\ (f\ n\ /\ h\ n) < e$ **for** *n e* **using** *2*[*of n*] **by** *simp*
have $(\lambda n.\ g\ n\ /\ h\ n) \longrightarrow 0$
unfolding *g-def h-def* **by** *real-asymp*
from *LIMSEQ-D*[*OF this*] *2*

have ($\lambda n.\ f\ n\ /\ h\ n$) $\longrightarrow$ *0*
by (*intro LIMSEQ-I*, *fastforce*)
with *1* **have** ($\lambda n.\ 6 * (real\ n)\hat{}16 * fact\ n\ /\ h\ n$) $\longrightarrow$ *0*
using *tendsto-asymp-equiv-cong* **by** *blast*
from *LIMSEQ-D*[*OF this*, *of 1*] **obtain** *n0* **where** *3*: $n \geq n0 \Longrightarrow norm\ (6 * (real\ n)\hat{}16 * fact\ n\ /\ h\ n) < 1$ **for** *n* **by** *auto*
{
fix *n*
assume *n*: $n \geq max\ 1\ n0$
hence *hn*: $h\ n > 0$ **unfolding** *h-def* **by** *auto*
from *n* **have** $n \geq n0$ **by** *simp*
from *3*[*OF this*] **have** $6 * n\ \hat{}\ 16 * fact\ n\ /\ abs\ (h\ n) < 1$ **by** *auto*
with *hn* **have** $6 * (real\ n)\ \hat{}\ 16 * fact\ n < h\ n$ **by** *simp*
}
hence $\exists\ n0.\ \forall\ n.\ n \geq n0 \longrightarrow 6 * n\ \hat{}\ 16 * fact\ n < h\ n$ **by** *blast*
from *someI-ex*[*OF this*[*unfolded h-def*], *folded L0'-def*, *rule-format*, *OF L0'*]
have $6 * real\ l\hat{}16 * fact\ l < real\ (l^2\ \hat{}\ 4)\ powr\ (1\ /\ 8 * real\ (l^2\ \hat{}\ 4)\ powr\ (1\ /\ 8))$ **by** *simp*
also have ... $= m\ powr\ (1\ /\ 8 * l)$ **using** *identities identities2 kl2*
by (*metis m-def*)
finally show *?thesis* **.**
qed

lemma *appendix-A-4*: $12 * L\hat{}2 \leq m\ powr\ (m\ powr\ (1\ /\ 8) * 0.51)$
proof −
let $?r = real$
define *Lappr* **where** $Lappr = m * m * fact\ l * p\ \hat{}\ l\ /\ 2$
have $L = (fact\ l * (p - 1)\ \hat{}\ l)$ **unfolding** *L-def* **by** *simp*
hence $?r\ L \leq (fact\ l * (p - 1)\ \hat{}\ l)$ **by** *linarith*
also have ... $= (1 * ?r\ (fact\ l)) * (?r\ (p - 1)\ \hat{}\ l)$ **by** *simp*
also have ... $\leq ((m * m\ /\ 2) * ?r\ (fact\ l)) * (?r\ (p - 1)\ \hat{}\ l)$
by (*intro mult-right-mono*, *insert m2*, *cases m*; *cases m − 1*, *auto*)
also have ... $= (6 * real\ (m * m) * fact\ l) * (?r\ (p - 1)\ \hat{}\ l)\ /\ 12$ **by** *simp*
also have $real\ (m * m) = real\ l\hat{}16$ **unfolding** *m-def* **unfolding** *kl2* **by** *simp*
also have $(6 * real\ l\hat{}16 * fact\ l) * (?r\ (p - 1)\ \hat{}\ l)\ /\ 12$
$\leq (m\ powr\ (1\ /\ 8 * l) * (m\ powr\ ((1\ /\ 8 + eps) * l)))\ /\ 12$
by (*intro divide-right-mono mult-mono*, *insert appendix-A-2 appendix-A-3*, *auto*)

also have ... $= (m\ powr\ (1\ /\ 8 * l + (1\ /\ 8 + eps) * l))\ /\ 12$
by (*simp add*: *powr-add*)
also have $1\ /\ 8 * l + (1\ /\ 8 + eps) * l = l * (1/4 + eps)$ **by** (*simp add*: *field-simps*)
also have $l = m\ powr\ (1/8)$ **unfolding** *identities identities2* **..**
finally have *LL*: $?r\ L \leq m\ powr\ (m\ powr\ (1\ /\ 8) * (1\ /\ 4 + eps))\ /\ 12$ **.**
from *power-mono*[*OF this*, *of 2*]
have $L\hat{}2 \leq (m\ powr\ (m\ powr\ (1\ /\ 8) * (1\ /\ 4 + eps))\ /\ 12)\hat{}2$
by *simp*
also have ... $= (m\ powr\ (m\ powr\ (1\ /\ 8) * (1\ /\ 4 + eps)))\hat{}2\ /\ 144$
by (*simp add*: *power2-eq-square*)

```
  also have ... = (m powr (m powr (1 / 8) * (1 / 4 + eps) * 2)) / 144
    by (subst powr-realpow[symmetric], (use m2 in force), unfold powr-powr, simp)
  also have ... = (m powr (m powr (1 / 8) * (1 / 2 + 2 * eps))) / 144
    by (simp add: algebra-simps)
  also have ... ≤ (m powr (m powr (1 / 8) * 0.51)) / 144
    by (intro divide-right-mono powr-mono mult-left-mono, insert m2, auto simp:
eps-def)
  finally have L^2 ≤ m powr (m powr (1 / 8) * 0.51) / 144 by simp
  from mult-left-mono[OF this, of 12]
  have 12 * L^2 ≤ 12 * m powr (m powr (1 / 8) * 0.51) / 144 by simp
  also have ... = m powr (m powr (1 / 8) * 0.51) / 12 by simp
  also have ... ≤ m powr (m powr (1 / 8) * 0.51) / 1
    by (rule divide-left-mono, auto)
  finally show ?thesis by simp
qed

lemma approximation4: fixes s :: nat
  assumes s > ((m − l) / k)^l / (6 * L^2)
  shows s > 2 * k powr (4 / 7 * sqrt k)
proof −
  let ?r = real
  have diff: ?r (m − l) = ?r m − ?r l using lm by simp
  have m powr (2/3) ≤ m powr (3/4) − 1 using appendix-A-1[OF M0'] by auto
  also have ... ≤ (m − m powr (1/8)) / m powr (1/4)
    unfolding diff-divide-distrib
    by (rule diff-mono, insert m2, auto simp: divide-powr-uminus powr-mult-base
powr-add[symmetric],
      auto simp: powr-minus-divide intro!: ge-one-powr-ge-zero)
  also have ... = (m − root 8 m) / root 4 m using m2
    by (simp add: root-powr-inverse)
  also have ... = (m − l) / k unfolding identities diff by simp
  finally have m powr (2/3) ≤ (m − l) / k by simp
  from power-mono[OF this, of l]
  have ineq1: (m powr (2 / 3)) ^ l ≤ ((m − l) / k) ^ l  using m2 by auto
  have (m powr (l / 7)) ≤ (m powr (2 / 3 * l − l * 0.51))
    by (intro powr-mono, insert m2, auto)
  also have ... = (m powr (2 / 3)) powr l / (m powr (m powr (1 / 8) * 0.51))
    unfolding powr-diff powr-powr identities identities2 by simp
  also have ... = (m powr (2 / 3)) ^ l / (m powr (m powr (1 / 8) * 0.51))
    by (subst powr-realpow, insert m2, auto)
  also have ... ≤ (m powr (2 / 3)) ^ l / (12 * L²)
   by (rule divide-left-mono[OF appendix-A-4], insert L3 m2, auto intro!: mult-pos-pos)
  also have ... = (m powr (2 / 3)) ^ l / (?r 12 * L²) by simp
  also have ... ≤ ((m − l) / k) ^ l / (?r 12 * L²)
    by (rule divide-right-mono[OF ineq1], insert L3, auto)
  also have ... < s / 2 using assms by simp
  finally have 2 * m powr (real l / 7) < s by simp
  also have m powr (real l / 7) = m powr (root 8 m / 7)
    unfolding identities by simp
```

finally have *s > 2 ∗ m powr (root 8 m / 7)* **by** *simp*
also have *root 8 m = root 2 k* **using** *m2*
by (*metis identities(2) kl2 of-nat-0-le-iff of-nat-power pos2 real-root-power-cancel*)
also have *?r m = k powr 4* **unfolding** *m-def* **by** *simp*
also have (*k powr 4*) *powr* ((*root 2 k*) / *7*)
= *k powr* (*4 ∗* (*root 2 k*) / *7*) **unfolding** *powr-powr* **by** *simp*
also have ... = *k powr* (*4 / 7 ∗ sqrt k*) **unfolding** *sqrt-def* **by** *simp*
finally show *s > 2 ∗ k powr* (*4 / 7 ∗ sqrt k*) **.**
qed

end

end
theory *Clique-Large-Monotone-Circuits*
imports
Sunflowers.Erdos-Rado-Sunflower
Preliminaries
Assumptions-and-Approximations
Monotone-Formula
begin

disable list-syntax

unbundle *no list-enumeration-syntax*
and *no list-comprehension-syntax*

hide-const (**open**) *Sigma-Algebra.measure*

4.2 Plain Graphs

definition *binprod* :: *'a set ⇒ 'a set ⇒ 'a set set* (**infixl** ‹·› *60*) **where**
$X \cdot Y = \{\{x,y\} \mid x\ y.\ x \in X \wedge y \in Y \wedge x \neq y\}$

abbreviation *sameprod* :: *'a set ⇒ 'a set set* (‹(-)^**2**›) **where**
X^**2** ≡ $X \cdot X$

lemma *sameprod-altdef*: X^**2** = $\{Y.\ Y \subseteq X \wedge card\ Y = 2\}$
unfolding *binprod-def* **by** (*auto simp*: *card-2-iff*)

definition *numbers* :: *nat ⇒ nat set* (‹[(-)]›) **where**
$[n] \equiv \{..<n\}$

lemma *card-sameprod*: *finite X ⟹ card* (*X*^**2**) = *card X choose 2*
unfolding *sameprod-altdef*
by (*subst n-subsets*, *auto*)

lemma *sameprod-mono*: *X ⊆ Y ⟹ X*^**2** ⊆ *Y*^**2**
unfolding *sameprod-altdef* **by** *auto*

lemma *sameprod-finite*: *finite X ⟹ finite* (*X*^**2**)

unfolding *sameprod-altdef* **by** *simp*

lemma *numbers2-mono*: $x \le y \Longrightarrow [x]\hat{}\mathbf{2} \subseteq [y]\hat{}\mathbf{2}$
by (*rule sameprod-mono*, *auto simp*: *numbers-def*)

lemma *card-numbers*[*simp*]: *card* $[n] = n$
by (*simp add*: *numbers-def*)

lemma *card-numbers2*[*simp*]: *card* $([n]\hat{}\mathbf{2}) = n$ *choose 2*
by (*subst card-sameprod*, *auto simp*: *numbers-def*)

type-synonym *vertex* = *nat*
type-synonym *graph* = *vertex set set*

definition *Graphs* :: *vertex set* ⇒ *graph set* **where**
Graphs V = { *G*. $G \subseteq V\hat{}\mathbf{2}$ }

definition *Clique* :: *vertex set* ⇒ *nat* ⇒ *graph set* **where**
Clique V k = { *G*. *G* ∈ *Graphs V* ∧ (∃ *C* ⊆ *V*. $C\hat{}\mathbf{2} \subseteq G$ ∧ *card C* = *k*) }

context *first-assumptions*
begin

abbreviation $\mathcal{G}$ **where** $\mathcal{G} \equiv$ *Graphs* [*m*]

lemmas $\mathcal{G}$*-def* = *Graphs-def*[*of* [*m*]]

lemma *empty-*$\mathcal{G}$[*simp*]: {} ∈ $\mathcal{G}$ **unfolding** $\mathcal{G}$*-def* **by** *auto*

definition *v* :: *graph* ⇒ *vertex set* **where**
v G = { *x* . ∃ *y*. {*x*,*y*} ∈ *G*}

lemma *v-union*: *v* (*G* ∪ *H*) = *v G* ∪ *v H*
unfolding *v-def* **by** *auto*

definition $\mathcal{K}$ :: *graph set* **where**
$\mathcal{K}$ = { *K* . *K* ∈ $\mathcal{G}$ ∧ *card* (*v K*) = *k* ∧ *K* = $(v\ K)\hat{}\mathbf{2}$ }

lemma *v-*$\mathcal{G}$: *G* ∈ $\mathcal{G}$ ⟹ *v G* ⊆ [*m*]
unfolding *v-def* $\mathcal{G}$*-def sameprod-altdef* **by** *auto*

lemma *v-mono*: *G* ⊆ *H* ⟹ *v G* ⊆ *v H* **unfolding** *v-def* **by** *auto*

lemma *v-sameprod*[*simp*]: **assumes** *card X* ≥ *2*
shows $v\ (X\hat{}\mathbf{2}) = X$
proof −
from *obtain-subset-with-card-n*[*OF assms*] **obtain** *Y* **where** *Y* ⊆ *X*
and *Y*: *card Y* = *2* **by** *auto*

then obtain *x y* **where** $x \in X$ $y \in X$ **and** $x \neq y$
by (*auto simp*: *card-2-iff*)
thus *?thesis* **unfolding** *sameprod-altdef v-def*
by (*auto simp*: *card-2-iff doubleton-eq-iff*) *blast*
qed

lemma *v-mem-sub*: **assumes** *card* $e = 2$ $e \in G$ **shows** $e \subseteq v\ G$
proof −
obtain *x y* **where** *e*: $e = \{x,y\}$ **and** *xy*: $x \neq y$ **using** *assms*
by (*auto simp*: *card-2-iff*)
from *assms*(*2*) **have** *x*: $x \in v\ G$ **unfolding** *e*
by (*auto simp*: *v-def*)
from *e* **have** *e*: $e = \{y,x\}$ **unfolding** *e* **by** *auto*
from *assms*(*2*) **have** *y*: $y \in v\ G$ **unfolding** *e*
by (*auto simp*: *v-def*)
show $e \subseteq v\ G$ **using** *x y* **unfolding** *e* **by** *auto*
qed

lemma *v-𝒢-2*: **assumes** $G \in \mathcal{G}$ **shows** $G \subseteq (v\ G)$^**2**
proof
fix *e*
assume *eG*: $e \in G$
with *assms*[*unfolded 𝒢-def binprod-def*] **obtain** *x y* **where** *e*: $e = \{x,y\}$ **and** *xy*: $x \neq y$ **by** *auto*
from *eG e xy* **have** *x*: $x \in v\ G$ **by** (*auto simp*: *v-def*)
from *e* **have** *e*: $e = \{y,x\}$ **unfolding** *e* **by** *auto*
from *eG e xy* **have** *y*: $y \in v\ G$ **by** (*auto simp*: *v-def*)
from *x y xy* **show** $e \in (v\ G)$^**2** **unfolding** *binprod-def e* **by** *auto*
qed

lemma *v-numbers2*[*simp*]: $x \geq 2 \Longrightarrow v\ ([x]^{**2**}) = [x]$
by (*rule v-sameprod*, *auto*)

lemma *sameprod-𝒢*: **assumes** $X \subseteq [m]$ *card* $X \geq 2$
shows X^**2** $\in \mathcal{G}$
unfolding *𝒢-def* **using** *assms*(*2*) *sameprod-mono*[*OF assms*(*1*)]
by *auto*

lemma *finite-numbers*[*simp*,*intro*]: *finite* [*n*]
unfolding *numbers-def* **by** *auto*

lemma *finite-numbers2*[*simp*,*intro*]: *finite* ([*n*]^**2**)
unfolding *sameprod-altdef* **using** *finite-subset*[*of* - [*m*]] **by** *auto*

lemma *finite-members-𝒢*: $G \in \mathcal{G} \Longrightarrow$ *finite G*
unfolding *𝒢-def* **using** *finite-subset*[*of G* [*m*]^**2**] **by** *auto*

lemma *finite-𝒢*[*simp*,*intro*]: *finite* 𝒢

unfolding $\mathcal{G}$*-def* **by** *simp*

lemma *finite-vG*: **assumes** $G \in \mathcal{G}$
shows *finite* (*v G*)
proof −
from *finite-members-*$\mathcal{G}$[*OF assms*]
show *?thesis*
proof (*induct rule*: *finite-induct*)
case (*insert xy F*)
show *?case*
proof (*cases* ∃ *x y*. *xy* = {*x*,*y*})
case *False*
hence *v* (*insert xy F*) = *v F* **unfolding** *v-def* **by** *auto*
thus *?thesis* **using** *insert* **by** *auto*
next
case *True*
then obtain *x y* **where** *xy*: *xy* = {*x*,*y*} **by** *auto*
hence *v* (*insert xy F*) = *insert x* (*insert y* (*v F*))
unfolding *v-def* **by** *auto*
thus *?thesis* **using** *insert* **by** *auto*
qed
qed (*auto simp*: *v-def*)
qed

lemma *v-empty*[*simp*]: *v* {} = {} **unfolding** *v-def* **by** *auto*

lemma *v-card2*: **assumes** $G \in \mathcal{G}$ $G \neq \{\}$
shows *2* ≤ *card* (*v G*)
proof −
from *assms*[*unfolded* $\mathcal{G}$*-def*] **obtain** *edge* **where** ∗: *edge* ∈ *G edge* ∈ [*m*]**^2** **by** *auto*
then obtain *x y* **where** *edge*: *edge* = {*x*,*y*} *x* ≠ *y* **unfolding** *binprod-def* **by** *auto*
with ∗ **have** *sub*: {*x*,*y*} ⊆ *v G* **unfolding** *v-def*
by (*smt* (*verit*, *best*) *insert-commute insert-compr mem-Collect-eq singleton-iff subsetI*)
from *assms finite-vG* **have** *finite* (*v G*) **by** *auto*
from *sub* ‹*x* ≠ *y*› *this* **show** *2* ≤ *card* (*v G*)
by (*metis card-2-iff card-mono*)
qed

lemma $\mathcal{K}$*-altdef*: $\mathcal{K}$ = {*V*\^**2** | *V*. *V* ⊆ [*m*] ∧ *card V* = *k*}
(**is** - = *?R*)
proof −
{
fix *K*
assume $K \in \mathcal{K}$
hence *K*: $K \in \mathcal{G}$ **and** *card*: *card* (*v K*) = *k* **and** *KvK*: *K* = (*v K*)\^**2**

unfolding $\mathcal{K}$*-def* **by** *auto*
from *v-*$\mathcal{G}$[*OF K*] *card KvK* **have** $K \in$ *?R* **by** *auto*
}
moreover
{
fix *V*
assume *1*: $V \subseteq [m]$ **and** *card* $V = k$
hence V^**2** $\in \mathcal{K}$ **unfolding** $\mathcal{K}$*-def* **using** *k2 sameprod-*$\mathcal{G}$[*OF 1*]
by *auto*
}
ultimately show *?thesis* **by** *auto*
qed

lemma $\mathcal{K}$*-*$\mathcal{G}$: $\mathcal{K} \subseteq \mathcal{G}$
unfolding $\mathcal{K}$*-def* **by** *auto*

definition *CLIQUE* :: *graph set* **where**
CLIQUE = { *G*. $G \in \mathcal{G} \wedge (\exists\ K \in \mathcal{K}.\ K \subseteq G)$ }

lemma *empty-CLIQUE*[*simp*]: {} $\notin$ *CLIQUE* **unfolding** *CLIQUE-def* $\mathcal{K}$*-def* **using** *k2* **by** (*auto simp*: *v-def*)

4.3 Test Graphs

Positive test graphs are precisely the cliques of size *k*.

abbreviation *POS* $\equiv \mathcal{K}$

lemma *POS-*$\mathcal{G}$: *POS* $\subseteq \mathcal{G}$ **by** (*rule* $\mathcal{K}$*-*$\mathcal{G}$)

Negative tests are coloring-functions of vertices that encode graphs which have cliques of size at most $k - 1$.

type-synonym *colorf* = *vertex* $\Rightarrow$ *nat*

definition $\mathcal{F}$ :: *colorf set* **where**
$\mathcal{F} = [m] \rightarrow_E [k - 1]$

lemma *finite-*$\mathcal{F}$: *finite* $\mathcal{F}$
unfolding $\mathcal{F}$*-def numbers-def*
by (*meson finite-PiE finite-lessThan*)

definition *C* :: *colorf* $\Rightarrow$ *graph* **where**
$C\,f = \{\ \{x,\ y\} \mid x\ y\ .\ \{x,y\} \in [m]$^**2** $\wedge f\,x \neq f\,y\}$

definition *NEG* :: *graph set* **where**
NEG = *C* ‘ $\mathcal{F}$

Lemma 1 **lemma** *CLIQUE-NEG*: *CLIQUE* $\cap$ *NEG* = {}
proof –

```
{
  fix G
  assume GC: G ∈ CLIQUE and GN: G ∈ NEG
  from GC[unfolded CLIQUE-def] obtain K where
    K: K ∈ 𝒦 and G: G ∈ 𝒢 and KsubG: K ⊆ G by auto
  from GN[unfolded NEG-def] obtain f where fF: f ∈ ℱ and
    GCf: G = C f by auto
  from K[unfolded 𝒦-def] have KG: K ∈ 𝒢 and
    KvK: K = v K^2 and card1: card (v K) = k by auto
  from k2 card1 have ineq: card (v K) > card [k − 1] by auto
  from v-𝒢[OF KG] have vKm: v K ⊆ [m] by auto
  from fF[unfolded ℱ-def] vKm have f: f ∈ v K → [k − 1]
    by auto
  from card-inj[OF f] ineq
  have ¬ inj-on f (v K) by auto
  then obtain x y where ∗: x ∈ v K y ∈ v K x ≠ y and ineq: f x = f y
    unfolding inj-on-def by auto
  have {x,y} ∉ G unfolding GCf C-def using ineq
    by (auto simp: doubleton-eq-iff)
  with KsubG KvK have {x,y} ∉ v K^2 by auto
  with ∗ have False unfolding binprod-def by auto
}
thus ?thesis by auto
qed

lemma NEG-𝒢: NEG ⊆ 𝒢
proof −
  {
    fix f
    assume f ∈ ℱ
    hence C f ∈ 𝒢
      unfolding NEG-def C-def 𝒢-def
      by (auto simp: sameprod-altdef)
  }
  thus NEG ⊆ 𝒢 unfolding NEG-def by auto
qed

lemma finite-POS-NEG: finite (POS ∪ NEG)
  using POS-𝒢 NEG-𝒢
  by (intro finite-subset[OF - finite-𝒢], auto)

lemma POS-sub-CLIQUE: POS ⊆ CLIQUE
  unfolding CLIQUE-def using 𝒦-𝒢 by auto

lemma POS-CLIQUE: POS ⊂ CLIQUE
proof −
  have [k+1]^2 ∈ CLIQUE
    unfolding CLIQUE-def
  proof (standard, intro conjI bexI[of - [k]^2])
```

```
    show [k]^2 ⊆ [k+1]^2
      by (rule numbers2-mono, auto)
    show [k]^2 ∈ 𝒦 unfolding 𝒦-altdef using km
      by (auto intro!: exI[of - [k]], auto simp: numbers-def)
    show [k+1]^2 ∈ 𝒢 using km k2
      by (intro sameprod-𝒢, auto simp: numbers-def)
  qed
  moreover have [k+1]^2 ∉ POS unfolding 𝒦-def using v-numbers2[of k + 1]
k2
    by auto
  ultimately show ?thesis using POS-sub-CLIQUE by blast
qed

lemma card-POS: card POS = m choose k
proof −
  have m choose k =
    card {B. B ⊆ [m] ∧ card B = k} (is - = card ?A)
    by (subst n-subsets[of [m] k], auto simp: numbers-def)
  also have ... = card (sameprod ` ?A)
  proof (rule card-image[symmetric])
    {
      fix A
      assume A ∈ ?A
      hence v (sameprod A) = A using k2
        by (subst v-sameprod, auto)
    }
    thus inj-on sameprod ?A by (rule inj-on-inverseI)
  qed
  also have sameprod ` {B. B ⊆ [m] ∧ card B = k} = POS
    unfolding 𝒦-altdef by auto
  finally show ?thesis by simp
qed
```

4.4 Basic operations on sets of graphs

```
definition odot :: graph set ⇒ graph set ⇒ graph set (infixl ‹⊙› 65) where
  X ⊙ Y = { D ∪ E | D E. D ∈ X ∧ E ∈ Y}

lemma union-𝒢[intro]: G ∈ 𝒢 ⟹ H ∈ 𝒢 ⟹ G ∪ H ∈ 𝒢
  unfolding 𝒢-def by auto

lemma odot-𝒢: X ⊆ 𝒢 ⟹ Y ⊆ 𝒢 ⟹ X ⊙ Y ⊆ 𝒢
  unfolding odot-def by auto
```

4.5 Acceptability

Definition 2

```
definition accepts :: graph set ⇒ graph ⇒ bool (infixl ‹⊩› 55) where
  (X ⊩ G) = (∃ D ∈ X. D ⊆ G)
```

lemma *acceptsI*[*intro*]: $D \subseteq G \Longrightarrow D \in X \Longrightarrow X \Vdash G$
  unfolding *accepts-def* **by** *auto*

definition *ACC* :: *graph set* ⇒ *graph set* **where**
  $ACC\ X = \{\ G.\ G \in \mathcal{G} \wedge X \Vdash G\}$

definition *ACC-cf* :: *graph set* ⇒ *colorf set* **where**
  $ACC\text{-}cf\ X = \{\ F.\ F \in \mathcal{F} \wedge X \Vdash C\ F\}$

lemma *ACC-cf-*$\mathcal{F}$: $ACC\text{-}cf\ X \subseteq \mathcal{F}$
  unfolding *ACC-cf-def* **by** *auto*

lemma *finite-ACC*[*intro*,*simp*]: *finite* (*ACC-cf X*)
  by (*rule finite-subset*[*OF ACC-cf-*$\mathcal{F}$ *finite-*$\mathcal{F}$])

lemma *ACC-I*[*intro*]: $G \in \mathcal{G} \Longrightarrow X \Vdash G \Longrightarrow G \in ACC\ X$
  unfolding *ACC-def* **by** *auto*

lemma *ACC-cf-I*[*intro*]: $F \in \mathcal{F} \Longrightarrow X \Vdash C\ F \Longrightarrow F \in ACC\text{-}cf\ X$
  unfolding *ACC-cf-def* **by** *auto*

lemma *ACC-cf-mono*: $X \subseteq Y \Longrightarrow ACC\text{-}cf\ X \subseteq ACC\text{-}cf\ Y$
  unfolding *ACC-cf-def accepts-def* **by** *auto*

Lemma 3

lemma *ACC-cf-empty*: $ACC\text{-}cf\ \{\} = \{\}$
  unfolding *ACC-cf-def accepts-def* **by** *auto*

lemma *ACC-empty*[*simp*]: $ACC\ \{\} = \{\}$
  unfolding *ACC-def accepts-def* **by** *auto*

lemma *ACC-cf-union*: $ACC\text{-}cf\ (X \cup Y) = ACC\text{-}cf\ X \cup ACC\text{-}cf\ Y$
  unfolding *ACC-cf-def accepts-def* **by** *blast*

lemma *ACC-union*: $ACC\ (X \cup Y) = ACC\ X \cup ACC\ Y$
  unfolding *ACC-def accepts-def* **by** *blast*

lemma *ACC-odot*: $ACC\ (X \odot Y) = ACC\ X \cap ACC\ Y$
proof −
  {
    fix G
    assume $G \in ACC\ (X \odot Y)$
    from *this*[*unfolded ACC-def accepts-def*]
    obtain $D\ E\ F$ :: *graph* **where** $*$: $D \in X\ E \in Y\ G \in \mathcal{G}\ D \cup E \subseteq G$
      by (*force simp*: *odot-def*)
    hence $G \in ACC\ X \cap ACC\ Y$
      unfolding *ACC-def accepts-def* **by** *auto*

}
moreover
{
fix *G*
assume $G \in ACC\ X \cap ACC\ Y$
from *this*[*unfolded ACC-def accepts-def*]
obtain *D E* **where** ∗: $D \in X\ E \in Y\ G \in \mathcal{G}\ D \subseteq G\ E \subseteq G$
by *auto*
let $?F = D \cup E$
from ∗ **have** $?F \in X \odot Y$ **unfolding** *odot-def* **using** ∗ **by** *blast*
moreover have $?F \subseteq G$ **using** ∗ **by** *auto*
ultimately have $G \in ACC\ (X \odot Y)$ **using** ∗
unfolding *ACC-def accepts-def* **by** *blast*
}
ultimately show *?thesis* **by** *blast*
qed

lemma *ACC-cf-odot*: $ACC\text{-}cf\ (X \odot Y) = ACC\text{-}cf\ X \cap ACC\text{-}cf\ Y$
proof −
{
fix *G*
assume $G \in ACC\text{-}cf\ (X \odot Y)$
from *this*[*unfolded ACC-cf-def accepts-def*]
obtain *D E* :: *graph* **where** ∗: $D \in X\ E \in Y\ G \in \mathcal{F}\ D \cup E \subseteq C\ G$
by (*force simp*: *odot-def*)
hence $G \in ACC\text{-}cf\ X \cap ACC\text{-}cf\ Y$
unfolding *ACC-cf-def accepts-def* **by** *auto*
}
moreover
{
fix *F*
assume $F \in ACC\text{-}cf\ X \cap ACC\text{-}cf\ Y$
from *this*[*unfolded ACC-cf-def accepts-def*]
obtain *D E* **where** ∗: $D \in X\ E \in Y\ F \in \mathcal{F}\ D \subseteq C\ F\ E \subseteq C\ F$
by *auto*
let $?F = D \cup E$
from ∗ **have** $?F \in X \odot Y$ **unfolding** *odot-def* **using** ∗ **by** *blast*
moreover have $?F \subseteq C\ F$ **using** ∗ **by** *auto*
ultimately have $F \in ACC\text{-}cf\ (X \odot Y)$ **using** ∗
unfolding *ACC-cf-def accepts-def* **by** *blast*
}
ultimately show *?thesis* **by** *blast*
qed

4.6 Approximations and deviations

definition $\mathcal{G}l$ :: *graph set* **where**
$\mathcal{G}l = \{ G.\ G \in \mathcal{G} \wedge card\ (v\ G) \leq l \}$

definition *v-gs* :: *graph set* ⇒ *vertex set set* **where**
v-gs X = *v* ‘ *X*

lemma *v-gs-empty*[*simp*]: *v-gs* {} = {}
unfolding *v-gs-def* **by** *auto*

lemma *v-gs-union*: *v-gs* (*X* ∪ *Y*) = *v-gs X* ∪ *v-gs Y*
unfolding *v-gs-def* **by** *auto*

lemma *v-gs-mono*: *X* ⊆ *Y* ⟹ *v-gs X* ⊆ *v-gs Y*
using *v-gs-def* **by** *auto*

lemma *finite-v-gs*: **assumes** *X* ⊆ 𝒢
shows *finite* (*v-gs X*)
proof −
have *v-gs X* ⊆ *v* ‘ 𝒢
using *assms* **unfolding** *v-gs-def* **by** *force*
moreover have *finite* 𝒢 **using** *finite-*𝒢 **by** *auto*
ultimately show *?thesis* **by** (*metis finite-surj*)
qed

lemma *finite-v-gs-Gl*: **assumes** *X* ⊆ 𝒢*l*
shows *finite* (*v-gs X*)
by (*rule finite-v-gs*, *insert assms*, *auto simp*: 𝒢*l-def*)

definition 𝒫*L*𝒢*l* :: *graph set set* **where**
𝒫*L*𝒢*l* = { *X* . *X* ⊆ 𝒢*l* ∧ *card* (*v-gs X*) ≤ *L*}

definition *odotl* :: *graph set* ⇒ *graph set* ⇒ *graph set* (**infixl** ‹⊙*l*› *65*) **where**
X ⊙*l Y* = (*X* ⊙ *Y*) ∩ 𝒢*l*

lemma *joinl-join*: *X* ⊙*l Y* ⊆ *X* ⊙ *Y*
unfolding *odot-def odotl-def* **by** *blast*

lemma *card-v-gs-join*: **assumes** *X*: *X* ⊆ 𝒢 **and** *Y*: *Y* ⊆ 𝒢
and *Z*: *Z* ⊆ *X* ⊙ *Y*
shows *card* (*v-gs Z*) ≤ *card* (*v-gs X*) ∗ *card* (*v-gs Y*)
proof −
note *fin* = *finite-v-gs*[*OF X*] *finite-v-gs*[*OF Y*]
have *card* (*v-gs Z*) ≤ *card* ((λ (*A*, *B*). *A* ∪ *B*) ‘ (*v-gs X* × *v-gs Y*))
proof (*rule card-mono*[*OF finite-imageI*])
show *finite* (*v-gs X* × *v-gs Y*)
using *fin* **by** *auto*
have *v-gs Z* ⊆ *v-gs* (*X* ⊙ *Y*)
using *v-gs-mono*[*OF Z*] **.**
also have . . . ⊆ (λ(*x*, *y*). *x* ∪ *y*) ‘ (*v-gs X* × *v-gs Y*) (**is** *?L* ⊆ *?R*)
unfolding *odot-def v-gs-def* **by** (*force split*: *if-splits simp*: *v-union*)

```
    finally show v-gs Z ⊆ (λ(x, y). x ∪ y) ‘ (v-gs X × v-gs Y) .
  qed
  also have ... ≤ card (v-gs X × v-gs Y)
    by (rule card-image-le, insert fin, auto)
  also have ... = card (v-gs X) ∗ card (v-gs Y)
    by (rule card-cartesian-product)
  finally show ?thesis .
qed
```

Definition 6 – elementary plucking step

```
definition plucking-step :: graph set ⇒ graph set where
  plucking-step X = (let vXp = v-gs X;
      S = (SOME S. S ⊆ vXp ∧ sunflower S ∧ card S = p);
      U = {E ∈ X. v E ∈ S};
      Vs = ⋂ S;
      Gs = Vs^2
     in X − U ∪ {Gs})
end

context second-assumptions
begin
```

Lemma 9 – for elementary plucking step

```
lemma v-sameprod-subset: v (Vs^2) ⊆ Vs unfolding binprod-def v-def
  by (auto simp: doubleton-eq-iff)

lemma plucking-step: assumes X: X ⊆ 𝒢l
  and L: card (v-gs X) > L
  and Y: Y = plucking-step X
shows card (v-gs Y) ≤ card (v-gs X) − p + 1
  Y ⊆ 𝒢l
  POS ∩ ACC X ⊆ ACC Y
  2 ^ p ∗ card (ACC-cf Y − ACC-cf X) ≤ (k − 1) ^ m
  Y ≠ {}
proof −
  let ?vXp = v-gs X
  have sf-precond: ∀ A∈ ?vXp. finite A ∧ card A ≤ l
    using X unfolding 𝒢l-def 𝒢l-def v-gs-def by (auto intro: finite-vG intro!: v-𝒢
v-card2)
  note sunflower = Erdos-Rado-sunflower[OF sf-precond]
  from p have p0: p ≠ 0 by auto
  have (p − 1) ^ l ∗ fact l < card ?vXp  using L[unfolded L-def]
    by (simp add: ac-simps)
  note sunflower = sunflower[OF this]
  define S where S = (SOME S. S ⊆ ?vXp ∧ sunflower S ∧ card S = p)
  define U where U = {E ∈ X. v E ∈ S}
  define Vs where Vs = ⋂ S
  define Gs where Gs = Vs^2
  let ?U = U
```

let *?New* = *Gs* :: *graph*
have *Y*: *Y* = *X* − *U* ∪ {*?New*}
using *Y*[*unfolded plucking-step-def Let-def*, *folded S-def*, *folded U-def*, *folded Vs-def*, *folded Gs-def*] **.**
have *U*: *U* ⊆ 𝒢*l* **using** *X* **unfolding** *U-def* **by** *auto*
hence *U* ⊆ 𝒢 **unfolding** 𝒢*l-def* **by** *auto*
from *sunflower*
have ∃ *S*. *S* ⊆ *?vXp* ∧ *sunflower S* ∧ *card S* = *p* **by** *auto*
from *someI-ex*[*OF this*, *folded S-def*]
have *S*: *S* ⊆ *?vXp sunflower S card S* = *p* **by** (*auto simp*: *Vs-def*)
have *fin1*: *finite ?vXp* **using** *finite-v-gs-Gl*[*OF X*] **.**
from *X* **have** *finX*: *finite X* **unfolding** 𝒢*l-def*
using *finite-subset*[*of X*, *OF - finite-*𝒢] **by** *auto*
from *fin1 S* **have** *finS*: *finite S* **by** (*metis finite-subset*)
from *finite-subset*[*OF - finX*] **have** *finU*: *finite U* **unfolding** *U-def* **by** *auto*
from *S p* **have** *Snempty*: *S* ≠ {} **by** *auto*
have *UX*: *U* ⊆ *X* **unfolding** *U-def* **by** *auto*
{
from *Snempty* **obtain** *s* **where** *sS*: *s* ∈ *S* **by** *auto*
with *S* **have** *s* ∈ *v-gs X* **by** *auto*
then obtain *Sp* **where** *Sp* ∈ *X* **and** *sSp*: *s* = *v Sp*
unfolding *v-gs-def* **by** *auto*
hence ∗: *Sp* ∈ *U* **using** ‹*s* ∈ *S*› **unfolding** *U-def* **by** *auto*
from ∗ *X UX* **have** *le*: *card* (*v Sp*) ≤ *l finite* (*v Sp*) *Sp* ∈ 𝒢
unfolding 𝒢*l-def* 𝒢*l-def* **using** *finite-vG*[*of Sp*] **by** *auto*
hence *m*: *v Sp* ⊆ [*m*] **by** (*intro v-*𝒢)
have *Vs* ⊆ *v Sp* **using** *sS sSp* **unfolding** *Vs-def* **by** *auto*
with *card-mono*[*OF* ‹*finite* (*v Sp*)› *this*] *finite-subset*[*OF this* ‹*finite* (*v Sp*)›] *le* ∗ *m*
have *card Vs* ≤ *l U* ≠ {} *finite Vs Vs* ⊆ [*m*] **by** *auto*
}
hence *card-Vs*: *card Vs* ≤ *l* **and** *Unempty*: *U* ≠ {}
and *fin-Vs*: *finite Vs* **and** *Vsm*: *Vs* ⊆ [*m*] **by** *auto*
have *vGs*: *v Gs* ⊆ *Vs* **unfolding** *Gs-def* **by** (*rule v-sameprod-subset*)
have *GsG*: *Gs* ∈ 𝒢 **unfolding** *Gs-def* 𝒢*-def*
by (*intro CollectI Inter-subset sameprod-mono Vsm*)
have *GsGl*: *Gs* ∈ 𝒢*l* **unfolding** 𝒢*l-def* **using** *GsG vGs card-Vs card-mono*[*OF - vGs*]
by (*simp add*: *fin-Vs*)
hence *DsDl*: *?New* ∈ 𝒢*l* **using** *UX*
unfolding 𝒢*l-def* 𝒢*-def* 𝒢*l-def* 𝒢*-def* **by** *auto*
with *X U* **show** *Y* ⊆ 𝒢*l* **unfolding** *Y* **by** *auto*
from *X* **have** *XD*: *X* ⊆ 𝒢 **unfolding** 𝒢*l-def* **by** *auto*
have *vplus-dsU*: *v-gs U* = *S* **using** *S*(*1*)
unfolding *v-gs-def U-def* **by** *force*
have *vplus-dsXU*: *v-gs* (*X* − *U*) = *v-gs X* − *v-gs U*
unfolding *v-gs-def U-def* **by** *auto*
have *card* (*v-gs Y*) = *card* (*v-gs* (*X* − *U* ∪ {*?New*}))
unfolding *Y* **by** *simp*

```
  also have v-gs (X − U ∪ {?New}) = v-gs (X − U) ∪ v-gs ({?New})
    unfolding v-gs-union ..
  also have v-gs ({?New}) = {v (Gs)} unfolding v-gs-def image-comp o-def by
simp
  also have card (v-gs (X − U) ∪ . . .) ≤ card (v-gs (X − U)) + card . . .
    by (rule card-Un-le)
  also have . . . ≤ card (v-gs (X − U)) + 1 by auto
  also have v-gs (X − U) = v-gs X − v-gs U by fact
  also have card . . . = card (v-gs X) − card (v-gs U)
    by (rule card-Diff-subset, force simp: vplus-dsU finS,
      insert UX, auto simp: v-gs-def)
  also have card (v-gs U) = card S unfolding vplus-dsU ..
  finally show card (v-gs Y) ≤ card (v-gs X) − p + 1
    using S by auto
  show Y ≠ {} unfolding Y using Unempty by auto
  {
    fix G
    assume G ∈ ACC X and GPOS: G ∈ POS
    from this[unfolded ACC-def] POS-𝒢 have G: G ∈ 𝒢 X ⊩ G by auto
    from this[unfolded accepts-def] obtain D :: graph where
      D: D ∈ X D ⊆ G by auto
    have G ∈ ACC Y
    proof (cases D ∈ Y)
      case True
      with D G show ?thesis unfolding accepts-def ACC-def by auto
    next
      case False
      with D have DU: D ∈ U unfolding Y by auto
      from GPOS[unfolded POS-def 𝒦-def] obtain K where GK: G = (v K)^2
card (v K) = k by auto
      from DU[unfolded U-def] have v D ∈ S by auto
      hence Vs ⊆ v D unfolding Vs-def by auto
      also have . . . ⊆ v G
        by (intro v-mono D)
      also have . . . = v K unfolding GK
        by (rule v-sameprod, unfold GK, insert k2, auto)
      finally have Gs ⊆ G unfolding Gs-def GK
        by (intro sameprod-mono)
      with D DU have D ∈ ?U ?New ⊆ G by (auto)
      hence Y ⊩ G unfolding accepts-def Y by auto
      thus ?thesis using G by auto
    qed
  }
  thus POS ∩ ACC X ⊆ ACC Y by auto

  from ex-bij-betw-nat-finite[OF finS, unfolded ‹card S = p›]
  obtain Si where Si: bij-betw Si {0 ..< p} S by auto
  define G where G = (λ i. SOME Gb. Gb ∈ X ∧ v Gb = Si i)
  {
```

```
    fix i
    assume i < p
    with Si have SiS: Si i ∈ S unfolding bij-betw-def by auto
    with S have Si i ∈ v-gs X by auto
    hence ∃ G. G ∈ X ∧ v G = Si i
      unfolding v-gs-def by auto
    from someI-ex[OF this]
    have (G i) ∈ X ∧ v (G i) = Si i
      unfolding G-def by blast
    hence G i ∈ X v (G i) = Si i
      G i ∈ U v (G i) ∈ S using SiS unfolding U-def
      by auto
  } note G = this
  have SvG: S = v ‘ G ‘ {0 ..< p} unfolding Si[unfolded bij-betw-def,
        THEN conjunct2, symmetric] image-comp o-def using G(2) by auto
  have injG: inj-on G {0 ..< p}
  proof (standard, goal-cases)
    case (1 i j)
    hence Si i = Si j using G[of i] G[of j] by simp
    with 1(1,2) Si show i = j
      by (metis Si bij-betw-iff-bijections)
  qed
  define r where r = card U
  have rq: r ≥ p unfolding r-def ‹card S = p›[symmetric] vplus-dsU[symmetric]
    unfolding v-gs-def
    by (rule card-image-le[OF finU])

  let ?Vi = λ i. v (G i)
  let ?Vis = λ i. ?Vi i − Vs
  define s where s = card Vs
  define si where si i = card (?Vi i) for i
  define ti where ti i = card (?Vis i) for i
  {
    fix i
    assume i: i < p
    have Vs-Vi: Vs ⊆ ?Vi i using i unfolding Vs-def
      using G[OF i] unfolding SvG by auto
    have finVi: finite (?Vi i)
      using G(4)[OF i] S(1) sf-precond
      by (meson finite-numbers finite-subset subset-eq)
    from S(1) have G i ∈ 𝒢 using G(1)[OF i] X unfolding 𝒢l-def 𝒢-def 𝒢l-def
by auto
    hence finGi: finite (G i)
      using finite-members-𝒢 by auto
    have ti: ti i = si i − s unfolding ti-def si-def s-def
      by (rule card-Diff-subset[OF fin-Vs Vs-Vi])
    have size1: s ≤ si i unfolding s-def si-def
      by (intro card-mono finVi Vs-Vi)
     have size2: si i ≤ l unfolding si-def using G(4)[OF i] S(1) sf-precond by
```

auto
  note *Vs-Vi finVi ti size1 size2 finGi* ‹$G\ i \in \mathcal{G}$›
 } **note** *i-props* = *this*
 define *fstt* **where** *fstt e* = (*SOME x*. $x \in e \wedge x \notin Vs$) **for** *e*
 define *sndd* **where** *sndd e* = (*SOME x*. $x \in e \wedge x \neq$ *fstt e*) **for** *e*
 {
  fix *e* :: *nat set*
  assume ∗: *card e* = *2* ¬ $e \subseteq Vs$
  from ∗(*1*) **obtain** *x y* **where** *e*: $e = \{x,y\}$ $x \neq y$
   by (*meson card-2-iff*)
  with ∗ **have** $\exists\ x.\ x \in e \wedge x \notin Vs$ **by** *auto*
  from *someI-ex*[*OF this*, *folded fstt-def*]
  have *fst*: *fstt e* $\in$ *e fstt e* $\notin$ *Vs* **by** *auto*
  with ∗ *e* **have** $\exists\ x.\ x \in e \wedge x \neq$ *fstt e*
   by (*metis insertCI*)
  from *someI-ex*[*OF this*, *folded sndd-def*] **have** *snd*: *sndd e* $\in$ *e sndd e* $\neq$ *fstt e*
by *auto*
  from *fst snd e* **have** {*fstt e*, *sndd e*} = *e fstt e* $\notin$ *Vs fstt e* $\neq$ *sndd e* **by** *auto*
 } **note** *fstt* = *this*
 {
  fix *f*
  assume $f \in$ *ACC-cf Y* − *ACC-cf X*
  hence *fake*: $f \in$ *ACC-cf* {*?New*} − *ACC-cf U* **unfolding** *Y ACC-cf-def accepts-def*
   Diff-iff U-def Un-iff mem-Collect-eq **by** *blast*
  hence *f*: $f \in \mathcal{F}$ **using** *ACC-cf-*$\mathcal{F}$ **by** *auto*
  hence *C f* $\in$ *NEG* **unfolding** *NEG-def* **by** *auto*
  with *NEG-*$\mathcal{G}$ **have** *Cf*: *C f* $\in \mathcal{G}$ **by** *auto*
  from *fake* **have** $f \in$ *ACC-cf* {*?New*} **by** *auto*
  from *this*[*unfolded ACC-cf-def accepts-def*] *Cf*
  have *GsCf*: *Gs* $\subseteq$ *C f* **and** *Cf*: *C f* $\in \mathcal{G}$ **by** *auto*
  from *fake* **have** $f \notin$ *ACC-cf U* **by** *auto*
  from *this*[*unfolded ACC-cf-def*] *Cf f* **have** ¬ (*U* ⊩ *C f*) **by** *auto*
  from *this*[*unfolded accepts-def*]
  have *UCf*: $D \in U \Longrightarrow \neg\ D \subseteq$ *C f* **for** *D* **by** *auto*
  let *?prop* = λ *i e*. *fstt e* $\in$ *v* (*G i*) − *Vs* ∧
     sndd e $\in$ *v* (*G i*) ∧ *e* $\in$ *G i* ∩ ([*m*]^**2**)
    ∧ *f* (*fstt e*) = *f* (*sndd e*) ∧ *f* (*sndd e*) $\in$ [*k* − *1*] ∧ {*fstt e*, *sndd e*} = *e*
  define *pair* **where** *pair i* = (*if i* < *p then* (*SOME pair*. *?prop i pair*) *else undefined*) **for** *i*
  define *u* **where** *u i* = *fstt* (*pair i*) **for** *i*
  define *w* **where** *w i* = *sndd* (*pair i*) **for** *i*
  {
   fix *i*
   assume *i*: *i* < *p*
   from *i* **have** *?Vi i* $\in$ *S* **unfolding** *SvG* **by** *auto*
   hence *Vs* $\subseteq$ *?Vi i* **unfolding** *Vs-def* **by** *auto*
   from *sameprod-mono*[*OF this*, *folded Gs-def*]
   have ∗: *Gs* $\subseteq$ *v* (*G i*)^**2** **.**

from *i* **have** *Gi*: *G i* ∈ *U* **using** *G*[*OF i*] **by** *auto*
from *UCf*[*OF Gi*] *i-props*[*OF i*] **have** ¬ *G i* ⊆ *C f* **and** *Gi*: *G i* ∈ 𝒢 **by** *auto*
then obtain *edge* **where**
edgep: *edge* ∈ *G i* **and** *edgen*: *edge* ∉ *C f* **by** *auto*
from *edgep Gi* **obtain** *x y* **where** *edge*: *edge* = {*x*,*y*}
and *xy*: {*x*,*y*} ∈ [*m*]^**2** {*x*,*y*} ⊆ [*m*] *card* {*x*,*y*} = *2* **unfolding** 𝒢*-def binprod-def*
by *force*
define *a* **where** *a* = *fstt edge*
define *b* **where** *b* = *sndd edge*
from *edgen*[*unfolded C-def edge*] *xy* **have** *id*: *f x* = *f y* **by** *simp*
from *edgen GsCf edge* **have** *edgen*: {*x*,*y*} ∉ *Gs* **by** *auto*
from *edgen*[*unfolded Gs-def sameprod-altdef*] *xy* **have** ¬ {*x*,*y*} ⊆ *Vs* **by** *auto*
from *fstt*[*OF* ‹*card* {*x*,*y*} = *2*› *this*, *folded edge*, *folded a-def b-def*] *edge*
have *a*: *a* ∉ *Vs* **and** *id-ab*: {*x*,*y*} = {*a*,*b*} **by** *auto*
from *id-ab id* **have** *id*: *f a* = *f b* **by** (*auto simp*: *doubleton-eq-iff*)
let *?pair* = (*a*,*b*)
note *ab* = *xy*[*unfolded id-ab*]
from *f*[*unfolded* ℱ*-def*] *ab* **have** *fb*: *f b* ∈ [*k* − *1*] **by** *auto*
note *edge* = *edge*[*unfolded id-ab*]
from *edgep*[*unfolded edge*] *v-mem-sub*[*OF* ‹*card* {*a*,*b*} = *2*›, *of G i*] *id*
have *?prop i edge* **using** *edge ab a fb* **unfolding** *a-def b-def* **by** *auto*
from *someI*[*of ?prop i*, *OF this*] **have** *?prop i* (*pair i*) **using** *i* **unfolding** *pair-def* **by** *auto*
from *this*[*folded u-def w-def*] *edgep*
have *u i* ∈ *v* (*G i*) − *Vs* *w i* ∈ *v* (*G i*) *pair i* ∈ *G i* ∩ [*m*]^**2**
f (*u i*) = *f* (*w i*) *f* (*w i*) ∈ [*k* − *1*] *pair i* = {*u i*, *w i*}
by *auto*
} **note** *uw* = *this*
from *uw*(*3*) **have** *Pi*: *pair* ∈ Pi_E {*0* ..< *p*} *G* **unfolding** *pair-def* **by** *auto*
define *Us* **where** *Us* = *u* ‘ {*0* ..< *p*}
define *Ws* **where** *Ws* = [*m*] − *Us*
{
fix *i*
assume *i*: *i* < *p*
note *uwi* = *uw*[*OF this*]
from *uwi* **have** *ex*: ∃ *x* ∈ [*k* − *1*]. *f* ‘ {*u i*, *w i*} = {*x*} **by** *auto*
from *uwi* **have** ∗: *u i* ∈ [*m*] *w i* ∈ [*m*] {*u i*, *w i*} ∈ *G i* **by** (*auto simp*: *sameprod-altdef*)
have *w i* ∉ *Us*
proof
assume *w i* ∈ *Us*
then obtain *j* **where** *j*: *j* < *p* **and** *wij*: *w i* = *u j* **unfolding** *Us-def* **by** *auto*
with *uwi* **have** *ij*: *i* ≠ *j* **unfolding** *binprod-def* **by** *auto*
note *uwj* = *uw*[*OF j*]
from *ij i j Si*[*unfolded bij-betw-def*]
have *diff*: *v* (*G i*) ≠ *v* (*G j*) **unfolding** *G*(*2*)[*OF i*] *G*(*2*)[*OF j*] *inj-on-def* **by** *auto*

from *uwi wij* **have** *uj*: *u j* ∈ *v* (*G i*) **by** *auto*
with ‹*sunflower S*›[*unfolded sunflower-def*, *rule-format*] *G*(*4*)[*OF i*] *G*(*4*)[*OF j*] *uwj*(*1*) *diff*
have *u j* ∈ ⋂ *S* **by** *blast*
with *uwj*(*1*)[*unfolded Vs-def*] **show** *False* **by** *simp*
qed
with ∗ **have** *wi*: *w i* ∈ *Ws* **unfolding** *Ws-def* **by** *auto*
from *uwi* **have** *wi2*: *w i* ∈ *v* (*G i*) **by** *auto*
define *W* **where** *W* = *Ws* ∩ *v* (*G i*)
from *G*(*1*)[*OF i*] *X*[*unfolded 𝒢l-def 𝒢l-def*] *i-props*[*OF i*]
have *finite* (*v* (*G i*)) *card* (*v* (*G i*)) ≤ *l* **by** *auto*
with *card-mono*[*OF this*(*1*), *of W*] **have**
W: *finite W card W* ≤ *l W* ⊆ [*m*] − *Us* **unfolding** *W-def Ws-def* **by** *auto*
from *wi wi2* **have** *wi*: *w i* ∈ *W* **unfolding** *W-def* **by** *auto*
from *wi ex W* ∗ **have** {*u i*, *w i*} ∈ *G i* ∧ *u i* ∈ [*m*] ∧ *w i* ∈ [*m*] − *Us* ∧ *f* (*u i*) = *f* (*w i*) **by** *force*
} **note** *uw1* = *this*
have *inj*: *inj-on u* {*0* ..< *p*}
proof −
{
fix *i j*
assume *i*: *i* < *p* **and** *j*: *j* < *p*
and *id*: *u i* = *u j* **and** *ij*: *i* ≠ *j*
from *ij i j Si*[*unfolded bij-betw-def*]
have *diff*: *v* (*G i*) ≠ *v* (*G j*) **unfolding** *G*(*2*)[*OF i*] *G*(*2*)[*OF j*] *inj-on-def* **by** *auto*
from *uw*[*OF i*] **have** *ui*: *u i* ∈ *v* (*G i*) − *Vs* **by** *auto*
from *uw*[*OF j*, *folded id*] **have** *uj*: *u i* ∈ *v* (*G j*) **by** *auto*
with ‹*sunflower S*›[*unfolded sunflower-def*, *rule-format*] *G*(*4*)[*OF i*] *G*(*4*)[*OF j*] *uw*[*OF i*] *diff*
have *u i* ∈ ⋂ *S* **by** *blast*
with *ui* **have** *False* **unfolding** *Vs-def* **by** *auto*
}
thus *?thesis* **unfolding** *inj-on-def* **by** *fastforce*
qed
have *card*: *card* ([*m*] − *Us*) = *m* − *p*
proof (*subst card-Diff-subset*)
show *finite Us* **unfolding** *Us-def* **by** *auto*
show *Us* ⊆ [*m*] **unfolding** *Us-def* **using** *uw1* **by** *auto*
have *card Us* = *p* **unfolding** *Us-def* **using** *inj*
by (*simp add*: *card-image*)
thus *card* [*m*] − *card Us* = *m* − *p* **by** *simp*
qed
hence (∀ *i* < *p*. *pair i* ∈ *G i*) ∧ *inj-on u* {*0* ..< *p*} ∧ (∀ *i* < *p*. *w i* ∈ [*m*] − *u* ‘ {*0* ..< *p*} ∧ *f* (*u i*) = *f* (*w i*))
using *inj uw1 uw* **unfolding** *Us-def* **by** *auto*
from *this*[*unfolded u-def w-def*] *Pi card*[*unfolded Us-def u-def w-def*]
have ∃ *e* ∈ Pi_E {*0*..<*p*} *G*. (∀ *i*<*p*. *e i* ∈ *G i*) ∧
card ([*m*] − (λ*i*. *fstt* (*e i*)) ‘ {*0*..<*p*}) = *m* − *p* ∧

(∀ *i*<*p*. *sndd* (*e i*) ∈ [*m*] − (λ*i*. *fstt* (*e i*)) ' {*0*..<*p*} ∧ *f* (*fstt* (*e i*)) = *f* (*sndd* (*e i*)))
by *blast*
} **note** *fMem* = *this*
define *Pi2* **where** *Pi2 W* = Pi_E ([*m*] − *W*) (λ -. [*k* − *1*]) **for** *W*
define *merge* **where** *merge* =
(λ *e* (*g* :: *nat* ⇒ *nat*) *v*. *if v* ∈ (λ *i*. *fstt* (*e i*)) ' {*0* ..< *p*} *then g* (*sndd* (*e* (*SOME i*. *i* < *p* ∧ *v* = *fstt* (*e i*)))) *else g v*)
let *?W* = λ *e*. (λ *i*. *fstt* (*e i*)) ' {*0*..<*p*}
have *ACC-cf Y* − *ACC-cf X* ⊆ { *merge e g* | *e g*. *e* ∈ Pi_E {*0*..<*p*} *G* ∧ *card* ([*m*] − *?W e*) = *m* − *p* ∧ *g* ∈ *Pi2* (*?W e*)}
(**is** - ⊆ *?R*)
proof
fix *f*
assume *mem*: *f* ∈ *ACC-cf Y* − *ACC-cf X*
with *ACC-cf-$\mathcal{F}$* **have** *f* ∈ $\mathcal{F}$ **by** *auto*
hence *f*: *f* ∈ [*m*] $\to_E$ [*k* − *1*] **unfolding** *$\mathcal{F}$-def* **.**
from *fMem*[*OF mem*] **obtain** *e* **where** *e*: *e* ∈ Pi_E {*0*..<*p*} *G*
⋀ *i*. *i*<*p* ⟹ *e i* ∈ *G i*
card ([*m*] − *?W e*) = *m* − *p*
⋀ *i*. *i*<*p* ⟹ *sndd* (*e i*) ∈ [*m*] − *?W e* ∧ *f* (*fstt* (*e i*)) = *f* (*sndd* (*e i*)) **by** *auto*
define *W* **where** *W* = *?W e*
note *e* = *e*[*folded W-def*]
let *?g* = *restrict f* ([*m*] − *W*)
let *?h* = *merge e ?g*
have *f* ∈ *?R*
proof (*intro CollectI exI*[*of - e*] *exI*[*of - ?g*], *unfold W-def*[*symmetric*], *intro conjI e*)
show *?g* ∈ *Pi2 W* **unfolding** *Pi2-def* **using** *f* **by** *auto*
{
fix *v* :: *nat*
have *?h v* = *f v*
proof (*cases v* ∈ *W*)
case *False*
thus *?thesis* **using** *f* **unfolding** *merge-def* **unfolding** *W-def*[*symmetric*] **by** *auto*
next
case *True*
from *this*[*unfolded W-def*] **obtain** *i* **where** *i*: *i* < *p* **and** *v*: *v* = *fstt* (*e i*) **by** *auto*
define *j* **where** *j* = (*SOME j*. *j* < *p* ∧ *v* = *fstt* (*e j*))
from *i v* **have** ∃ *j*. *j* < *p* ∧ *v* = *fstt* (*e j*) **by** *auto*
from *someI-ex*[*OF this*, *folded j-def*] **have** *j*: *j* < *p* **and** *v*: *v* = *fstt* (*e j*) **by** *auto*
have *?h v* = *restrict f* ([*m*] − *W*) (*sndd* (*e j*))
unfolding *merge-def* **unfolding** *W-def*[*symmetric*] *j-def* **using** *True* **by** *auto*
also have . . . = *f* (*sndd* (*e j*)) **using** *e*(*4*)[*OF j*] **by** *auto*

also have ... = *f* (*fstt* (*e j*)) **using** *e*(*4*)[*OF j*] **by** *auto*
also have ... = *f v* **using** *v* **by** *simp*
finally show *?thesis* **.**
qed
}
thus *f* = *?h* **by** *auto*
qed
thus *f* ∈ *?R* **by** *auto*
qed
also have ... ⊆ (λ (*e*,*g*). (*merge e g*)) ‘ (*Sigma* (Pi_E {*0*..<*p*} *G* ∩ {*e*. *card* ([*m*] − *?W e*) = *m* − *p*}) (λ *e*. *Pi2* (*?W e*)))
(**is** - ⊆ *?f* ‘ *?R*)
by *auto*
finally have *sub*: *ACC-cf Y* − *ACC-cf X* ⊆ *?f* ‘ *?R* **.**
have *fin*[*simp*,*intro*]: *finite* [*m*] *finite* [*k* − *Suc 0*] **unfolding** *numbers-def* **by** *auto*
have *finPie*[*simp*, *intro*]: *finite* (Pi_E {*0*..<*p*} *G*)
by (*intro finite-PiE*, *auto intro*: *i-props*)
have *finR*: *finite ?R* **unfolding** *Pi2-def*
by (*intro finite-SigmaI finite-Int allI finite-PiE i-props*, *auto*)
have *card* (*ACC-cf Y* − *ACC-cf X*) ≤ *card* (*?f* ‘ *?R*)
by (*rule card-mono*[*OF finite-imageI*[*OF finR*] *sub*])
also have ... ≤ *card ?R*
by (*rule card-image-le*[*OF finR*])
also have ... = ($\sum$ *e*∈(Pi_E {*0*..<*p*} *G* ∩ {*e*. *card* ([*m*] − *?W e*) = *m* − *p*}). *card* (*Pi2* (*?W e*)))
by (*rule card-SigmaI*, *unfold Pi2-def*,
(*intro finite-SigmaI allI finite-Int finite-PiE i-props*, *auto*)+)
also have ... = ($\sum$ *e*∈Pi_E {*0*..<*p*} *G* ∩ {*e*. *card* ([*m*] − *?W e*) = *m* − *p*}. (*k* − *1*) ^ (*card* ([*m*] − *?W e*)))
by (*rule sum.cong*[*OF refl*], *unfold Pi2-def*, *subst card-PiE*, *auto*)
also have ... = ($\sum$ *e*∈Pi_E {*0*..<*p*} *G* ∩ {*e*. *card* ([*m*] − *?W e*) = *m* − *p*}. (*k* − *1*) ^ (*m* − *p*))
by (*rule sum.cong*[*OF refl*], *rule arg-cong*[*of* - - λ *n*. (*k* − *1*) ^ *n*], *auto*)
also have ... ≤ ($\sum$ *e*∈Pi_E {*0*..<*p*} *G*. (*k* − *1*) ^ (*m* − *p*))
by (*rule sum-mono2*, *auto*)
also have ... = *card* (Pi_E {*0*..<*p*} *G*) ∗ (*k* − *1*) ^ (*m* − *p*) **by** *simp*
also have ... = ($\prod$ *i* = *0*..<*p*. *card* (*G i*)) ∗ (*k* − *1*) ^ (*m* − *p*)
by (*subst card-PiE*, *auto*)
also have ... ≤ ($\prod$ *i* = *0*..<*p*. (*k* − *1*) *div 2*) ∗ (*k* − *1*) ^ (*m* − *p*)
proof −
{
fix *i*
assume *i*: *i* < *p*
from *G*[*OF i*] *X*
have *GiG*: *G i* ∈ $\mathcal{G}$
unfolding $\mathcal{G}$*l-def* $\mathcal{G}$*-def* $\mathcal{G}$*-def sameprod-altdef* **by** *force*
from *i-props*[*OF i*] **have** *finGi*: *finite* (*G i*) **by** *auto*
have *finvGi*: *finite* (*v* (*G i*)) **by** (*rule finite-vG*, *insert i-props*[*OF i*], *auto*)

```
      have card (G i) ≤ card ((v (G i))^2)
        by (intro card-mono[OF sameprod-finite], rule finvGi, rule v-𝒢-2[OF GiG])
      also have … ≤ l choose 2
      proof (subst card-sameprod[OF finvGi], rule choose-mono)
         show card (v (G i)) ≤ l using i-props[OF i] unfolding ti-def si-def by
simp
      qed
      also have l choose 2 = l * (l − 1) div 2 unfolding choose-two by simp
     also have l * (l − 1) = k − l unfolding kl2 power2-eq-square by (simp add:
algebra-simps)
      also have … div 2 ≤ (k − 1) div 2
        by (rule div-le-mono, insert l2, auto)
      finally have card (G i) ≤ (k − 1) div 2 .
    }
    thus ?thesis by (intro mult-right-mono prod-mono, auto)
  qed
  also have … = ((k − 1) div 2) ^ p * (k − 1) ^ (m − p)
    by simp
  also have … ≤ ((k − 1) ^ p div (2^p)) * (k − 1) ^ (m − p)
    by (rule mult-right-mono; auto simp: div-mult-pow-le)
  also have … ≤ ((k − 1) ^ p * (k − 1) ^ (m − p)) div 2^p
    by (rule div-mult-le)
  also have … = (k − 1)^m div 2^p
  proof −
    have p + (m − p) = m using mp by simp
    thus ?thesis by (subst power-add[symmetric], simp)
  qed
  finally have card (ACC-cf Y − ACC-cf X) ≤ (k − 1) ^ m div 2 ^ p .
  hence 2 ^ p * card (ACC-cf Y − ACC-cf X) ≤ 2^p * ((k − 1) ^ m div 2 ^ p)
by simp
  also have … ≤ (k − 1)^m  by simp
  finally show 2^p * card (ACC-cf Y − ACC-cf X) ≤ (k − 1) ^ m .
qed
```

Definition 6

```
function PLU-main :: graph set ⇒ graph set × nat where
  PLU-main X = (if X ⊆ 𝒢l ∧ L < card (v-gs X) then
     map-prod id Suc (PLU-main (plucking-step X)) else
     (X, 0))
  by pat-completeness auto

termination
proof (relation measure (λ X. card (v-gs X)), force, goal-cases)
  case (1 X)
  hence X ⊆ 𝒢l and LL: L < card (v-gs X) by auto
  from plucking-step(1)[OF this refl]
  have card (v-gs (plucking-step X)) ≤ card (v-gs X) − p + 1 .
  also have … < card (v-gs X) using p L3 LL
    by auto
```

```
  finally show ?case by simp
qed

declare PLU-main.simps[simp del]

definition PLU :: graph set ⇒ graph set where
  PLU X = fst (PLU-main X)
```

Lemma 7

```
lemma PLU-main-n: assumes X ⊆ 𝒢l and PLU-main X = (Z, n)
  shows n ∗ (p − 1) ≤ card (v-gs X)
  using assms
proof (induct X  arbitrary: Z n  rule: PLU-main.induct)
  case (1 X Z n)
  note [simp] = PLU-main.simps[of X]
  show ?case
  proof (cases card (v-gs X) ≤ L)
    case True
    thus ?thesis using 1 by auto
  next
    case False
    define Y where Y = plucking-step X
    obtain q where PLU: PLU-main Y = (Z, q) and n: n = Suc q
      using ‹PLU-main X = (Z,n)›[unfolded PLU-main.simps[of X], folded Y-def]
using False 1(2) by (cases PLU-main Y, auto)
    from False have L: card (v-gs X) > L by auto
    note step = plucking-step[OF 1(2) this Y-def]
    from False 1 have X ⊆ 𝒢l ∧ L < card (v-gs X) by auto
    note IH = 1(1)[folded Y-def, OF this step(2) PLU]
    have n ∗ (p − 1) = (p − 1) + q ∗ (p − 1) unfolding n by simp
    also have . . . ≤ (p − 1) + card (v-gs Y) using IH by simp
    also have . . . ≤ p − 1 + (card (v-gs X) − p + 1) using step(1) by simp
    also have . . . = card (v-gs X) using L Lp p by simp
    finally show ?thesis .
  qed
qed
```

Definition 8

```
definition sqcup :: graph set ⇒ graph set ⇒ graph set (infixl ‹⊔› 65) where
  X ⊔ Y = PLU (X ∪ Y)

definition sqcap :: graph set ⇒ graph set ⇒ graph set (infixl ‹⊓› 65) where
  X ⊓ Y = PLU (X ⊙l Y)

definition deviate-pos-cup :: graph set ⇒ graph set ⇒ graph set (‹∂⊔Pos›) where
  ∂⊔Pos X Y = POS ∩ ACC (X ∪ Y) − ACC (X ⊔ Y)

definition deviate-pos-cap :: graph set ⇒ graph set ⇒ graph set (‹∂⊓Pos›) where
  ∂⊓Pos X Y = POS ∩ ACC (X ⊙ Y) − ACC (X ⊓ Y)
```

```
definition deviate-neg-cup :: graph set ⇒ graph set ⇒ colorf set (‹∂⊔Neg›) where
  ∂⊔Neg X Y = ACC-cf (X ⊔ Y) − ACC-cf (X ∪ Y)

definition deviate-neg-cap :: graph set ⇒ graph set ⇒ colorf set (‹∂⊓Neg›) where
  ∂⊓Neg X Y = ACC-cf (X ⊓ Y) − ACC-cf (X ⊙ Y)
```

Lemma 9 – without applying Lemma 7

```
lemma PLU-main: assumes X ⊆ 𝒢l
  and PLU-main X = (Z, n)
shows Z ∈ 𝒫L𝒢l
  ∧ (Z = {} ⟷ X = {})
  ∧ POS ∩ ACC X ⊆ ACC Z
  ∧ 2 ^ p ∗ card (ACC-cf Z − ACC-cf X) ≤ (k − 1) ^ m ∗ n
  using assms
proof (induct X  arbitrary: Z n  rule: PLU-main.induct)
  case (1 X Z n)
  note [simp] = PLU-main.simps[of X]
  show ?case
  proof (cases card (v-gs X) ≤ L)
    case True
    from True show ?thesis using 1 by (auto simp: id 𝒫L𝒢l-def)
  next
    case False
    define Y where Y = plucking-step X
    obtain q where PLU: PLU-main Y = (Z, q) and n: n = Suc q
      using ‹PLU-main X = (Z,n)›[unfolded PLU-main.simps[of X], folded Y-def]
using False 1(2) by (cases PLU-main Y, auto)
    from False have card (v-gs X) > L by auto
    note step = plucking-step[OF 1(2) this Y-def]
    from False 1 have X ⊆ 𝒢l ∧ L < card (v-gs X) by auto
    note IH = 1(1)[folded Y-def, OF this step(2) PLU] ‹Y ≠ {}›
    let ?Diff = λ X Y. ACC-cf X − ACC-cf Y
    have finNEG: finite NEG
      using NEG-𝒢 infinite-super by blast
    have ?Diff Z X ⊆ ?Diff Z Y ∪ ?Diff Y X by auto
    from card-mono[OF finite-subset[OF - finite-ℱ] this] ACC-cf-ℱ
     have 2 ^ p ∗ card (?Diff Z X) ≤ 2 ^ p ∗ card (?Diff Z Y ∪ ?Diff Y X) by
auto
    also have . . . ≤ 2 ^ p ∗ (card (?Diff Z Y) + card (?Diff Y X))
      by (rule mult-left-mono, rule card-Un-le, simp)
    also have . . . = 2 ^ p ∗ card (?Diff Z Y) + 2 ^ p ∗ card (?Diff Y X)
      by (simp add: algebra-simps)
    also have . . . ≤ ((k − 1) ^ m) ∗ q + (k − 1) ^ m using IH step by auto
    also have . . . = ((k − 1) ^ m) ∗ Suc q by (simp add: ac-simps)
    finally have c: 2 ^ p ∗ card (ACC-cf Z − ACC-cf X) ≤ ((k − 1) ^ m) ∗ Suc
q by simp
    from False have X ≠ {} by auto
    thus ?thesis unfolding n using IH step c by auto
```

```
  qed
qed
```

Lemma 9

```
lemma assumes X: X ∈ 𝒫L𝒢l and Y: Y ∈ 𝒫L𝒢l
  shows PLU-union: PLU (X ∪ Y) ∈ 𝒫L𝒢l and
  sqcup: X ⊔ Y ∈ 𝒫L𝒢l and
  sqcup-sub: POS ∩ ACC (X ∪ Y) ⊆ ACC (X ⊔ Y) and
  deviate-pos-cup: ∂⊔Pos X Y = {} and
  deviate-neg-cup: card (∂⊔Neg X Y) < (k − 1)^m ∗ L / 2^(p − 1)
proof −
  obtain Z n where res: PLU-main (X ∪ Y) = (Z, n) by force
  hence PLU: PLU (X ∪ Y) = Z unfolding PLU-def by simp
  from X Y have XY: X ∪ Y ⊆ 𝒢l unfolding 𝒫L𝒢l-def by auto
  note main = PLU-main[OF this(1) res]
  from main show PLU (X ∪ Y) ∈ 𝒫L𝒢l unfolding PLU by simp
  thus X ⊔ Y ∈ 𝒫L𝒢l unfolding sqcup-def .
  from main show POS ∩ ACC (X ∪ Y) ⊆ ACC (X ⊔ Y)
    unfolding sqcup-def PLU by simp
  thus ∂⊔Pos X Y = {} unfolding deviate-pos-cup-def PLU sqcup-def by auto
  have card (v-gs (X ∪ Y)) ≤ card (v-gs X) + card (v-gs Y)
    unfolding v-gs-union by (rule card-Un-le)
  also have ... ≤ L + L using X Y unfolding 𝒫L𝒢l-def by simp
  finally have card (v-gs (X ∪ Y)) ≤ 2 ∗ L by simp
  with PLU-main-n[OF XY(1) res] have n ∗ (p − 1) ≤ 2 ∗ L by simp
  with p Lm m2 have n: n < 2 ∗ L by (cases n, auto, cases p − 1, auto)
  let ?r = real
  have ∗: (k − 1) ^ m > 0 using k l2 by simp
  have 2 ^ p ∗ card (∂⊔Neg X Y) ≤ 2 ^ p ∗ card (ACC-cf Z − ACC-cf (X ∪ Y))
unfolding deviate-neg-cup-def PLU sqcup-def
    by (rule mult-left-mono, rule card-mono[OF finite-subset[OF - finite-ℱ]], insert
ACC-cf-ℱ, force, auto)
  also have ... ≤ (k − 1) ^ m ∗ n using main by simp
  also have ... < (k − 1) ^ m ∗ (2 ∗ L) unfolding mult-less-cancel1 using n ∗
by simp
  also have ... = 2 ∗ ((k − 1) ^ m ∗ L) by simp
  finally have 2 ∗ (2^(p − 1) ∗ card (∂⊔Neg X Y)) < 2 ∗ ((k − 1) ^ m ∗ L)
using p by (cases p, auto)
  hence 2 ^(p − 1) ∗ card (∂⊔Neg X Y) < (k − 1)^m ∗ L by simp
  hence ?r (2 ^(p − 1) ∗ card (∂⊔Neg X Y)) < ?r ((k − 1)^m ∗ L) by linarith
  thus card (∂⊔Neg X Y) < (k − 1)^m ∗ L / 2^(p − 1) by (simp add: field-simps)
qed
```

Lemma 10

```
lemma assumes X: X ∈ 𝒫L𝒢l and Y: Y ∈ 𝒫L𝒢l
  shows PLU-joinl: PLU (X ⊙l Y) ∈ 𝒫L𝒢l and
  sqcap: X ⊓ Y ∈ 𝒫L𝒢l and
  deviate-neg-cap: card (∂⊓Neg X Y) < (k − 1)^m ∗ L^2 / 2^(p − 1) and
  deviate-pos-cap: card (∂⊓Pos X Y) ≤ ((m − l − 1) choose (k − l − 1)) ∗ L^2
```

proof −
obtain *Z n* **where** *res*: *PLU-main* (*X* ⊙*l Y*) = (*Z*, *n*) **by** *force*
hence *PLU*: *PLU* (*X* ⊙*l Y*) = *Z* **unfolding** *PLU-def* **by** *simp*
from *X Y* **have** *XY*: *X* ⊆ 𝒢*l Y* ⊆ 𝒢*l X* ⊆ 𝒢 *Y* ⊆ 𝒢 **unfolding** *𝒫L𝒢l-def 𝒢l-def* **by** *auto*
have *sub*: *X* ⊙*l Y* ⊆ 𝒢*l* **unfolding** *odotl-def* **using** *XY*
by (*auto split*: *option.splits*)
note *main* = *PLU-main*[*OF sub res*]
note *finV* = *finite-v-gs-Gl*[*OF XY*(*1*)] *finite-v-gs-Gl*[*OF XY*(*2*)]
have *X* ⊙ *Y* ⊆ 𝒢 **by** (*rule odot-𝒢*, *insert XY*, *auto simp*: *𝒢l-def*)
hence *XYD*: *X* ⊙ *Y* ⊆ 𝒢 **by** *auto*
have *finvXY*: *finite* (*v-gs* (*X* ⊙ *Y*)) **by** (*rule finite-v-gs*[*OF XYD*])
have *card* (*v-gs* (*X* ⊙ *Y*)) ≤ *card* (*v-gs X*) * *card* (*v-gs Y*)
using *XY*(*1*−*2*) **by** (*intro card-v-gs-join*, *auto simp*: *𝒢l-def*)
also have . . . ≤ *L* * *L* **using** *X Y* **unfolding** *𝒫L𝒢l-def*
by (*intro mult-mono*, *auto*)
also have . . . = *L*^*2* **by** *algebra*
finally have *card-join*: *card* (*v-gs* (*X* ⊙ *Y*)) ≤ *L*^*2* **.**
with *card-mono*[*OF finvXY v-gs-mono*[*OF joinl-join*]]
have *card*: *card* (*v-gs* (*X* ⊙*l Y*)) ≤ *L*^*2* **by** *simp*
with *PLU-main-n*[*OF sub res*] **have** *n* * (*p* − *1*) ≤ *L*^*2* **by** *simp*
with *p Lm m2* **have** *n*: *n* < *2* * *L*^*2* **by** (*cases n*, *auto*, *cases p* − *1*, *auto*)
have *: (*k* − *1*) ^ *m* > *0* **using** *k l2* **by** *simp*
show *PLU* (*X* ⊙*l Y*) ∈ *𝒫L𝒢l* **unfolding** *PLU* **using** *main* **by** *auto*
thus *X* ⊓ *Y* ∈ *𝒫L𝒢l* **unfolding** *sqcap-def* **.**
let *?r* = *real*
have *2*^*p* * *card* (∂⊓*Neg X Y*) ≤ *2* ^ *p* * *card* (*ACC-cf Z* − *ACC-cf* (*X* ⊙*l Y*))
unfolding *deviate-neg-cap-def PLU sqcap-def*
by (*rule mult-left-mono*, *rule card-mono*[*OF finite-subset*[*OF - finite-ℱ*]], *insert ACC-cf-ℱ*, *force*,
insert ACC-cf-mono[*OF joinl-join*, *of X Y*], *auto*)
also have . . . ≤ (*k* − *1*) ^ *m* * *n* **using** *main* **by** *simp*
also have . . . < (*k* − *1*) ^ *m* * (*2* * *L*^*2*) **unfolding** *mult-less-cancel1* **using** *n* * **by** *simp*
finally have *2* * (*2*^(*p* − *1*) * *card* (∂⊓*Neg X Y*)) < *2* * ((*k* − *1*) ^ *m* * *L*^*2*) **using** *p* **by** (*cases p*, *auto*)
hence *2* ^(*p* − *1*) * *card* (∂⊓*Neg X Y*) < (*k* − *1*)^*m* * *L*^*2* **by** *simp*
hence *?r* (*2* ^(*p* − *1*) * *card* (∂⊓*Neg X Y*)) < (*k* − *1*)^*m* * *L*^*2* **by** *linarith*
thus *card* (∂⊓*Neg X Y*) < (*k* − *1*)^*m* * *L*^*2* / *2*^(*p* − *1*) **by** (*simp add*: *field-simps*)

define *Vs* **where** *Vs* = *v-gs* (*X* ⊙ *Y*) ∩ {*V* . *V* ⊆ [*m*] ∧ *card V* ≥ *Suc l*}
define *C* **where** *C* (*V* :: *nat set*) = (*SOME C*. *C* ⊆ *V* ∧ *card C* = *Suc l*) **for** *V*
define *K* **where** *K C* = { *W*. *W* ⊆ [*m*] − *C* ∧ *card W* = *k* − *Suc l* } **for** *C*
define *merge* **where** *merge C V* = (*C* ∪ *V*)^**2** **for** *C V* :: *nat set*
define *GS* **where** *GS* = { *merge* (*C V*) *W* | *V W*. *V* ∈ *Vs* ∧ *W* ∈ *K* (*C V*)}
{
fix *V*

assume V: $V \in Vs$
hence $card$: $card\ V \geq Suc\ l$ **and** Vm: $V \subseteq [m]$ **unfolding** *Vs-def* **by** *auto*
from *card* **obtain** D **where** C: $D \subseteq V$ **and** $cardV$: $card\ D = Suc\ l$
by (*rule obtain-subset-with-card-n*)
hence $\exists\ C.\ C \subseteq V \wedge card\ C = Suc\ l$ **by** *blast*
from *someI-ex*[*OF this*, *folded C-def*] **have** $*$: $C\ V \subseteq V$ $card\ (C\ V) = Suc\ l$
by *blast+*
with *Vm* **have** *sub*: $C\ V \subseteq [m]$ **by** *auto*
from *finite-subset*[*OF this*] **have** *finCV*: *finite* $(C\ V)$ **unfolding** *numbers-def* **by** *simp*
have $card\ (K\ (C\ V)) = (m - Suc\ l)\ choose\ (k - Suc\ l)$ **unfolding** *K-def*
proof (*subst n-subsets*, (*rule finite-subset*[*of - [m]*], *auto*)[*1*], *rule arg-cong*[*of - - λ x. x choose -*])
show $card\ ([m] - C\ V) = m - Suc\ l$
by (*subst card-Diff-subset*, *insert sub $*$ finCV*, *auto*)
qed
note $*$ *finCV sub this*
} **note** *Vs-C* = *this*
have *finK*: *finite* $(K\ V)$ **for** V **unfolding** *K-def* **by** *auto*
{
fix G
assume G: $G \in POS \cap ACC\ (X \odot Y)$
have $G \in ACC\ (X \odot l\ Y) \cup GS$
proof (*rule ccontr*)
assume $\neg$ *?thesis*
with G **have** G: $G \in POS$ $G \in ACC\ (X \odot Y)$ $G \notin ACC\ (X \odot l\ Y)$
and *contra*: $G \notin GS$ **by** *auto*
from $G(1)$[*unfolded $\mathcal{K}$-def*] **have** $card\ (v\ G) = k \wedge (v\ G)\hat{}\mathbf{2} = G$ **and** *G0*: $G \in \mathcal{G}$
by *auto*
hence *vGk*: $card\ (v\ G) = k$ $(v\ G)\hat{}\mathbf{2} = G$ **by** *auto*
from *G0* **have** *vm*: $v\ G \subseteq [m]$ **by** (*rule v-$\mathcal{G}$*)
from $G(2-3)$[*unfolded ACC-def accepts-def*] **obtain** H
where H: $H \in X \odot Y$ $H \notin X \odot l\ Y$
and *HG*: $H \subseteq G$ **by** *auto*
from *v-mono*[*OF HG*] **have** *vHG*: $v\ H \subseteq v\ G$ **by** *auto*
{
from $H(1)$[*unfolded odot-def*] **obtain** $D\ E$ **where** D: $D \in X$ **and** E: $E \in Y$ **and** *HDE*: $H = D \cup E$
by *force*
from $D\ E\ X\ Y$ **have** *Dl*: $D \in \mathcal{G}l$ $E \in \mathcal{G}l$ **unfolding** *$\mathcal{PLG}l$-def* **by** *auto*
have *Dp*: $D \in \mathcal{G}$ **using** *Dl* **by** (*auto simp*: *$\mathcal{G}l$-def*)
have *Ep*: $E \in \mathcal{G}$ **using** *Dl* **by** (*auto simp*: *$\mathcal{G}l$-def*)
from *Dl HDE* **have** *HD*: $H \in \mathcal{G}$ **unfolding** *$\mathcal{G}l$-def* **by** *auto*
have *HG0*: $H \in \mathcal{G}$ **using** *Dp Ep* **unfolding** *HDE* **by** *auto*
have *HDL*: $H \notin \mathcal{G}l$
proof
assume $H \in \mathcal{G}l$
hence $H \in X \odot l\ Y$

unfolding *odotl-def HDE odot-def* **using** *D E* **by** *blast*
thus *False* **using** *H* **by** *auto*
qed
from *HDL HD* **have** *HGl*: $H \notin \mathcal{G}l$ **unfolding** $\mathcal{G}l$*-def* **by** *auto*
have *vm*: $v\ H \subseteq [m]$ **using** *HG0* **by** (*rule v-*$\mathcal{G}$)
have *lower*: $l <$ *card* $(v\ H)$ **using** *HGl HG0* **unfolding** $\mathcal{G}l$*-def* **by** *auto*
have $v\ H \in Vs$ **unfolding** *Vs-def* **using** *lower vm H* **unfolding** *v-gs-def*
by *auto*
} **note** *in-Vs* = *this*
note *C* = *Vs-C*[*OF this*]
let *?C* = *C* (*v H*)
from *C vHG* **have** *CG*: *?C* $\subseteq$ *v G* **by** *auto*
hence *id*: *v G* = *?C* $\cup$ (*v G* $-$ *?C*) **by** *auto*
from *arg-cong*[*OF this*, *of card*] *vGk*(*1*) *C*
have *card* (*v G* $-$ *?C*) = *k* $-$ *Suc l*
by (*metis CG card-Diff-subset*)
hence *v G* $-$ *?C* $\in$ *K ?C* **unfolding** *K-def* **using** *vm* **by** *auto*
hence *merge ?C* (*v G* $-$ *?C*) $\in$ *GS* **unfolding** *GS-def* **using** *in-Vs* **by** *auto*
also have *merge ?C* (*v G* $-$ *?C*) = *v G*^**2** **unfolding** *merge-def*
by (*rule arg-cong*[*of - - sameprod*], *insert id*, *auto*)
also have ... = *G* **by** *fact*
finally have *G* $\in$ *GS* **.**
with *contra* **show** *False* **..**
qed
}
hence $\partial\sqcap$*Pos X Y* $\subseteq$ (*POS* $\cap$ *ACC* (*X* $\odot$*l Y*) $-$ *ACC* (*X* $\sqcap$ *Y*)) $\cup$ *GS*
unfolding *deviate-pos-cap-def* **by** *auto*
also have *POS* $\cap$ *ACC* (*X* $\odot$*l Y*) $-$ *ACC* (*X* $\sqcap$ *Y*) = {}
proof $-$
have *POS* $-$ *ACC* (*X* $\sqcap$ *Y*) $\subseteq$ *UNIV* $-$ *ACC* (*X* $\odot$*l Y*)
unfolding *sqcap-def* **using** *PLU main* **by** *auto*
thus *?thesis* **by** *auto*
qed
finally have *sub*: $\partial\sqcap$*Pos X Y* $\subseteq$ *GS* **by** *auto*
have *finVs*: *finite Vs* **unfolding** *Vs-def numbers-def* **by** *simp*
let *?Sig* = *Sigma Vs* (λ *V*. *K* (*C V*))
have *GS-def*: *GS* = (λ (*V*,*W*). *merge* (*C V*) *W*) ' *?Sig* **unfolding** *GS-def*
by *auto*
have *finSig*: *finite ?Sig* **using** *finVs finK* **by** *simp*
have *finGS*: *finite GS* **unfolding** *GS-def*
by (*rule finite-imageI*[*OF finSig*])
have *card* ($\partial\sqcap$*Pos X Y*) $\leq$ *card GS* **by** (*rule card-mono*[*OF finGS sub*])
also have ... $\leq$ *card ?Sig* **unfolding** *GS-def*
by (*rule card-image-le*[*OF finSig*])
also have ... = ($\sum$ *a*$\in$*Vs*. *card* (*K* (*C a*)))
by (*rule card-SigmaI*[*OF finVs*], *auto simp*: *finK*)
also have ... = ($\sum$ *a*$\in$*Vs*. (*m* $-$ *Suc l*) *choose* (*k* $-$ *Suc l*)) **using** *Vs-C*
by (*intro sum.cong*, *auto*)
also have ... = ((*m* $-$ *Suc l*) *choose* (*k* $-$ *Suc l*)) $*$ *card Vs*

by *simp*
also have ... $\leq$ ((*m* $-$ *Suc l*) *choose* (*k* $-$ *Suc l*)) $*$ *L*ˆ*2*
proof (*rule mult-left-mono*)
have *card Vs* $\leq$ *card* (*v-gs* (*X* $\odot$ *Y*))
by (*rule card-mono*[*OF finvXY*], *auto simp*: *Vs-def*)
also have ... $\leq$ *L*ˆ*2* **by** *fact*
finally show *card Vs* $\leq$ *L*ˆ*2* **.**
qed *simp*
finally show *card* ($\partial\sqcap$*Pos X Y*) $\leq$ ((*m* $-$ *l* $-$ *1*) *choose* (*k* $-$ *l* $-$ *1*)) $*$ *L*ˆ*2*
by *simp*
qed
end

4.7 Formalism

Fix a variable set of cardinality m over 2.

locale *forth-assumptions* = *third-assumptions* +
fixes $\mathcal{V}$ :: *'a set* **and** π :: *'a* $\Rightarrow$ *vertex set*
assumes *cV*: *card* $\mathcal{V}$ = (*m choose 2*)
and *bij-betw-π*: *bij-betw* π $\mathcal{V}$ ([*m*]ˆ**2**)
begin

definition *n* **where** *n* = (*m choose 2*)

the formulas over the fixed variable set

definition $\mathcal{A}$ :: *'a mformula set* **where**
$\mathcal{A}$ = { φ. *vars* $\varphi \subseteq \mathcal{V}$}

lemma $\mathcal{A}$*-simps*[*simp*]:
FALSE $\in \mathcal{A}$
(*Var x* $\in \mathcal{A}$) = ($x \in \mathcal{V}$)
(*Conj* φ ψ $\in \mathcal{A}$) = ($\varphi \in \mathcal{A} \wedge \psi \in \mathcal{A}$)
(*Disj* φ ψ $\in \mathcal{A}$) = ($\varphi \in \mathcal{A} \wedge \psi \in \mathcal{A}$)
by (*auto simp*: $\mathcal{A}$*-def*)

lemma *inj-on-π*: *inj-on* π $\mathcal{V}$
using *bij-betw-π* **by** (*metis bij-betw-imp-inj-on*)

lemma *πm2*[*simp*,*intro*]: $x \in \mathcal{V} \Longrightarrow \pi\ x \in$ [*m*]ˆ**2**
using *bij-betw-π* **by** (*rule bij-betw-apply*)

lemma *card-v-π*[*simp*,*intro*]: **assumes** $x \in \mathcal{V}$
shows *card* (*v* {π *x*}) = *2*
proof $-$
from *πm2*[*OF assms*] **have** *mem*: $\pi\ x \in$ [*m*]ˆ**2** **by** *auto*
from *this*[*unfolded binprod-def*] **obtain** *a b* **where** *π*: $\pi\ x = \{a,b\}$ **and** *diff*: $a \neq b$
by *auto*
hence *v* {π *x*} = {*a*,*b*} **unfolding** *v-def* **by** *auto*

```
  thus ?thesis using diff by simp
qed

lemma π-singleton[simp,intro]: assumes x ∈ 𝒱
  shows {π x} ∈ 𝒢
    {{π x}} ∈ 𝒫L𝒢l
  using assms L3 l2
  by (auto simp: 𝒢-def 𝒫L𝒢l-def v-gs-def 𝒢l-def)

lemma empty-𝒫L𝒢l[simp,intro]: {} ∈ 𝒫L𝒢l
  by (auto simp: 𝒢-def 𝒫L𝒢l-def v-gs-def 𝒢l-def)

fun SET :: 'a mformula ⇒ graph set where
  SET FALSE = {}
| SET (Var x) = {{π x}}
| SET (Disj φ ψ) = SET φ ∪ SET ψ
| SET (Conj φ ψ) = SET φ ⊙ SET ψ

lemma ACC-cf-SET[simp]:
  ACC-cf (SET (Var x)) = {f ∈ ℱ. π x ∈ C f}
  ACC-cf (SET FALSE) = {}
  ACC-cf (SET (Disj φ ψ)) = ACC-cf (SET φ) ∪ ACC-cf (SET ψ)
  ACC-cf (SET (Conj φ ψ)) = ACC-cf (SET φ) ∩ ACC-cf (SET ψ)
  using ACC-cf-odot
  by (auto simp: ACC-cf-union ACC-cf-empty, auto simp: ACC-cf-def accepts-def)

lemma ACC-SET[simp]:
  ACC (SET (Var x)) = {G ∈ 𝒢. π x ∈ G}
  ACC (SET FALSE) = {}
  ACC (SET (Disj φ ψ)) = ACC (SET φ) ∪ ACC (SET ψ)
  ACC (SET (Conj φ ψ)) = ACC (SET φ) ∩ ACC (SET ψ)
  by (auto simp: ACC-union ACC-odot, auto simp: ACC-def accepts-def)

lemma SET-𝒢: φ ∈ tf-mformula ⟹ φ ∈ 𝒜 ⟹ SET φ ⊆ 𝒢
proof (induct φ rule: tf-mformula.induct)
  case (tf-Conj φ ψ)
  hence SET φ ⊆ 𝒢 SET ψ ⊆ 𝒢 by auto
  from odot-𝒢[OF this] show ?case by simp
qed auto

fun APR :: 'a mformula ⇒ graph set where
  APR FALSE = {}
| APR (Var x) = {{π x}}
| APR (Disj φ ψ) = APR φ ⊔ APR ψ
| APR (Conj φ ψ) = APR φ ⊓ APR ψ

lemma APR: φ ∈ tf-mformula ⟹ φ ∈ 𝒜 ⟹ APR φ ∈ 𝒫L𝒢l
  by (induct φ rule: tf-mformula.induct, auto intro!: sqcup sqcap)
```

definition *ACC-cf-mf* :: *'a mformula* ⇒ *colorf set* **where**
ACC-cf-mf φ = *ACC-cf* (*SET* φ)

definition *ACC-mf* :: *'a mformula* ⇒ *graph set* **where**
ACC-mf φ = *ACC* (*SET* φ)

definition *deviate-pos* :: *'a mformula* ⇒ *graph set* (‹∂*Pos*›) **where**
∂*Pos* φ = *POS* ∩ *ACC-mf* φ − *ACC* (*APR* φ)

definition *deviate-neg* :: *'a mformula* ⇒ *colorf set* (‹∂*Neg*›) **where**
∂*Neg* φ = *ACC-cf* (*APR* φ) − *ACC-cf-mf* φ

Lemma 11.1

lemma *deviate-subset-Disj*:
∂*Pos* (*Disj* φ ψ) ⊆ ∂⊔*Pos* (*APR* φ) (*APR* ψ) ∪ ∂*Pos* φ ∪ ∂*Pos* ψ
∂*Neg* (*Disj* φ ψ) ⊆ ∂⊔*Neg* (*APR* φ) (*APR* ψ) ∪ ∂*Neg* φ ∪ ∂*Neg* ψ
unfolding
deviate-pos-def deviate-pos-cup-def
deviate-neg-def deviate-neg-cup-def
ACC-cf-mf-def ACC-cf-SET ACC-cf-union
ACC-mf-def ACC-SET ACC-union
by *auto*

Lemma 11.2

lemma *deviate-subset-Conj*:
∂*Pos* (*Conj* φ ψ) ⊆ ∂⊓*Pos* (*APR* φ) (*APR* ψ) ∪ ∂*Pos* φ ∪ ∂*Pos* ψ
∂*Neg* (*Conj* φ ψ) ⊆ ∂⊓*Neg* (*APR* φ) (*APR* ψ) ∪ ∂*Neg* φ ∪ ∂*Neg* ψ
unfolding
deviate-pos-def deviate-pos-cap-def
ACC-mf-def ACC-SET ACC-odot
deviate-neg-def deviate-neg-cap-def
ACC-cf-mf-def ACC-cf-SET ACC-cf-odot
by *auto*

lemmas *deviate-subset* = *deviate-subset-Disj deviate-subset-Conj*

lemma *deviate-finite*:
finite (∂*Pos* φ)
finite (∂*Neg* φ)
finite (∂⊔*Pos A B*)
finite (∂⊔*Neg A B*)
finite (∂⊓*Pos A B*)
finite (∂⊓*Neg A B*)
unfolding
deviate-pos-def deviate-pos-cup-def deviate-pos-cap-def
deviate-neg-def deviate-neg-cup-def deviate-neg-cap-def
by (*intro finite-subset*[*OF - finite-POS-NEG*], *auto*)+

Lemma 12

lemma *no-deviation*[*simp*]:
∂*Pos FALSE* = {}
∂*Neg FALSE* = {}
∂*Pos* (*Var x*) = {}
∂*Neg* (*Var x*) = {}
unfolding *deviate-pos-def deviate-neg-def*
by (*auto simp add*: *ACC-cf-mf-def ACC-mf-def*)

Lemma 12.1-2

fun *approx-pos* **where**
approx-pos (*Conj phi psi*) = ∂⊓*Pos* (*APR phi*) (*APR psi*)
| *approx-pos* - = {}

fun *approx-neg* **where**
approx-neg (*Conj phi psi*) = ∂⊓*Neg* (*APR phi*) (*APR psi*)
| *approx-neg* (*Disj phi psi*) = ∂⊔*Neg* (*APR phi*) (*APR psi*)
| *approx-neg* - = {}

lemma *finite-approx-pos*: *finite* (*approx-pos* φ)
by (*cases* φ, *auto intro*: *deviate-finite*)

lemma *finite-approx-neg*: *finite* (*approx-neg* φ)
by (*cases* φ, *auto intro*: *deviate-finite*)

lemma *card-deviate-Pos*: **assumes** *phi*: φ ∈ *tf-mformula* φ ∈ $\mathcal{A}$
shows *card* (∂*Pos* φ) ≤ *cs* φ ∗ L^2 ∗ ((*m* − *l* − *1*) *choose* (*k* − *l* − *1*))
proof −
let *?Pos* = λ φ. ⋃ (*approx-pos* ‘ *SUB* φ)
have ∂*Pos* φ ⊆ *?Pos* φ
using *phi*
proof (*induct* φ *rule*: *tf-mformula.induct*)
case (*tf-Disj* φ ψ)
from *tf-Disj* **have** ∗: φ ∈ *tf-mformula* ψ ∈ *tf-mformula* φ ∈ $\mathcal{A}$ ψ ∈ $\mathcal{A}$ **by** *auto*
note *IH* = *tf-Disj*(*2*)[*OF* ∗(*3*)] *tf-Disj*(*4*)[*OF* ∗(*4*)]
have ∂*Pos* (*Disj* φ ψ) ⊆ ∂⊔*Pos* (*APR* φ) (*APR* ψ) ∪ ∂*Pos* φ ∪ ∂*Pos* ψ
by (*rule deviate-subset*)
also have ∂⊔*Pos* (*APR* φ) (*APR* ψ) = {}
by (*rule deviate-pos-cup*; *intro APR* ∗)
also have . . . ∪ ∂*Pos* φ ∪ ∂*Pos* ψ ⊆ *?Pos* φ ∪ *?Pos* ψ **using** *IH* **by** *auto*
also have . . . ⊆ *?Pos* (*Disj* φ ψ) ∪ *?Pos* (*Disj* φ ψ)
by (*intro Un-mono*, *auto*)
finally show *?case* **by** *simp*
next
case (*tf-Conj* φ ψ)
from *tf-Conj* **have** ∗: φ ∈ $\mathcal{A}$ ψ ∈ $\mathcal{A}$
by (*auto intro*: *tf-mformula.intros*)
note *IH* = *tf-Conj*(*2*)[*OF* ∗(*1*)] *tf-Conj*(*4*)[*OF* ∗(*2*)]
have ∂*Pos* (*Conj* φ ψ) ⊆ ∂⊓*Pos* (*APR* φ) (*APR* ψ) ∪ ∂*Pos* φ ∪ ∂*Pos* ψ
by (*rule deviate-subset*)

also have . . . ⊆ ∂⊓*Pos* (*APR* φ) (*APR* ψ) ∪ *?Pos* φ ∪ *?Pos* ψ **using** *IH* **by** *auto*
also have . . . ⊆ *?Pos* (*Conj* φ ψ) ∪ *?Pos* (*Conj* φ ψ) ∪ *?Pos* (*Conj* φ ψ)
by (*intro Un-mono*, *insert* ∗, *auto*)
finally show *?case* **by** *simp*
qed *auto*
from *card-mono*[*OF finite-UN-I*[*OF finite-SUB finite-approx-pos*] *this*]
have *card* (∂*Pos* φ) ≤ *card* (⋃ (*approx-pos* ‘ *SUB* φ)) **by** *simp*
also have . . . ≤ (∑ *i*∈*SUB* φ. *card* (*approx-pos i*))
by (*rule card-UN-le*[*OF finite-SUB*])
also have . . . ≤ (∑ *i*∈*SUB* φ. L^2 ∗ ((*m* − *l* − *1*) *choose* (*k* − *l* − *1*)))
proof (*rule sum-mono*, *goal-cases*)
case (*1 psi*)
from *phi 1* **have** *psi*: *psi* ∈ *tf-mformula psi* ∈ 𝒜
by (*induct* φ *rule*: *tf-mformula.induct*, *auto intro*: *tf-mformula.intros*)
show *?case*
proof (*cases psi*)
case (*Conj phi1 phi2*)
from *psi this* **have** ∗: *phi1* ∈ *tf-mformula phi1* ∈ 𝒜 *phi2* ∈ *tf-mformula phi2* ∈ 𝒜
by (*cases rule*: *tf-mformula.cases*, *auto*)+
from *deviate-pos-cap*[*OF APR*[*OF* ∗(*1*−*2*)] *APR*[*OF* ∗(*3*−*4*)]]
show *?thesis* **unfolding** *Conj* **by** (*simp add*: *ac-simps*)
qed *auto*
qed
also have . . . = *cs* φ ∗ L^2 ∗ ((*m* − *l* − *1*) *choose* (*k* − *l* − *1*)) **unfolding** *cs-def* **by** *simp*
finally show *card* (∂*Pos* φ) ≤ *cs* φ ∗ L^2 ∗ (*m* − *l* − *1 choose* (*k* − *l* − *1*)) **by** *simp*
qed

lemma *card-deviate-Neg*: **assumes** *phi*: φ ∈ *tf-mformula* φ ∈ 𝒜
shows *card* (∂*Neg* φ) ≤ *cs* φ ∗ L^2 ∗ (*k* − *1*)^*m* / *2*^(*p* − *1*)
proof −
let *?r* = *real*
let *?Neg* = λ φ. ⋃ (*approx-neg* ‘ *SUB* φ)
have ∂*Neg* φ ⊆ *?Neg* φ
using *phi*
proof (*induct* φ *rule*: *tf-mformula.induct*)
case (*tf-Disj* φ ψ)
from *tf-Disj* **have** ∗: φ ∈ *tf-mformula* ψ ∈ *tf-mformula* φ ∈ 𝒜 ψ ∈ 𝒜 **by** *auto*
note *IH* = *tf-Disj*(*2*)[*OF* ∗(*3*)] *tf-Disj*(*4*)[*OF* ∗(*4*)]
have ∂*Neg* (*Disj* φ ψ) ⊆ ∂⊔*Neg* (*APR* φ) (*APR* ψ) ∪ ∂*Neg* φ ∪ ∂*Neg* ψ
by (*rule deviate-subset*)
also have . . . ⊆ ∂⊔*Neg* (*APR* φ) (*APR* ψ) ∪ *?Neg* φ ∪ *?Neg* ψ **using** *IH* **by** *auto*
also have . . . ⊆ *?Neg* (*Disj* φ ψ) ∪ *?Neg* (*Disj* φ ψ) ∪ *?Neg* (*Disj* φ ψ)
by (*intro Un-mono*, *auto*)
finally show *?case* **by** *simp*

next
case (*tf-Conj* φ ψ)
from *tf-Conj* **have** ∗: φ ∈ 𝒜 ψ ∈ 𝒜
by (*auto intro*: *tf-mformula.intros*)
note *IH* = *tf-Conj*(*2*)[*OF* ∗(*1*)] *tf-Conj*(*4*)[*OF* ∗(*2*)]
have ∂*Neg* (*Conj* φ ψ) ⊆ ∂⊓*Neg* (*APR* φ) (*APR* ψ) ∪ ∂*Neg* φ ∪ ∂*Neg* ψ
by (*rule deviate-subset*)
also have ... ⊆ ∂⊓*Neg* (*APR* φ) (*APR* ψ) ∪ *?Neg* φ ∪ *?Neg* ψ **using** *IH* **by** *auto*
also have ... ⊆ *?Neg* (*Conj* φ ψ) ∪ *?Neg* (*Conj* φ ψ) ∪ *?Neg* (*Conj* φ ψ)
by (*intro Un-mono*, *auto*)
finally show *?case* **by** *simp*
qed *auto*
hence ∂*Neg* φ ⊆ ⋃ (*approx-neg* ‘ *SUB* φ) **by** *auto*
from *card-mono*[*OF finite-UN-I*[*OF finite-SUB finite-approx-neg*] *this*]
have *card* (∂*Neg* φ) ≤ *card* (⋃ (*approx-neg* ‘ *SUB* φ)) **.**
also have ... ≤ (∑ *i*∈*SUB* φ. *card* (*approx-neg i*))
by (*rule card-UN-le*[*OF finite-SUB*])
finally have *?r* (*card* (∂*Neg* φ)) ≤ (∑ *i*∈*SUB* φ. *card* (*approx-neg i*)) **by** *linarith*
also have ... = (∑ *i*∈*SUB* φ. *?r* (*card* (*approx-neg i*))) **by** *simp*
also have ... ≤ (∑ *i*∈*SUB* φ. *L*^*2* ∗ (*k* − *1*)^*m* / *2*^(*p* − *1*))
proof (*rule sum-mono*, *goal-cases*)
case (*1 psi*)
from *phi 1* **have** *psi*: *psi* ∈ *tf-mformula psi* ∈ 𝒜
by (*induct* φ *rule*: *tf-mformula.induct*, *auto intro*: *tf-mformula.intros*)
show *?case*
proof (*cases psi*)
case (*Conj phi1 phi2*)
from *psi this* **have** ∗: *phi1* ∈ *tf-mformula phi1* ∈ 𝒜 *phi2* ∈ *tf-mformula phi2* ∈ 𝒜
by (*cases rule*: *tf-mformula.cases*, *auto*)+
from *deviate-neg-cap*[*OF APR*[*OF* ∗(*1*−*2*)] *APR*[*OF* ∗(*3*−*4*)]]
show *?thesis* **unfolding** *Conj* **by** (*simp add*: *ac-simps*)
next
case (*Disj phi1 phi2*)
from *psi this* **have** ∗: *phi1* ∈ *tf-mformula phi1* ∈ 𝒜 *phi2* ∈ *tf-mformula phi2* ∈ 𝒜
by (*cases rule*: *tf-mformula.cases*, *auto*)+
from *deviate-neg-cup*[*OF APR*[*OF* ∗(*1*−*2*)] *APR*[*OF* ∗(*3*−*4*)]]
have *card* (*approx-neg psi*) ≤ ((*L* ∗ *1*) ∗ (*k* − *1*) ^ *m*) / *2* ^ (*p* − *1*)
unfolding *Disj* **by** (*simp add*: *ac-simps*)
also have ... ≤ ((*L* ∗ *L*) ∗ (*k* − *1*) ^ *m*) / *2* ^ (*p* − *1*)
by (*intro divide-right-mono*, *unfold of-nat-le-iff*, *intro mult-mono*, *insert L3*, *auto*)
finally show *?thesis* **unfolding** *power2-eq-square* **by** *simp*
qed *auto*
qed
also have ... = *cs* φ ∗ *L*^*2* ∗ (*k* − *1*)^*m* / *2*^(*p* − *1*) **unfolding** *cs-def* **by** *simp*

finally show *card* $(\partial Neg\ \varphi) \leq cs\ \varphi * L^2 * (k - 1)\hat{}m\ /\ 2\hat{}(p - 1)$ **.**
qed

Lemma 12.3

lemma *ACC-cf-non-empty-approx*: **assumes** *phi*: $\varphi \in$ *tf-mformula* $\varphi \in \mathcal{A}$
and *ne*: *APR* $\varphi \neq \{\}$
shows *card* (*ACC-cf* (*APR* φ)) $>$ $(k - 1)\hat{}m\ /\ 3$
proof $-$
from *ne* **obtain** *E* :: *graph* **where** *Ephi*: $E \in$ *APR* φ
by (*auto simp*: *ACC-def accepts-def*)
from *APR*[*OF phi*, *unfolded* $\mathcal{PLG}l$*-def*] *Ephi*
have *EDl*: $E \in \mathcal{G}l$ **by** *auto*
hence *vEl*: *card* (*v E*) $\leq l$ **and** *ED*: $E \in \mathcal{G}$
unfolding $\mathcal{G}l$*-def* $\mathcal{G}l$*-def* **by** *auto*
have *E*: $E \in \mathcal{G}$ **using** *ED*[*unfolded* $\mathcal{G}l$*-def*] **by** *auto*
have *sub*: *v E* $\subseteq [m]$ **by** (*rule v-*$\mathcal{G}$[*OF E*])
have $l \leq$ *card* $[m]$ **using** *lm* **by** *auto*
from *exists-subset-between*[*OF vEl this sub finite-numbers*]
obtain *V* **where** *V*: *v E* $\subseteq V$ $V \subseteq [m]$ *card* $V = l$ **by** *auto*
from *finite-subset*[*OF V(2)*] **have** *finV*: *finite V* **by** *auto*
have *finPart*: *finite A* **if** $A \subseteq \{P.$ *partition-on* $[n]$ $P\}$ **for** *n A*
by (*rule finite-subset*[*OF that finitely-many-partition-on*], *simp*)
have *finmv*: *finite* $([m] - V)$ **using** *finite-numbers*[*of m*] **by** *auto*
have *finK*: *finite* $[k - 1]$ **unfolding** *numbers-def* **by** *auto*
define *F* **where** $F = \{f \in [m] \to_E [k - 1].$ *inj-on f V*$\}$
have *FF*: $F \subseteq \mathcal{F}$ **unfolding** $\mathcal{F}$*-def F-def* **by** *auto*
{
fix *f*
assume *f*: $f \in F$
{
from *this*[*unfolded F-def*]
have *f*: $f \in [m] \to_E [k - 1]$ **and** *inj*: *inj-on f V* **by** *auto*
from *V l2* **have** *2*: *card* $V \geq 2$ **by** *auto*
then obtain *x* **where** *x*: $x \in V$ **by** (*cases* $V = \{\}$, *auto*)
have *card* $V =$ *card* $(V - \{x\}) + 1$ **using** *x finV*
by (*metis One-nat-def add.right-neutral add-Suc-right card-Suc-Diff1*)
with *2* **have** *card* $(V - \{x\}) > 0$ **by** *auto*
hence $V - \{x\} \neq \{\}$ **by** *fastforce*
then obtain *y* **where** *y*: $y \in V$ **and** *diff*: $x \neq y$ **by** *auto*
from *inj diff x y* **have** *neq*: $f\ x \neq f\ y$ **by** (*auto simp*: *inj-on-def*)
from *x y diff V* **have** $\{x, y\} \in [m]\hat{}\mathbf{2}$ **unfolding** *sameprod-altdef* **by** *auto*
with *neq* **have** $\{x,y\} \in C\ f$ **unfolding** *C-def* **by** *auto*
hence $C\ f \neq \{\}$ **by** *auto*
}
with *NEG-*$\mathcal{G}$ *FF f* **have** *CfG*: $C\ f \in \mathcal{G}$ $C\ f \neq \{\}$ **by** (*auto simp*: *NEG-def*)
have $E \subseteq C\ f$
proof
fix *e*
assume *eE*: $e \in E$

with *E*[*unfolded* $\mathcal{G}$*-def*] **have** *em*: *e* ∈ [*m*]^**2** **by** *auto*
then obtain *x y* **where** *e*: *e* = {*x*,*y*} *x* ≠ *y* {*x*,*y*} ⊆ [*m*]
and *card*: *card e* = *2*
unfolding *binprod-def* **by** *auto*
from *v-mem-sub*[*OF card eE*]
have {*x*,*y*} ⊆ *v E* **using** *e* **by** *auto*
hence {*x*,*y*} ⊆ *V* **using** *V* **by** *auto*
hence *f x* ≠ *f y* **using** *e*(*2*) *f*[*unfolded F-def*] **by** (*auto simp*: *inj-on-def*)
thus *e* ∈ *C f* **unfolding** *C-def* **using** *em e* **by** *auto*
qed
with *Ephi CfG* **have** *APR* φ ⊩ *C f*
unfolding *accepts-def* **by** *auto*
hence *f* ∈ *ACC-cf* (*APR* φ) **using** *CfG f FF* **unfolding** *ACC-cf-def* **by** *auto*
}
with *FF* **have** *sub*: *F* ⊆ *ACC-cf* (*APR* φ) **by** *auto*
from *card-mono*[*OF finite-subset*[*OF - finite-ACC*] *this*]
have *approx*: *card F* ≤ *card* (*ACC-cf* (*APR* φ)) **by** *auto*
from *card-inj-on-subset-funcset*[*OF finite-numbers finK V*(*2*), *unfolded card-numbers V*(*3*),
folded F-def]
have *real* (*card F*) = (*real* (*k* − *1*)) ^ (*m* − *l*) ∗ *prod* (λ *i*. *real* (*k* − *1* − *i*)) {*0*..<*l*}
by *simp*
also have ... > (*real* (*k* − *1*)) ^ *m* / *3*
by (*rule approximation1*)
finally have *cardF*: *card F* > (*k* − *1*) ^ *m* / *3* **by** *simp*
with *approx* **show** *?thesis* **by** *simp*
qed

Theorem 13

lemma *theorem-13*: **assumes** *phi*: φ ∈ *tf-mformula* φ ∈ $\mathcal{A}$
and *sub*: *POS* ⊆ *ACC-mf* φ *ACC-cf-mf* φ = {}
shows *cs* φ > *k powr* (*4* / *7* ∗ *sqrt k*)
proof −
let *?r* = *real* :: *nat* ⇒ *real*
have *cs* φ > ((*m* − *l*) / *k*)^*l* / (*6* ∗ *L*^*2*)
proof (*cases POS* ∩ *ACC* (*APR* φ) = {})
case *empty*: *True*
have ∂*Pos* φ = *POS* ∩ *ACC-mf* φ − *ACC* (*APR* φ) **unfolding** *deviate-pos-def* **by** *auto*
also have ... = *POS* − *ACC* (*APR* φ) **using** *sub* **by** *blast*
also have ... = *POS* **using** *empty* **by** *auto*
finally have *id*: ∂*Pos* φ = *POS* **by** *simp*
have *m choose k* = *card POS* **by** (*simp add*: *card-POS*)
also have ... = *card* (∂*Pos* φ) **unfolding** *id* **by** *simp*
also have ... ≤ *cs* φ ∗ L^2 ∗ (*m* − *l* − *1* *choose* (*k* − *l* − *1*)) **using** *card-deviate-Pos*[*OF phi*] **by** *auto*
finally have *m choose k* ≤ *cs* φ ∗ L^2 ∗ (*m* − *l* − *1* *choose* (*k* − *l* − *1*))
by *simp*

from *approximation2*[*OF this*]
show $((m - l) / k)\hat{~}l / (6 * L\hat{~}2) < cs\ \varphi$ **by** *simp*
next
case *False*
have $POS \cap ACC\ (APR\ \varphi) \neq \{\}$ **by** *fact*
hence *nempty*: $APR\ \varphi \neq \{\}$ **by** *auto*
have $card\ (\partial Neg\ \varphi) = card\ (ACC\text{-}cf\ (APR\ \varphi) - ACC\text{-}cf\text{-}mf\ \varphi)$ **unfolding** *deviate-neg-def* **by** *auto*
also have $\ldots = card\ (ACC\text{-}cf\ (APR\ \varphi))$ **using** *sub* **by** *auto*
also have $\ldots > (k - 1)\hat{~}m / 3$ **using** *ACC-cf-non-empty-approx*[*OF phi nempty*] **.**
finally have $(k - 1)\hat{~}m / 3 < card\ (\partial Neg\ \varphi)$ **.**
also have $\ldots \leq cs\ \varphi * L^2 * (k - 1)\ \hat{~} m / 2\ \hat{~}(p - 1)$
using *card-deviate-Neg*[*OF phi*] *sub* **by** *auto*
finally have $(k - 1)\hat{~}m / 3 < (cs\ \varphi * (L^2 * (k - 1)\ \hat{~} m)) / 2\ \hat{~}(p - 1)$ **by** *simp*
from *approximation3*[*OF this*] **show** *?thesis* **.**
qed
hence *part1*: $cs\ \varphi > ((m - l) / k)\hat{~}l / (6 * L\hat{~}2)$ **.**
from *approximation4*[*OF this*] **show** *?thesis* **using** *k2* **by** *simp*
qed

Definition 14

definition *eval-g* :: *'a VAS* ⇒ *graph* ⇒ *bool* **where**
$eval\text{-}g\ \vartheta\ G = (\forall\ v \in \mathcal{V}.\ (\pi\ v \in G \longrightarrow \vartheta\ v))$

definition *eval-gs* :: *'a VAS* ⇒ *graph set* ⇒ *bool* **where**
$eval\text{-}gs\ \vartheta\ X = (\exists\ G \in X.\ eval\text{-}g\ \vartheta\ G)$

lemmas *eval-simps* = *eval-g-def eval-gs-def eval.simps*

lemma *eval-gs-union*:
$eval\text{-}gs\ \vartheta\ (X \cup Y) = (eval\text{-}gs\ \vartheta\ X \vee eval\text{-}gs\ \vartheta\ Y)$
by (*auto simp*: *eval-gs-def*)

lemma *eval-gs-odot*: **assumes** $X \subseteq \mathcal{G}\ Y \subseteq \mathcal{G}$
shows $eval\text{-}gs\ \vartheta\ (X \odot Y) = (eval\text{-}gs\ \vartheta\ X \wedge eval\text{-}gs\ \vartheta\ Y)$
proof
assume $eval\text{-}gs\ \vartheta\ (X \odot Y)$
from *this*[*unfolded eval-gs-def*] **obtain** *DE* **where** *DE*: $DE \in X \odot Y$
and *eval*: $eval\text{-}g\ \vartheta\ DE$ **by** *auto*
from *DE*[*unfolded odot-def*] **obtain** *D E* **where** *id*: $DE = D \cup E$ **and** *DE*: $D \in X\ E \in Y$
by *auto*
from *eval* **have** $eval\text{-}g\ \vartheta\ D\ eval\text{-}g\ \vartheta\ E$ **unfolding** *id eval-g-def*
by *auto*
with *DE* **show** $eval\text{-}gs\ \vartheta\ X \wedge eval\text{-}gs\ \vartheta\ Y$ **unfolding** *eval-gs-def* **by** *auto*
next

assume *eval-gs* ϑ $X \wedge$ *eval-gs* ϑ Y
then obtain D E **where** *DE*: $D \in X$ $E \in Y$ **and** *eval*: *eval-g* ϑ D *eval-g* ϑ E
unfolding *eval-gs-def* **by** *auto*
from *DE assms* **have** *D*: $D \in \mathcal{G}$ $E \in \mathcal{G}$ **by** *auto*
let $?U = D \cup E$
from *eval* **have** *eval*: *eval-g* ϑ $?U$
unfolding *eval-g-def* **by** *auto*
from *DE* **have** *1*: $?U \in X \odot Y$ **unfolding** *odot-def* **by** *auto*
with *1 eval* **show** *eval-gs* ϑ $(X \odot Y)$ **unfolding** *eval-gs-def* **by** *auto*
qed

Lemma 15

lemma *eval-set*: **assumes** *phi*: $\varphi \in$ *tf-mformula* $\varphi \in \mathcal{A}$
shows *eval* ϑ $\varphi =$ *eval-gs* ϑ $(SET\ \varphi)$
using *phi*
proof (*induct* φ *rule*: *tf-mformula.induct*)
case *tf-False*
then show *?case* **unfolding** *eval-simps* **by** *simp*
next
case (*tf-Var x*)
then show *?case* **using** *inj-on-*π **unfolding** *eval-simps*
by (*auto simp add*: *inj-on-def*)
next
case (*tf-Disj* $\varphi 1$ $\varphi 2$)
thus *?case* **by** (*auto simp*: *eval-gs-union*)
next
case (*tf-Conj* $\varphi 1$ $\varphi 2$)
thus *?case* **by** (*simp*, *intro eval-gs-odot*[*symmetric*]; *intro SET-*$\mathcal{G}$, *auto*)
qed

definition ϑ_g :: *graph* $\Rightarrow$ *'a VAS* **where**
ϑ_g G $x = (x \in \mathcal{V} \wedge \pi\ x \in G)$

From here on we deviate from Gordeev's paper as we do not use positive bases, but a more direct approach.

lemma *eval-ACC*: **assumes** *phi*: $\varphi \in$ *tf-mformula* $\varphi \in \mathcal{A}$
and *G*: $G \in \mathcal{G}$
shows *eval* $(\vartheta_g\ G)$ $\varphi = (G \in ACC\text{-}mf\ \varphi)$
using *phi* **unfolding** *ACC-mf-def*
proof (*induct* φ *rule*: *tf-mformula.induct*)
case (*tf-Var x*)
thus *?case* **by** (*auto simp*: *ACC-def G accepts-def* ϑ_g*-def*)
next
case (*tf-Disj phi psi*)
thus *?case* **by** (*auto simp*: *ACC-union*)
next
case (*tf-Conj phi psi*)
thus *?case* **by** (*auto simp*: *ACC-odot*)
qed *simp*

lemma *CLIQUE-solution-imp-POS-sub-ACC*: **assumes** *solution*: $\forall\ G \in \mathcal{G}.\ G \in$ *CLIQUE* $\longleftrightarrow$ *eval* $(\vartheta_g\ G)\ \varphi$
  and *tf*: $\varphi \in$ *tf-mformula*
  and *phi*: $\varphi \in \mathcal{A}$
 shows *POS* $\subseteq$ *ACC-mf* φ
proof
 fix *G*
 assume *POS*: $G \in$ *POS*
 with *POS-*$\mathcal{G}$ **have** *G*: $G \in \mathcal{G}$ **by** *auto*
 with *POS solution POS-CLIQUE*
 have *eval* $(\vartheta_g\ G)\ \varphi$ **by** *auto*
 thus $G \in$ *ACC-mf* φ **unfolding** *eval-ACC*[*OF tf phi G*] **.**
qed

lemma *CLIQUE-solution-imp-ACC-cf-empty*: **assumes** *solution*: $\forall\ G \in \mathcal{G}.\ G \in$ *CLIQUE* $\longleftrightarrow$ *eval* $(\vartheta_g\ G)\ \varphi$
  and *tf*: $\varphi \in$ *tf-mformula*
  and *phi*: $\varphi \in \mathcal{A}$
 shows *ACC-cf-mf* $\varphi = \{\}$
proof (*rule ccontr*)
 assume $\neg$ *?thesis*
 from *this*[*unfolded ACC-cf-mf-def ACC-cf-def*]
 obtain *F* **where** *F*: $F \in \mathcal{F}$ *SET* $\varphi \Vdash C\ F$ **by** *auto*
 define *G* **where** $G = C\ F$
 have *NEG*: $G \in$ *NEG* **unfolding** *NEG-def G-def* **using** *F* **by** *auto*
 hence $G \notin$ *CLIQUE* **using** *CLIQUE-NEG* **by** *auto*
 have *GG*: $G \in \mathcal{G}$ **unfolding** *G-def* **using** *F*
  using *G-def NEG NEG-*$\mathcal{G}$ **by** *blast*
 have *GAcc*: *SET* $\varphi \Vdash G$ **using** *F*[*folded G-def*] **by** *auto*
 then obtain *D* :: *graph* **where**
  D: $D \in$ *SET* φ **and** *sub*: $D \subseteq G$
  unfolding *accepts-def* **by** *blast*
 from *SET-*$\mathcal{G}$[*OF tf phi*] *D*
 have *DG*: $D \in \mathcal{G}$ **by** *auto*
 have *eval*: *eval* $(\vartheta_g\ D)\ \varphi$ **unfolding** *eval-set*[*OF tf phi*] *eval-gs-def*
  by (*intro bexI*[*OF - D*], *unfold eval-g-def*, *insert DG*, *auto simp*: ϑ_g*-def*)
 hence $D \in$ *CLIQUE* **using** *solution*[*rule-format*, *OF DG*] **by** *auto*
 hence $G \in$ *CLIQUE* **using** *GG sub* **unfolding** *CLIQUE-def* **by** *blast*
 with ‹$G \notin$ *CLIQUE*› **show** *False* **by** *auto*
qed

4.8 Conclusion

Theorem 22

We first consider monotone formulas without TRUE.

theorem *Clique-not-solvable-by-small-tf-mformula*: **assumes** *solution*: $\forall\ G \in \mathcal{G}.$ $G \in$ *CLIQUE* $\longleftrightarrow$ *eval* $(\vartheta_g\ G)\ \varphi$

and *tf*: $\varphi \in$ *tf-mformula*
and *phi*: $\varphi \in \mathcal{A}$
shows *cs* $\varphi >$ *k powr* (*4* / *7* ∗ *sqrt k*)
proof −
from *CLIQUE-solution-imp-POS-sub-ACC*[*OF solution tf phi*] **have** *POS*: *POS* ⊆ *ACC-mf* φ **.**
from *CLIQUE-solution-imp-ACC-cf-empty*[*OF solution tf phi*] **have** *CF*: *ACC-cf-mf* φ = {} **.**
from *theorem-13*[*OF tf phi POS CF*]
show *?thesis* **by** *auto*
qed

Next we consider general monotone formulas.

theorem *Clique-not-solvable-by-poly-mono*: **assumes** *solution*: ∀ *G* ∈ $\mathcal{G}$. *G* ∈ *CLIQUE* ⟷ *eval* (ϑ_g *G*) φ
and *phi*: $\varphi \in \mathcal{A}$
shows *cs* $\varphi >$ *k powr* (*4* / *7* ∗ *sqrt k*)
proof −
note *vars* = *phi*[*unfolded* $\mathcal{A}$*-def*]
have *CL*: *CLIQUE* = *Clique* [*k^4*] *k* $\mathcal{G}$ = *Graphs* [*k^4*]
unfolding *CLIQUE-def* $\mathcal{K}$*-altdef m-def Clique-def* **by** *auto*
with *empty-CLIQUE* **have** {} ∉ *Clique* [*k^4*] *k* **by** *simp*
with *solution*[*rule-format*, *of* {}]
have ¬ *eval* (ϑ_g {}) φ **by** (*auto simp*: *Graphs-def*)
from *to-tf-mformula*[*OF this*]
obtain ψ **where** ∗: $\psi \in$ *tf-mformula*
(∀ ϑ. *eval* ϑ φ = *eval* ϑ ψ) *vars* ψ ⊆ *vars* φ *cs* $\psi \leq$ *cs* φ **by** *auto*
with *phi solution* **have** *psi*: $\psi \in \mathcal{A}$
and *solution*: ∀ *G*∈$\mathcal{G}$. (*G* ∈ *CLIQUE*) = *eval* (ϑ_g *G*) ψ **unfolding** $\mathcal{A}$*-def* **by** *auto*
from *Clique-not-solvable-by-small-tf-mformula*[*OF solution* ∗(*1*) *psi*]
show *?thesis* **using** ∗(*4*) **by** *auto*
qed

We next expand all abbreviations and definitions of the locale, but stay within the locale

theorem *Clique-not-solvable-by-small-monotone-circuit-in-locale*: **assumes** *phi-solves-clique*:

∀ *G* ∈ *Graphs* [*k^4*]. *G* ∈ *Clique* [*k^4*] *k* ⟷ *eval* (λ *x*. π *x* ∈ *G*) φ
and *vars*: *vars* $\varphi \subseteq \mathcal{V}$
shows *cs* $\varphi >$ *k powr* (*4* / *7* ∗ *sqrt k*)
proof −
{
fix *G*
assume *G*: *G* ∈ $\mathcal{G}$
have *eval* (λ *x*. π *x* ∈ *G*) φ = *eval* (ϑ_g *G*) φ **using** *vars*
by (*intro eval-vars*, *auto simp*: ϑ_g*-def*)
}
have *CL*: *CLIQUE* = *Clique* [*k^4*] *k* $\mathcal{G}$ = *Graphs* [*k^4*]

unfolding *CLIQUE-def* $\mathcal{K}$*-altdef m-def Clique-def* **by** *auto*
{
fix G
assume G: $G \in \mathcal{G}$
have *eval* (λ x. π $x \in G$) φ = *eval* (ϑ_g G) φ **using** *vars*
by (*intro eval-vars*, *auto simp*: ϑ_g*-def*)
}
with *phi-solves-clique CL* **have** *solves*: $\forall$ $G \in \mathcal{G}$. $G \in$ *CLIQUE* $\longleftrightarrow$ *eval* (ϑ_g G) φ
by *auto*
from *vars* **have** *inA*: $\varphi \in \mathcal{A}$ **by** (*auto simp*: $\mathcal{A}$*-def*)
from *Clique-not-solvable-by-poly-mono*[*OF solves inA*]
show *?thesis* **by** *auto*
qed
end

Let us now move the theorem outside the locale

definition *Large-Number* **where** *Large-Number* = *Max* {*64*, *L0''^2*, *L0^2*, *L0'^2*, *M0*, *M0'*}

theorem *Clique-not-solvable-by-small-monotone-circuit-squared*:
fixes φ :: *'a mformula*
assumes k: $\exists$ l. $k = l$^*2*
and *LARGE*: $k \geq$ *Large-Number*
and π: *bij-betw* π V [k^*4*]**^2**
and *solution*: $\forall$ $G \in$*Graphs* [k ^ *4*]. ($G \in$ *Clique* [k ^ *4*] k) = *eval* (λ x. π $x \in G$) φ
and *vars*: *vars* $\varphi \subseteq V$
shows *cs* φ > k *powr* (*4* / *7* $*$ *sqrt* k)
proof −
from k **obtain** l **where** *kk*: $k = l$^*2* **by** *auto*
note *LARGE* = *LARGE*[*unfolded Large-Number-def*]
have *k8*: $k \geq$ *8^2* **using** *LARGE* **by** *auto*
from *this*[*unfolded kk power2-nat-le-eq-le*]
have *l8*: $l \geq$ *8* **.**
define p **where** p = *nat* (*ceiling* (l $*$ *log 2* (k^*4*)))
have *tedious*: l $*$ *log 2* (k ^ *4*) $\geq$ *0* **using** *l8 k8* **by** *auto*
have *int* p = *ceiling* (l $*$ *log 2* (k ^ *4*)) **unfolding** *p-def*
by (*rule nat-0-le*, *insert tedious*, *auto*)
from *arg-cong*[*OF this*, *of real-of-int*]
have *rp*: *real* p = *ceiling* (l $*$ *log 2* (k ^ *4*)) **by** *simp*
have *one*: *real* l $*$ *log 2* (k ^ *4*) $\leq p$ **unfolding** *rp* **by** *simp*
have *two*: $p \leq$ *real* l $*$ *log 2* (k ^ *4*) + *1* **unfolding** *rp* **by** *simp*
have *real* l < *real* l + *1* **by** *simp*
also have ... $\leq$ *real* l + *real* l **using** *l8* **by** *simp*
also have ... = *real* l $*$ *2* **by** *simp*
also have ... = *real* l $*$ *log 2* (*2^2*)
by (*subst log-pow-cancel*, *auto*)
also have ... $\leq$ *real* l $*$ *log 2* (k ^ *4*)

```
  proof (intro mult-left-mono, subst log-le-cancel-iff)
    have (4 :: real) ≤ 2^4 by simp
    also have ... ≤ real k^4
      by (rule power-mono, insert k8, auto)
    finally show 2² ≤ real (k ^ 4) by simp
  qed (insert k8, auto)
  also have ... ≤ p by fact
  finally have lp: l < p by auto
  interpret second-assumptions l p k
  proof (unfold-locales)
    show 2 < l using l8 by auto
    show 8 ≤ l by fact
    show k = l^2 by fact
    show l < p by fact
    from LARGE have L0''^2 ≤ k by auto
    from this[unfolded kk power2-nat-le-eq-le]
    have L0''l: L0'' ≤ l .
    have p ≤ real l * log 2 (k ^ 4) + 1 by fact
    also have ... < k unfolding kk
      by (intro L0'' L0''l)
    finally show p < k by simp
  qed
  interpret third-assumptions l p k
  proof
    show real l * log 2 (real m) ≤ p using one unfolding m-def .
    show p ≤ real l * log 2 (real m) + 1 using two unfolding m-def .
    from LARGE have L0^2 ≤ k by auto
    from this[unfolded kk power2-nat-le-eq-le]
    show L0 ≤ l .
    from LARGE have L0'^2 ≤ k by auto
    from this[unfolded kk power2-nat-le-eq-le]
    show L0' ≤ l .
    show M0' ≤ m using km LARGE by simp
    show M0 ≤ m using km LARGE by simp
  qed
  interpret forth-assumptions l p k V π
    by (standard, insert π m-def, auto simp: bij-betw-same-card[OF π])
  from Clique-not-solvable-by-small-monotone-circuit-in-locale[OF solution vars]
  show ?thesis .
qed
```

A variant where we get rid of the $k = l^2$-assumption by just taking squares everywhere.

```
theorem Clique-not-solvable-by-small-monotone-circuit:
  fixes φ :: 'a mformula
  assumes LARGE: k ≥ Large-Number
  and π: bij-betw π V [k^8]^2
  and solution: ∀ G∈Graphs [k ^ 8]. (G ∈ Clique [k ^ 8] (k^2)) = eval (λ x. π x
∈ G) φ
```

and *vars*: *vars* $\varphi \subseteq V$
shows *cs* $\varphi >$ *k powr* (*8* / *7* $*$ *k*)
proof −
from *LARGE* **have** *LARGE*: *Large-Number* $\leq k^2$
by (*simp add*: *power2-nat-le-imp-le*)
have *id*: k^2 ^ *4* = *k*^*8* *sqrt* (*k*^*2*) = *k* **by** *auto*
from *Clique-not-solvable-by-small-monotone-circuit-squared*[*of k*^*2*, *unfolded id*, *OF* - *LARGE* π *solution vars*]
have *cs* $\varphi >$ (*k*^*2*) *powr* (*4* / *7* $*$ *k*) **by** *auto*
also have (*k*^*2*) *powr* (*4* / *7* $*$ *k*) = *k powr* (*8* / *7* $*$ *k*)
unfolding *of-nat-power* **using** *powr-powr*[*of real k 2*] **by** *simp*
finally show *?thesis* **.**
qed

definition *large-number* **where** *large-number* = *Large-Number*^*8*

Finally a variant, where the size is formulated depending on n, the number of vertices.

theorem *Clique-with-n-nodes-not-solvable-by-small-monotone-circuit*:
fixes φ :: *'a mformula*
assumes *large*: $n \geq$ *large-number*
and *kn*: $\exists$ *k*. *n* = *k*^*8*
and π: *bij-betw* π *V* [*n*]^**2**
and *s*: *s* = *root 4 n*
and *solution*: $\forall$ *G*$\in$*Graphs* [*n*]. (*G* $\in$ *Clique* [*n*] *s*) = *eval* (λ *x*. π *x* $\in$ *G*) φ
and *vars*: *vars* $\varphi \subseteq V$
shows *cs* $\varphi >$ (*root 7 n*) *powr* (*root 8 n*)
proof −
from *kn* **obtain** *k* **where** *nk*: *n* = *k*^*8* **by** *auto*
have *kn*: *k* = *root 8 n* **unfolding** *nk of-nat-power*
by (*subst real-root-pos2*, *auto*)
have *root 4 n* = *root 4* ((*real* (*k*^*2*))^*4*) **unfolding** *nk* **by** *simp*
also have ... = *k*^*2* **by** (*simp add*: *real-root-pos-unique*)
finally have *r4*: *root 4 n* = *k*^*2* **by** *simp*
have *s*: *s* = *k*^*2* **using** *s* **unfolding** *r4* **by** *simp*
from *large*[*unfolded nk large-number-def*] **have** *Large*: *k* $\geq$ *Large-Number* **by** *simp*
have *0* < *Large-Number* **unfolding** *Large-Number-def* **by** *simp*
with *Large* **have** *k0*: *k* > *0* **by** *auto*
hence *n0*: *n* > *0* **using** *nk* **by** *simp*
from *Clique-not-solvable-by-small-monotone-circuit*[*OF Large* π[*unfolded nk*] - *vars*]
solution[*unfolded s*] *nk*
have *real k powr* (*8* / *7* $*$ *real k*) < *cs* φ **by** *auto*
also have *real k powr* (*8* / *7* $*$ *real k*) = *root 8 n powr* (*8* / *7* $*$ *root 8 n*)
unfolding *kn* **by** *simp*
also have ... = ((*root 8 n*) *powr* (*8* / *7*)) *powr* (*root 8 n*)
unfolding *powr-powr* **by** *simp*
also have (*root 8 n*) *powr* (*8* / *7*) = *root 7 n* **using** *n0*

by (*simp add*: *root-powr-inverse powr-powr*)
finally show *?thesis* **.**
qed

end

References

[1] N. Alon and R. B. Boppana. The monotone circuit complexity of Boolean functions. *Combinatorica*, 7(1):1–22, 1987.

[2] R. B. Boppana and M. Sipser. The complexity of finite functions. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity*, pages 757–804. Elsevier and MIT Press, 1990.

[3] P. Erds and R. Rado. Intersection theorems for systems of sets. *Journal of the London Mathematical Society*, 35:85–90, 1960.

[4] L. Gordeev. On P versus NP. Avaible at http://arxiv.org/abs/2005.00809v3.

[5] C. H. Papadimitriou. *Computational complexity*. Addison-Wesley, 1994.