The Földes–Hammer Characterization of Split Graphs

Arthur Freitas Ramos David Barros Hulak
Ruy J. G. B. de Queiroz

September 11, 2026

Abstract

This entry formalizes the classical Földes–Hammer theorem characterizing finite split graphs as exactly the graphs with no induced copy of $2K_2$, C_4, or C_5. It builds on the AFP session `Undirected_Graph_Theory`.

1 Overview

A finite graph is split when its vertex set can be partitioned into a clique and an independent set. The main theorem of this entry is `foldes_hammer` in `Foldes_Hammer.thy`, which proves the equivalence between split graphs and the absence of induced $2K_2$, C_4, and C_5.

The forward direction is established by analyzing how any split partition excludes each forbidden configuration. The converse direction follows an extremal argument: choose a maximum clique whose complement minimizes the number of induced edges, orient any remaining outside edge according to nested nonneighbor sets in the clique, and swap one clique vertex with an outside vertex to derive a contradiction.

2 Sources

The entry follows the classical theorem due to Földes and Hammer [2] and standard graph-class background as presented by Brandstädt, Le, and Spinrad [1].

Contents

2 Sources 1

theory *Foldes-Hammer*
imports *Undirected-Graph-Theory.Undirected-Graphs-Root*
begin

context *fin-sgraph*
begin

definition *is-clique* :: *'a set* ⇒ *bool* **where**
is-clique C ⟷ *C* ⊆ *V* ∧ *induced-edges C* = *all-edges C*

definition *cliques* :: *'a set set* **where**
cliques = {*C*. *is-clique C*}

definition *maximum-clique* :: *'a set* ⇒ *bool* **where**
maximum-clique K ⟷ *K* ∈ *cliques* ∧ (∀ *C*∈*cliques*. *card C* ≤ *card K*)

definition *outside-edge-count* :: *'a set* ⇒ *nat* **where**
outside-edge-count K = *card* (*induced-edges* (*V* − *K*))

definition *is-split-partition* :: *'a set* ⇒ *'a set* ⇒ *bool* **where**
is-split-partition C I ⟷ *C* ∩ *I* = {} ∧ *C* ∪ *I* = *V* ∧ *is-clique C* ∧ *is-independent-set I*

definition *is-split-graph* :: *bool* **where**
is-split-graph ⟷ (∃ *C I*. *is-split-partition C I*)

definition *induces-2K2* :: *'a* ⇒ *'a* ⇒ *'a* ⇒ *'a* ⇒ *bool* **where**
induces-2K2 a b c d ⟷
distinct [*a*, *b*, *c*, *d*] ∧
vert-adj a b ∧ *vert-adj c d* ∧
¬ *vert-adj a c* ∧ ¬ *vert-adj a d* ∧ ¬ *vert-adj b c* ∧ ¬ *vert-adj b d*

definition *induces-C4* :: *'a* ⇒ *'a* ⇒ *'a* ⇒ *'a* ⇒ *bool* **where**
induces-C4 a b c d ⟷
distinct [*a*, *b*, *c*, *d*] ∧
vert-adj a b ∧ *vert-adj b c* ∧ *vert-adj c d* ∧ *vert-adj d a* ∧
¬ *vert-adj a c* ∧ ¬ *vert-adj b d*

definition *induces-C5* :: *'a* ⇒ *'a* ⇒ *'a* ⇒ *'a* ⇒ *'a* ⇒ *bool* **where**
induces-C5 a b c d e ⟷
distinct [*a*, *b*, *c*, *d*, *e*] ∧
vert-adj a b ∧ *vert-adj b c* ∧ *vert-adj c d* ∧ *vert-adj d e* ∧ *vert-adj e a* ∧
¬ *vert-adj a c* ∧ ¬ *vert-adj a d* ∧ ¬ *vert-adj b d* ∧ ¬ *vert-adj b e* ∧ ¬ *vert-adj c e*

definition *has-induced-2K2* :: *bool* **where**
has-induced-2K2 ⟷ (∃ *a b c d*. *induces-2K2 a b c d*)

definition *has-induced-C4* :: *bool* **where**
has-induced-C4 $\longleftrightarrow$ ($\exists$ *a b c d*. *induces-C4 a b c d*)

definition *has-induced-C5* :: *bool* **where**
has-induced-C5 $\longleftrightarrow$ ($\exists$ *a b c d e*. *induces-C5 a b c d e*)

definition *foldes-hammer-free* :: *bool* **where**
foldes-hammer-free $\longleftrightarrow$ $\neg$ *has-induced-2K2* $\wedge$ $\neg$ *has-induced-C4* $\wedge$ $\neg$ *has-induced-C5*

lemma *is-clique-alt*:
is-clique C $\longleftrightarrow$ *C* $\subseteq$ *V* $\wedge$ ($\forall$ *u*$\in$*C*. $\forall$ *v*$\in$*C*. *u* $\neq$ *v* $\longrightarrow$ *vert-adj u v*)
proof
assume *H*: *is-clique C*
then have *C* $\subseteq$ *V* **by** (*simp add*: *is-clique-def*)
moreover have $\forall$ *u*$\in$*C*. $\forall$ *v*$\in$*C*. *u* $\neq$ *v* $\longrightarrow$ *vert-adj u v*
proof (*intro ballI impI*)
fix *u v*
assume *uC*: *u* $\in$ *C* **and** *vC*: *v* $\in$ *C* **and** *uv*: *u* $\neq$ *v*
from *H* **have** *eq*: *induced-edges C* = *all-edges C*
by (*simp add*: *is-clique-def*)
have {*u*, *v*} $\in$ *all-edges C*
using *uC vC uv* **by** (*auto simp*: *all-edges-alt*)
with *eq* **have** {*u*, *v*} $\in$ *induced-edges C* **by** *simp*
then show *vert-adj u v*
by (*auto simp*: *induced-edges-def vert-adj-def*)
qed
ultimately show *C* $\subseteq$ *V* $\wedge$ ($\forall$ *u*$\in$*C*. $\forall$ *v*$\in$*C*. *u* $\neq$ *v* $\longrightarrow$ *vert-adj u v*)
by *simp*
next
assume *H*: *C* $\subseteq$ *V* $\wedge$ ($\forall$ *u*$\in$*C*. $\forall$ *v*$\in$*C*. *u* $\neq$ *v* $\longrightarrow$ *vert-adj u v*)
then have *Csub*: *C* $\subseteq$ *V*
and *adj*: $\forall$ *u*$\in$*C*. $\forall$ *v*$\in$*C*. *u* $\neq$ *v* $\longrightarrow$ *vert-adj u v*
by *auto*
have *induced-edges C* = *all-edges C*
proof (*rule subset-antisym*)
show *induced-edges C* $\subseteq$ *all-edges C*
using *induced-edges-alt* **by** *auto*
next
show *all-edges C* $\subseteq$ *induced-edges C*
proof
fix *e*
assume *eC*: *e* $\in$ *all-edges C*
then obtain *u v* **where** *e*: *e* = {*u*, *v*} **and** *uC*: *u* $\in$ *C* **and** *vC*: *v* $\in$ *C* **and** *uv*: *u* $\neq$ *v*
by (*auto simp*: *all-edges-alt*)
from *adj uC vC uv* **have** *vert-adj u v* **by** *blast*
then have *e* $\in$ *E*
by (*simp add*: *e vert-adj-def*)

```
      moreover from eC have e ⊆ C
        by (auto simp: all-edges-def)
      ultimately show e ∈ induced-edges C
        by (simp add: induced-edges-def)
    qed
  qed
  with Csub show is-clique C
    by (simp add: is-clique-def)
qed

lemma clique-pair-adj:
  assumes is-clique C u ∈ C v ∈ C u ≠ v
  shows vert-adj u v
  using assms by (auto simp: is-clique-alt)

lemma clique-mono:
  assumes is-clique C D ⊆ C
  shows is-clique D
  using assms by (auto simp: is-clique-alt)

lemma vert-adj-neq:
  assumes vert-adj u v
  shows u ≠ v
proof
  assume uv-eq: u = v
  from assms have card2: card {u, v} = 2
    unfolding vert-adj-def using two-edges by blast
  from uv-eq have card {u, v} = 1
    by simp
  with card2 show False
    by simp
qed

lemma empty-clique: is-clique {}
  by (simp add: is-clique-alt)

lemma singleton-clique:
  assumes v ∈ V
  shows is-clique {v}
  using assms by (auto simp: is-clique-alt)

lemma finite-cliques: finite cliques
proof −
  have cliques ⊆ Pow V
    by (auto simp: cliques-def is-clique-def)
  moreover have finite (Pow V)
    using finV by simp
  ultimately show ?thesis
    by (rule finite-subset)
```

qed

lemma *cliques-nonempty*: $\{\} \in$ *cliques*
by (*simp add*: *cliques-def empty-clique*)

lemma *maximum-clique-exists*:
obtains *K* **where** *maximum-clique K*
proof −
have *fin*: *finite* (*card* ‘ *cliques*)
using *finite-cliques* **by** *simp*
have *ne*: *card* ‘ *cliques* $\neq \{\}$
using *cliques-nonempty* **by** *auto*
let *?m* = *Max* (*card* ‘ *cliques*)
have *m-in*: *?m* $\in$ *card* ‘ *cliques*
using *Max-in*[*OF fin ne*] **.**
then obtain *K* **where** *K-in*: *K* $\in$ *cliques* **and** *K-card*: *card K* = *?m*
by *auto*
have *maxK*: $\forall$ *C*$\in$*cliques*. *card C* $\leq$ *card K*
proof
fix *C*
assume *C-in*: *C* $\in$ *cliques*
then have *card C* $\in$ *card* ‘ *cliques*
by *auto*
then have *card C* $\leq$ *?m*
using *Max-ge*[*OF fin*] **by** *blast*
then show *card C* $\leq$ *card K*
by (*simp add*: *K-card*)
qed
have *maximum-clique K*
using *K-in maxK* **by** (*simp add*: *maximum-clique-def*)
then show *?thesis*
using *that* **by** *blast*
qed

lemma *maximum-clique-finite*:
assumes *maximum-clique K*
shows *finite K*
using *assms finV finite-subset*
by (*auto simp*: *maximum-clique-def cliques-def is-clique-def*)

lemma *finite-induced-edges-subset-V*:
assumes *S* $\subseteq$ *V*
shows *finite* (*induced-edges S*)
proof −
have *finS*: *finite S*
using *assms finV finite-subset* **by** *blast*
have *induced-edges S* $\subseteq$ *Pow S*
by (*auto simp*: *induced-edges-def*)
moreover have *finite* (*Pow S*)

```
    using finS by simp
  ultimately show ?thesis
    by (rule finite-subset)
qed

lemma obtain-best-clique:
  obtains K where maximum-clique K
    and ∀ L. maximum-clique L ⟶ outside-edge-count K ≤ outside-edge-count L
proof −
  let ?M = {K. maximum-clique K}
  have finM: finite ?M
  proof (rule finite-subset[of ?M cliques])
    show ?M ⊆ cliques
      by (auto simp: maximum-clique-def)
    show finite cliques
      using finite-cliques .
  qed
  have neM: ?M ≠ {}
    using maximum-clique-exists by blast
  have finImg: finite (outside-edge-count ‘ ?M)
    using finM by simp
  have neImg: outside-edge-count ‘ ?M ≠ {}
    using neM by auto
  let ?m = Min (outside-edge-count ‘ ?M)
  have mIn: ?m ∈ outside-edge-count ‘ ?M
    using Min-in[OF finImg neImg] .
  then obtain K where Kmax: maximum-clique K and Kmin: outside-edge-count
K = ?m
    by auto
    have Kbest: ∀ L. maximum-clique L ⟶ outside-edge-count K ≤ out-
side-edge-count L
    using Kmin finImg neImg
    by (simp add: Min-le)
  show ?thesis
    using that Kmax Kbest by blast
qed

lemma insert-clique-if-all-adj:
  assumes C: is-clique C
    and vV: v ∈ V
    and vnot: v ∉ C
    and adj: ∀ u∈C. vert-adj u v
  shows is-clique (insert v C)
proof −
  from C have Csub: C ⊆ V
    and Cadj: ∀ u∈C. ∀ w∈C. u ≠ w ⟶ vert-adj u w
    by (auto simp: is-clique-alt)
  have insert v C ⊆ V
    using Csub vV by auto
```

```
  moreover have ∀ u∈insert v C. ∀ w∈insert v C. u ≠ w ⟶ vert-adj u w
  proof (intro ballI impI)
    fix u w
    assume uin: u ∈ insert v C and win: w ∈ insert v C and uw: u ≠ w
    show vert-adj u w
    proof (cases u = v)
      case u-eq-v: True
      with win vnot uw have w ∈ C by auto
      then have vert-adj w v
        using adj by simp
      then show ?thesis
        using u-eq-v by (simp add: vert-adj-sym)
    next
      case u-ne-v: False
      show ?thesis
      proof (cases w = v)
        case w-eq-v: True
        with uin vnot uw u-ne-v have u ∈ C by auto
        then have vert-adj u v
          using adj by simp
        then show ?thesis
          using w-eq-v by simp
      next
        case w-ne-v: False
        from uin win u-ne-v w-ne-v have u ∈ C w ∈ C by auto
        with Cadj uw show ?thesis by blast
      qed
    qed
  qed
  ultimately show ?thesis
    by (simp add: is-clique-alt)
qed

lemma maximum-clique-outside-has-nonneighbor:
  assumes maxK: maximum-clique K
    and vV: v ∈ V
    and vnot: v ∉ K
  shows ∃ u∈K. ¬ vert-adj u v
proof (rule ccontr)
  assume neg: ¬ (∃ u∈K. ¬ vert-adj u v)
  then have all-adj: ∀ u∈K. vert-adj u v
    by auto
  from maxK have K-clique: is-clique K
    by (auto simp: maximum-clique-def cliques-def)
  have ins-clique: is-clique (insert v K)
    using insert-clique-if-all-adj[OF K-clique vV vnot all-adj] .
  then have ins-in: insert v K ∈ cliques
    by (simp add: cliques-def)
  from maxK have bound: ∀ C∈cliques. card C ≤ card K
```

```
    by (auto simp: maximum-clique-def)
  from maxK have Ksub: K ⊆ V
    by (auto simp: maximum-clique-def cliques-def is-clique-def)
  then have finK: finite K
    using finV finite-subset by blast
  from ins-in bound have card (insert v K) ≤ card K
    by blast
  with finK vnot show False
    by simp
qed

lemma split-graph-alt:
  is-split-graph ⟷ (∃ C. is-clique C ∧ is-independent-set (V − C))
proof
  assume H: is-split-graph
  then obtain C I where sp: is-split-partition C I
    by (auto simp: is-split-graph-def)
  then have I = V − C
    by (auto simp: is-split-partition-def is-clique-def)
  with sp show ∃ C. is-clique C ∧ is-independent-set (V − C)
    by (auto simp: is-split-partition-def)
next
  assume H: ∃ C. is-clique C ∧ is-independent-set (V − C)
  then obtain C where C: is-clique C is-independent-set (V − C)
    by blast
  have is-split-partition C (V − C)
    using C by (auto simp: is-split-partition-def is-clique-def)
  then show is-split-graph
    by (auto simp: is-split-graph-def)
qed

lemma independent-pair-not-adj:
  assumes is-independent-set I u ∈ I v ∈ I
  shows ¬ vert-adj u v
  using assms by (auto simp: is-independent-alt)

lemma split-partition-edge-hits-clique:
  assumes sp: is-split-partition C I
    and uv: vert-adj u v
  shows u ∈ C ∨ v ∈ C
proof (rule ccontr)
  assume notC: ¬ (u ∈ C ∨ v ∈ C)
  from uv have uV: u ∈ V and vV: v ∈ V
    using vert-adj-imp-inV by auto
  from sp uV vV notC have uI: u ∈ I and vI: v ∈ I
    by (auto simp: is-split-partition-def)
  from sp have indep: is-independent-set I
    by (simp add: is-split-partition-def)
  from independent-pair-not-adj[OF indep uI vI] uv show False
```

by *contradiction*
qed

lemma *split-partition-nonedge-not-both-clique*:
assumes *sp*: *is-split-partition C I*
and *uV*: $u \in V$
and *vV*: $v \in V$
and *uv*: $u \neq v$
and *not-adj*: $\neg$ *vert-adj u v*
shows $\neg (u \in C \wedge v \in C)$
proof
assume *uvC*: $u \in C \wedge v \in C$
then have *uC*: $u \in C$ **and** *vC*: $v \in C$
by *auto*
from *sp* **have** *clique*: *is-clique C*
by (*simp add*: *is-split-partition-def*)
from *clique-pair-adj*[*OF clique uC vC uv*] *not-adj* **show** *False*
by *contradiction*
qed

lemma *induces-2K2-inV*:
assumes *induces-2K2 a b c d*
shows $a \in V$ $b \in V$ $c \in V$ $d \in V$
using *assms vert-adj-imp-inV* **by** (*auto simp*: *induces-2K2-def*)

lemma *induces-C4-inV*:
assumes *induces-C4 a b c d*
shows $a \in V$ $b \in V$ $c \in V$ $d \in V$
using *assms vert-adj-imp-inV* **by** (*auto simp*: *induces-C4-def*)

lemma *induces-C5-inV*:
assumes *induces-C5 a b c d e*
shows $a \in V$ $b \in V$ $c \in V$ $d \in V$ $e \in V$
using *assms vert-adj-imp-inV* **by** (*auto simp*: *induces-C5-def*)

lemma *split-partition-forbids-2K2*:
assumes *sp*: *is-split-partition C I*
and *H*: *induces-2K2 a b c d*
shows *False*
proof −
from *H* **have** *ab*: *vert-adj a b* **and** *cd*: *vert-adj c d*
and *aV*: $a \in V$ **and** *bV*: $b \in V$ **and** *cV*: $c \in V$ **and** *dV*: $d \in V$
and *ac*: $a \neq c$ **and** *ad*: $a \neq d$ **and** *bc*: $b \neq c$ **and** *bd*: $b \neq d$
and *nac*: $\neg$ *vert-adj a c* **and** *nad*: $\neg$ *vert-adj a d* **and** *nbc*: $\neg$ *vert-adj b c* **and** *nbd*: $\neg$ *vert-adj b d*
using *induces-2K2-inV*[*OF H*] **by** (*auto simp*: *induces-2K2-def*)
have $a \in C \vee b \in C$
using *split-partition-edge-hits-clique*[*OF sp ab*] **.**
moreover have $c \in C \vee d \in C$

using *split-partition-edge-hits-clique*[*OF sp cd*] .
moreover have ¬ (*a* ∈ *C* ∧ *c* ∈ *C*)
using *split-partition-nonedge-not-both-clique*[*OF sp aV cV ac nac*] .
moreover have ¬ (*a* ∈ *C* ∧ *d* ∈ *C*)
using *split-partition-nonedge-not-both-clique*[*OF sp aV dV ad nad*] .
moreover have ¬ (*b* ∈ *C* ∧ *c* ∈ *C*)
using *split-partition-nonedge-not-both-clique*[*OF sp bV cV bc nbc*] .
moreover have ¬ (*b* ∈ *C* ∧ *d* ∈ *C*)
using *split-partition-nonedge-not-both-clique*[*OF sp bV dV bd nbd*] .
ultimately show *False*
by *blast*
qed

lemma *split-partition-forbids-C4*:
assumes *sp*: *is-split-partition C I*
and *H*: *induces-C4 a b c d*
shows *False*
proof −
from *H* **have** *ab*: *vert-adj a b* **and** *bc*: *vert-adj b c* **and** *cd*: *vert-adj c d* **and** *da*: *vert-adj d a*
and *aV*: *a* ∈ *V* **and** *bV*: *b* ∈ *V* **and** *cV*: *c* ∈ *V* **and** *dV*: *d* ∈ *V*
and *ac*: *a* ≠ *c* **and** *bd*: *b* ≠ *d*
and *nac*: ¬ *vert-adj a c* **and** *nbd*: ¬ *vert-adj b d*
using *induces-C4-inV*[*OF H*] **by** (*auto simp*: *induces-C4-def*)
have *a* ∈ *C* ∨ *b* ∈ *C*
using *split-partition-edge-hits-clique*[*OF sp ab*] .
moreover have *b* ∈ *C* ∨ *c* ∈ *C*
using *split-partition-edge-hits-clique*[*OF sp bc*] .
moreover have *c* ∈ *C* ∨ *d* ∈ *C*
using *split-partition-edge-hits-clique*[*OF sp cd*] .
moreover have *d* ∈ *C* ∨ *a* ∈ *C*
using *split-partition-edge-hits-clique*[*OF sp da*] .
moreover have ¬ (*a* ∈ *C* ∧ *c* ∈ *C*)
using *split-partition-nonedge-not-both-clique*[*OF sp aV cV ac nac*] .
moreover have ¬ (*b* ∈ *C* ∧ *d* ∈ *C*)
using *split-partition-nonedge-not-both-clique*[*OF sp bV dV bd nbd*] .
ultimately show *False*
by *blast*
qed

lemma *split-partition-forbids-C5*:
assumes *sp*: *is-split-partition C I*
and *H*: *induces-C5 a b c d e*
shows *False*
proof −
from *H* **have** *ab*: *vert-adj a b* **and** *bc*: *vert-adj b c* **and** *cd*: *vert-adj c d* **and** *de*: *vert-adj d e* **and** *ea*: *vert-adj e a*
and *aV*: *a* ∈ *V* **and** *bV*: *b* ∈ *V* **and** *cV*: *c* ∈ *V* **and** *dV*: *d* ∈ *V* **and** *eV*: *e* ∈ *V*

```
    and ac: a ≠ c and ad: a ≠ d and bd: b ≠ d and be: b ≠ e and ce: c ≠ e
    and nac: ¬ vert-adj a c and nad: ¬ vert-adj a d and nbd: ¬ vert-adj b d and
nbe: ¬ vert-adj b e and nce: ¬ vert-adj c e
    using induces-C5-inV[OF H] by (auto simp: induces-C5-def)
  have a ∈ C ∨ b ∈ C
    using split-partition-edge-hits-clique[OF sp ab] .
  moreover have b ∈ C ∨ c ∈ C
    using split-partition-edge-hits-clique[OF sp bc] .
  moreover have c ∈ C ∨ d ∈ C
    using split-partition-edge-hits-clique[OF sp cd] .
  moreover have d ∈ C ∨ e ∈ C
    using split-partition-edge-hits-clique[OF sp de] .
  moreover have e ∈ C ∨ a ∈ C
    using split-partition-edge-hits-clique[OF sp ea] .
  moreover have ¬ (a ∈ C ∧ c ∈ C)
    using split-partition-nonedge-not-both-clique[OF sp aV cV ac nac] .
  moreover have ¬ (a ∈ C ∧ d ∈ C)
    using split-partition-nonedge-not-both-clique[OF sp aV dV ad nad] .
  moreover have ¬ (b ∈ C ∧ d ∈ C)
    using split-partition-nonedge-not-both-clique[OF sp bV dV bd nbd] .
  moreover have ¬ (b ∈ C ∧ e ∈ C)
    using split-partition-nonedge-not-both-clique[OF sp bV eV be nbe] .
  moreover have ¬ (c ∈ C ∧ e ∈ C)
    using split-partition-nonedge-not-both-clique[OF sp cV eV ce nce] .
  ultimately show False
    by blast
qed

lemma split-graph-imp-foldes-hammer-free:
  assumes is-split-graph
  shows foldes-hammer-free
proof −
  obtain C I where sp: is-split-partition C I
    using assms by (auto simp: is-split-graph-def)
  have ¬ has-induced-2K2
    using sp split-partition-forbids-2K2 by (auto simp: has-induced-2K2-def)
  moreover have ¬ has-induced-C4
    using sp split-partition-forbids-C4 by (auto simp: has-induced-C4-def)
  moreover have ¬ has-induced-C5
    using sp split-partition-forbids-C5 by (auto simp: has-induced-C5-def)
  ultimately show ?thesis
    by (simp add: foldes-hammer-free-def)
qed

lemma outside-adjacent-nonneighbor-sets-nested:
  assumes fh: foldes-hammer-free
    and Kmax: maximum-clique K
    and uO: u ∈ V − K
    and vO: v ∈ V − K
```

```
    and uv: vert-adj u v
  shows {x∈K. ¬ vert-adj x u} ⊆ {x∈K. ¬ vert-adj x v} ∨
        {x∈K. ¬ vert-adj x v} ⊆ {x∈K. ¬ vert-adj x u}
proof (rule ccontr)
  let ?A = {x∈K. ¬ vert-adj x u}
  let ?B = {x∈K. ¬ vert-adj x v}
  assume not-nested: ¬ (?A ⊆ ?B ∨ ?B ⊆ ?A)
  then obtain a b where aA: a ∈ ?A and aB: a ∉ ?B and bB: b ∈ ?B and bA:
b ∉ ?A
    by blast
  have aK: a ∈ K and aNu: ¬ vert-adj a u
    using aA by auto
  have bK: b ∈ K and bNv: ¬ vert-adj b v
    using bB by auto
  have av: vert-adj a v
    using aB aK by auto
  have bu: vert-adj b u
    using bA bK by auto
  have ab-ne: a ≠ b
    using aA bA by auto
  have uv-ne: u ≠ v
    using vert-adj-neq[OF uv] .
  have Kclq: is-clique K
    using Kmax by (auto simp: maximum-clique-def cliques-def)
  have ab: vert-adj a b
    using clique-pair-adj[OF Kclq aK bK ab-ne] .
  have vu: vert-adj v u
    using uv by (simp add: vert-adj-sym)
  have ub: vert-adj u b
    using bu by (simp add: vert-adj-sym)
  have ba: vert-adj b a
    using ab by (simp add: vert-adj-sym)
  have vNb: ¬ vert-adj v b
    using bNv by (simp add: vert-adj-sym)
  have dist: distinct [a, v, u, b]
    using aK bK uO vO ab-ne uv-ne by auto
  have induces-C4 a v u b
    using dist av vu ub ba aNu vNb
    by (simp add: induces-C4-def)
  then have has-induced-C4
    by (auto simp: has-induced-C4-def)
  with fh show False
    by (simp add: foldes-hammer-free-def)
qed

lemma outside-adjacent-smaller-nonneighbor-set-singleton:
  assumes fh: foldes-hammer-free
    and Kmax: maximum-clique K
    and uO: u ∈ V − K
```

```
    and vO: v ∈ V − K
    and uv: vert-adj u v
    and AB: {x∈K. ¬ vert-adj x u} ⊆ {x∈K. ¬ vert-adj x v}
  obtains a where {x∈K. ¬ vert-adj x u} = {a} and a ∈ K and ¬ vert-adj a u
and ¬ vert-adj a v
proof −
  have uV: u ∈ V and unotK: u ∉ K
    using uO by auto
  obtain a where aK: a ∈ K and aNu: ¬ vert-adj a u
    using maximum-clique-outside-has-nonneighbor[OF Kmax uV unotK] by blast
  have aA: a ∈ {x∈K. ¬ vert-adj x u}
    using aK aNu by auto
  have aB: a ∈ {x∈K. ¬ vert-adj x v}
    using AB aA by blast
  have aNv: ¬ vert-adj a v
    using aB by auto
  have Aeq: {x∈K. ¬ vert-adj x u} = {a}
  proof (rule subset-antisym)
    show {x∈K. ¬ vert-adj x u} ⊆ {a}
    proof
      fix b
      assume bA: b ∈ {x∈K. ¬ vert-adj x u}
      show b ∈ {a}
      proof (rule ccontr)
        assume bneq: b ∉ {a}
        then have ab-ne: a ≠ b
          by auto
        have bK: b ∈ K and bNu: ¬ vert-adj b u
          using bA by auto
        have bB: b ∈ {x∈K. ¬ vert-adj x v}
          using AB bA by blast
        have bNv: ¬ vert-adj b v
          using bB by auto
        have uv-ne: u ≠ v
          using vert-adj-neq[OF uv] .
        have Kclq: is-clique K
          using Kmax by (auto simp: maximum-clique-def cliques-def)
        have ab: vert-adj a b
          using clique-pair-adj[OF Kclq aK bK ab-ne] .
        have dist: distinct [a, b, u, v]
          using aK bK uO vO ab-ne uv-ne by auto
        have induces-2K2 a b u v
          using dist ab uv aNu aNv bNu bNv
          by (simp add: induces-2K2-def)
        then have has-induced-2K2
          by (auto simp: has-induced-2K2-def)
        with fh show False
          by (simp add: foldes-hammer-free-def)
      qed
```

```
    qed
  next
    show {a} ⊆ {x∈K. ¬ vert-adj x u}
      using aK aNu by auto
  qed
  show thesis
    using that Aeq aK aNu aNv by blast
qed

lemma orient-outside-edge-for-swap:
  assumes fh: foldes-hammer-free
    and Kmax: maximum-clique K
    and uO: u ∈ V − K
    and vO: v ∈ V − K
    and uv: vert-adj u v
  obtains u′ v′ a where
      {u′, v′} = {u, v}
    and u′ ∈ V − K
    and v′ ∈ V − K
    and vert-adj u′ v′
    and {x∈K. ¬ vert-adj x u′} = {a}
    and a ∈ K
    and ¬ vert-adj a u′
    and ¬ vert-adj a v′
proof −
  have nested:
    {x∈K. ¬ vert-adj x u} ⊆ {x∈K. ¬ vert-adj x v} ∨
     {x∈K. ¬ vert-adj x v} ⊆ {x∈K. ¬ vert-adj x u}
    using outside-adjacent-nonneighbor-sets-nested[OF fh Kmax uO vO uv] .
  then show thesis
  proof
    assume AB: {x∈K. ¬ vert-adj x u} ⊆ {x∈K. ¬ vert-adj x v}
    obtain a where Aeq: {x∈K. ¬ vert-adj x u} = {a} and aK: a ∈ K
      and aNu: ¬ vert-adj a u and aNv: ¬ vert-adj a v
      using outside-adjacent-smaller-nonneighbor-set-singleton[OF fh Kmax uO vO
uv AB] by blast
    show thesis
      using that[of u v a] uO vO uv Aeq aK aNu aNv by simp
  next
    assume BA: {x∈K. ¬ vert-adj x v} ⊆ {x∈K. ¬ vert-adj x u}
    have vu: vert-adj v u
      using uv by (simp add: vert-adj-sym)
    obtain a where Beq: {x∈K. ¬ vert-adj x v} = {a} and aK: a ∈ K
      and aNv: ¬ vert-adj a v and aNu: ¬ vert-adj a u
      using outside-adjacent-smaller-nonneighbor-set-singleton[OF fh Kmax vO uO
vu BA] by blast
    show thesis
      using that[of v u a] vO uO vu Beq aK aNv aNu by auto
  qed
```

qed

lemma *replace-unique-nonneighbor-maximum-clique*:
assumes *Kmax*: *maximum-clique K*
and *uO*: $u \in V - K$
and *Au*: $\{x{\in}K.\ \neg$ *vert-adj x u*$\} = \{a\}$
shows *maximum-clique* (*insert u* $(K - \{a\})$)
proof −
have *Kclq*: *is-clique K*
using *Kmax* **by** (*auto simp*: *maximum-clique-def cliques-def*)
have *aK*: $a \in K$ **and** *aNu*: $\neg$ *vert-adj a u*
using *Au* **by** *auto*
have *uV*: $u \in V$ **and** *unotK*: $u \notin K$
using *uO* **by** *auto*
have *Ksub*: $K - \{a\} \subseteq K$
by *auto*
have *Kminclq*: *is-clique* $(K - \{a\})$
using *clique-mono*[*OF Kclq Ksub*] **.**
have *uadj*: $\forall\, x{\in}K - \{a\}.$ *vert-adj x u*
proof
fix *x*
assume *xK*: $x \in K - \{a\}$
have *xK0*: $x \in K$ **and** *xnea*: $x \neq a$
using *xK* **by** *auto*
have $x \notin \{x{\in}K.\ \neg$ *vert-adj x u*$\}$
using *Au xnea* **by** *auto*
then show *vert-adj x u*
using *xK0* **by** *auto*
qed
have *newclq*: *is-clique* (*insert u* $(K - \{a\})$)
using *insert-clique-if-all-adj*[*OF Kminclq uV*] *unotK uadj* **by** *auto*
have *finK*: *finite K*
using *maximum-clique-finite*[*OF Kmax*] **.**
have *newcard*: *card* (*insert u* $(K - \{a\})) =$ *card K*
proof −
have *u-not-Ka*: $u \notin K - \{a\}$
using *unotK* **by** *auto*
have *card* (*insert u* $(K - \{a\})) =$ *Suc* (*card* $(K - \{a\})$)
using *finK u-not-Ka* **by** *simp*
also have ... = *Suc* (*card K* − *1*)
using *finK aK* **by** *simp*
also have ... = *card K*
proof −
have $K \neq \{\}$
using *aK* **by** *auto*
have *0* < *card K*
using *finK* ‹$K \neq \{\}$› **by** (*simp add*: *card-gt-0-iff*)
then show *?thesis*
by *arith*

```
    qed
    finally show ?thesis .
  qed
  have newin: insert u (K − {a}) ∈ cliques
    using newclq by (simp add: cliques-def)
  have newbound: ∀ C∈cliques. card C ≤ card (insert u (K − {a}))
  proof
    fix C
    assume Cin: C ∈ cliques
    from Kmax have ∀ C∈cliques. card C ≤ card K
      by (simp add: maximum-clique-def)
    then show card C ≤ card (insert u (K − {a}))
      using Cin newcard by simp
  qed
  show ?thesis
    using newin newbound by (simp add: maximum-clique-def)
qed

lemma replacement-edge-transfer:
  assumes fh: foldes-hammer-free
    and Kmax: maximum-clique K
    and uO: u ∈ V − K
    and vO: v ∈ V − K
    and uv: vert-adj u v
    and Au: {x∈K. ¬ vert-adj x u} = {a}
    and aNv: ¬ vert-adj a v
    and zO: z ∈ V − K
    and zne: z ≠ u
    and az: vert-adj a z
  shows vert-adj u z
proof (rule ccontr)
  assume uz: ¬ vert-adj u z
  have aK: a ∈ K and aNu: ¬ vert-adj a u
    using Au by auto
  have uv-ne: u ≠ v
    using vert-adj-neq[OF uv] .
  have vz: vert-adj v z
  proof (rule ccontr)
    assume vz-not: ¬ vert-adj v z
    have zv-ne: z ≠ v
      using az aNv by auto
    have dist: distinct [a, z, u, v]
      using aK uO vO zO uv-ne zne az zv-ne by auto
    have zNu: ¬ vert-adj z u
      using uz by (simp add: vert-adj-sym)
    have zNv: ¬ vert-adj z v
      using vz-not by (simp add: vert-adj-sym)
    have induces-2K2 a z u v
      using dist az uv aNu aNv zNu zNv
```

```
    by (simp add: induces-2K2-def)
  then have has-induced-2K2
    by (auto simp: has-induced-2K2-def)
  with fh show False
    by (simp add: foldes-hammer-free-def)
qed
have zv: vert-adj z v
  using vz by (simp add: vert-adj-sym)
have nested:
  {x∈K. ¬ vert-adj x z} ⊆ {x∈K. ¬ vert-adj x v} ∨
   {x∈K. ¬ vert-adj x v} ⊆ {x∈K. ¬ vert-adj x z}
  using outside-adjacent-nonneighbor-sets-nested[OF fh Kmax zO vO zv] .
have zV: z ∈ V and znotK: z ∉ K
  using zO by auto
obtain c where cK: c ∈ K and cNz: ¬ vert-adj c z
  using maximum-clique-outside-has-nonneighbor[OF Kmax zV znotK] by blast
have CsubB: {x∈K. ¬ vert-adj x z} ⊆ {x∈K. ¬ vert-adj x v}
proof −
  from nested show ?thesis
  proof
    assume h: {x∈K. ¬ vert-adj x z} ⊆ {x∈K. ¬ vert-adj x v}
    show ?thesis by fact
  next
    assume h: {x∈K. ¬ vert-adj x v} ⊆ {x∈K. ¬ vert-adj x z}
    have a ∈ {x∈K. ¬ vert-adj x z}
      using h aK aNv by auto
    then have aNz: ¬ vert-adj a z
      by auto
    then have False
      using az by contradiction
    then show ?thesis
      by blast
  qed
qed
have cNv: ¬ vert-adj c v
  using CsubB cK cNz by auto
have ca-ne: c ≠ a
  using az cK cNz by auto
have cu: vert-adj c u
proof −
  have c ∉ {x∈K. ¬ vert-adj x u}
    using Au ca-ne by auto
  then show ?thesis
    using cK by auto
qed
have Kclq: is-clique K
  using Kmax by (auto simp: maximum-clique-def cliques-def)
have ac: vert-adj a c
  using clique-pair-adj[OF Kclq aK cK ca-ne[symmetric]] .
```

```
  have zv: vert-adj z v
    using vz by (simp add: vert-adj-sym)
  have vu: vert-adj v u
    using uv by (simp add: vert-adj-sym)
  have uc: vert-adj u c
    using cu by (simp add: vert-adj-sym)
  have ca: vert-adj c a
    using ac by (simp add: vert-adj-sym)
  have zNu: ¬ vert-adj z u
    using uz by (simp add: vert-adj-sym)
  have zNc: ¬ vert-adj z c
    using cNz by (simp add: vert-adj-sym)
  have vNc: ¬ vert-adj v c
    using cNv by (simp add: vert-adj-sym)
  have dist: distinct [a, z, v, u, c]
    using aK cK uO vO zO uv-ne zne az ca-ne aNv by auto
  have induces-C5 a z v u c
    using dist az zv vu uc ca aNv aNu zNu zNc vNc
    by (simp add: induces-C5-def)
  then have has-induced-C5
    unfolding has-induced-C5-def by blast
  with fh show False
    by (simp add: foldes-hammer-free-def)
qed

lemma outside-set-after-swap:
  assumes uO: u ∈ V − K
    and aV: a ∈ V
    and aK: a ∈ K
  shows V − insert u (K − {a}) = insert a ((V − K) − {u})
  using assms by auto

lemma swap-outside-edges-inj:
  assumes uO: u ∈ V − K
    and aK: a ∈ K
  shows inj-on (%e. if a ∈ e then insert u (e − {a}) else e)
      (induced-edges (V − insert u (K − {a})))
proof (rule inj-onI)
  let ?I' = V − insert u (K − {a})
  let ?f = %e. if a ∈ e then insert u (e − {a}) else e
  fix e1 e2
  assume e1I: e1 ∈ induced-edges ?I'
    and e2I: e2 ∈ induced-edges ?I'
    and eq: ?f e1 = ?f e2
  have u-not-I': u ∉ ?I'
    using uO by auto
  have e1ss: e1 ⊆ ?I' and e2ss: e2 ⊆ ?I'
    using e1I e2I by (auto simp: induced-edges-def)
  show e1 = e2
```

```
proof (cases a ∈ e1)
  case False
  then have f1: ?f e1 = e1
    by simp
  show ?thesis
  proof (cases a ∈ e2)
    case False
    with eq f1 show ?thesis
      by simp
  next
    case True
    then have u ∈ ?f e2
      by simp
    then have u ∈ e1
      using eq f1 by simp
    with e1ss u-not-I' show ?thesis
      by blast
  qed
next
  case e1-has-a: True
  show ?thesis
  proof (cases a ∈ e2)
    case False
    then have f2: ?f e2 = e2
      by simp
    have u ∈ ?f e1
      using e1-has-a by simp
    then have u ∈ e2
      using eq f2 by simp
    with e2ss u-not-I' show ?thesis
      by blast
  next
    case e2-has-a: True
    have u-not-e1: u ∉ e1 − {a} and u-not-e2: u ∉ e2 − {a}
      using e1ss e2ss u-not-I' by blast+
    have insert u (e1 − {a}) = insert u (e2 − {a})
      using eq e1-has-a e2-has-a by simp
    then have diff-eq: e1 − {a} = e2 − {a}
    proof −
      have (insert u (e1 − {a})) − {u} = (insert u (e2 − {a})) − {u}
        using ‹insert u (e1 − {a}) = insert u (e2 − {a})› by simp
      then show ?thesis
        using u-not-e1 u-not-e2 by auto
    qed
    have e1 = insert a (e1 − {a})
      using e1-has-a by blast
    moreover have e2 = insert a (e2 − {a})
      using e2-has-a by blast
    ultimately show ?thesis
```

```
      using diff-eq by simp
    qed
  qed
qed

lemma swap-outside-edges-image-subset:
  assumes fh: foldes-hammer-free
    and Kmax: maximum-clique K
    and uO: u ∈ V − K
    and vO: v ∈ V − K
    and uv: vert-adj u v
    and Au: {x∈K. ¬ vert-adj x u} = {a}
    and aNv: ¬ vert-adj a v
  shows (%e. if a ∈ e then insert u (e − {a}) else e) ‘
      induced-edges (V − insert u (K − {a}))
      ⊆ induced-edges (V − K)
proof
  let ?I = V − K
  let ?I′ = V − insert u (K − {a})
  let ?f = %e. if a ∈ e then insert u (e − {a}) else e
  have aK: a ∈ K
    using Au by auto
  have aV: a ∈ V
    using Kmax aK by (auto simp: maximum-clique-def cliques-def is-clique-def)
  have Ieq: ?I′ = insert a (?I − {u})
    using outside-set-after-swap[OF uO aV aK] .
  fix x
  assume xin: x ∈ ?f ‘ induced-edges ?I′
  then obtain e where eI: e ∈ induced-edges ?I′ and xeq: x = ?f e
    by blast
  have ess: e ⊆ ?I′ and eE: e ∈ E
    using eI by (auto simp: induced-edges-def)
  show x ∈ induced-edges ?I
  proof (cases a ∈ e)
    case False
    then have xeqe: x = e
      using xeq by simp
    have e ⊆ ?I
    proof
      fix y
      assume y ∈ e
      with ess False show y ∈ ?I
        by (auto simp: Ieq)
    qed
    with eE xeqe show ?thesis
      by (auto simp: induced-edges-def)
  next
    case True
    from eE have ecard: card e = 2
```

```
    using two-edges by auto
  from True ecard obtain z where e: e = {a, z} and zne: z ≠ a
    by (auto simp: card-2-iff)
  have zI: z ∈ ?I − {u}
    using ess True e zne by (auto simp: Ieq)
  then have zO: z ∈ V − K and zne-u: z ≠ u
    by auto
  have az: vert-adj a z
    using eE e by (simp add: vert-adj-def)
  have uz: vert-adj u z
    using replacement-edge-transfer[OF fh Kmax uO vO uv Au aNv zO zne-u az]
.
  have xeqz: x = {u, z}
    using xeq True e zne by simp
  have uzE: {u, z} ∈ E
    using uz by (simp add: vert-adj-def)
  have uz-sub: {u, z} ⊆ ?I
    using uO zO by auto
  show ?thesis
    using xeqz uzE uz-sub by (auto simp: induced-edges-def)
 qed
qed

lemma swap-outside-edges-missing-uv:
 assumes uO: u ∈ V − K
   and vO: v ∈ V − K
   and uv: vert-adj u v
   and aK: a ∈ K
   and aNv: ¬ vert-adj a v
 shows {u, v} ∈ induced-edges (V − K)
   and {u, v} ∉ (%e. if a ∈ e then insert u (e − {a}) else e) ‘
     induced-edges (V − insert u (K − {a}))
proof −
 have uv-in: {u, v} ∈ induced-edges (V − K)
   using uO vO uv by (auto simp: induced-edges-def vert-adj-def)
 show {u, v} ∈ induced-edges (V − K)
   using uv-in .
 show {u, v} ∉ (%e. if a ∈ e then insert u (e − {a}) else e) ‘
     induced-edges (V − insert u (K − {a}))
 proof
   let ?I′ = V − insert u (K − {a})
   let ?f = %e. if a ∈ e then insert u (e − {a}) else e
   assume img: {u, v} ∈ ?f ‘ induced-edges ?I′
   then obtain e where eI: e ∈ induced-edges ?I′ and eq: {u, v} = ?f e
     by blast
   have u-not-I′: u ∉ ?I′
     using uO by auto
   have ess: e ⊆ ?I′ and eE: e ∈ E
     using eI by (auto simp: induced-edges-def)
```

```
    have uv-ne: u ≠ v
      using vert-adj-neq[OF uv] .
    show False
    proof (cases a ∈ e)
      case False
      then have {u, v} = e
        using eq by simp
      then have u ∈ e
        by auto
      with ess u-not-I' show False
        by blast
    next
      case True
      from eE have ecard: card e = 2
        using two-edges by auto
      from True ecard obtain z where e: e = {a, z} and zne: z ≠ a
        by (auto simp: card-2-iff)
      have zI': z ∈ ?I'
        using ess True e zne by auto
      then have zne-u: z ≠ u
        using u-not-I' by auto
      have {u, v} = {u, z}
        using eq True e zne by simp
      then have zeqv: z = v
        using uv-ne zne-u by auto
      have vert-adj a z
        using eE e by (simp add: vert-adj-def)
      with aNv zeqv show False
        by simp
    qed
  qed
qed

lemma outside-edge-count-swap-strict:
  assumes fh: foldes-hammer-free
    and Kmax: maximum-clique K
    and uO: u ∈ V − K
    and vO: v ∈ V − K
    and uv: vert-adj u v
    and Au: {x∈K. ¬ vert-adj x u} = {a}
    and aNv: ¬ vert-adj a v
  shows outside-edge-count (insert u (K − {a})) < outside-edge-count K
proof −
  let ?I = V − K
  let ?I' = V − insert u (K − {a})
  let ?f = %e. if a ∈ e then insert u (e − {a}) else e
  have aK: a ∈ K
    using Au by auto
  have inj: inj-on ?f (induced-edges ?I')
```

```
    using swap-outside-edges-inj[OF uO aK] .
  have imgsub: ?f ‘ induced-edges ?I' ⊆ induced-edges ?I
    using swap-outside-edges-image-subset[OF fh Kmax uO vO uv Au aNv] .
  have uv-in: {u, v} ∈ induced-edges ?I
    using swap-outside-edges-missing-uv[OF uO vO uv aK aNv] by blast
  have uv-not-img: {u, v} ∉ ?f ‘ induced-edges ?I'
    using swap-outside-edges-missing-uv[OF uO vO uv aK aNv] by blast
  have psub: ?f ‘ induced-edges ?I' ⊂ induced-edges ?I
    using imgsub uv-in uv-not-img by blast
  have finI': finite (induced-edges ?I')
    by (rule finite-induced-edges-subset-V) auto
  have finI: finite (induced-edges ?I)
    by (rule finite-induced-edges-subset-V) auto
  have card-img: card (?f ‘ induced-edges ?I') = card (induced-edges ?I')
    by (rule card-image[OF inj])
  have card (induced-edges ?I') = card (?f ‘ induced-edges ?I')
    using card-img by simp
  also have ... < card (induced-edges ?I)
    using finI psub by (rule psubset-card-mono)
  finally show ?thesis
    by (simp add: outside-edge-count-def)
qed

theorem foldes-hammer-converse:
  assumes fh: foldes-hammer-free
  shows is-split-graph
proof −
  obtain K where Kmax: maximum-clique K
      and Kbest: ∀ L. maximum-clique L ⟶ outside-edge-count K ≤ out-
side-edge-count L
    using obtain-best-clique by blast
  have Kclq: is-clique K
    using Kmax by (auto simp: maximum-clique-def cliques-def)
  have indep: is-independent-set (V − K)
  proof (rule ccontr)
    assume not-indep: ¬ is-independent-set (V − K)
   then obtain u v where uO: u ∈ V − K and vO: v ∈ V − K and uv: vert-adj
u v
      by (auto simp: is-independent-alt)
    obtain u' v' a where uvset: {u', v'} = {u, v}
      and u'O: u' ∈ V − K and v'O: v' ∈ V − K and uv': vert-adj u' v'
      and Au': {x∈K. ¬ vert-adj x u'} = {a}
      and aK: a ∈ K and aNu': ¬ vert-adj a u' and aNv': ¬ vert-adj a v'
      using orient-outside-edge-for-swap[OF fh Kmax uO vO uv] by blast
    let ?K' = insert u' (K − {a})
    have K'max: maximum-clique ?K'
      using replace-unique-nonneighbor-maximum-clique[OF Kmax u'O Au'] .
    have dec: outside-edge-count ?K' < outside-edge-count K
      using outside-edge-count-swap-strict[OF fh Kmax u'O v'O uv' Au' aNv'] .
```

```
    have outside-edge-count K ≤ outside-edge-count ?K'
      using Kbest K'max by blast
    with dec show False
      by simp
  qed
  show ?thesis
    using Kclq indep by (auto simp: split-graph-alt)
qed

theorem foldes-hammer:
  is-split-graph ⟷ foldes-hammer-free
  using foldes-hammer-converse split-graph-imp-foldes-hammer-free by blast

end

end
```

References

[1] A. Brandstädt, V. B. Le, and J. P. Spinrad. *Graph Classes: A Survey*. SIAM Monographs on Discrete Mathematics and Applications. Society for Industrial and Applied Mathematics, Philadelphia, PA, 1999. DOI: https://doi.org/10.1137/1.9780898719796.

[2] S. Földes and P. L. Hammer. Split graphs. *Congressus Numerantium*, 19:311–315, 1977.