Towards a constructive formalization of Perfect Graph Theorems

Abhishek Kr Singh   Raja Natarajan

School of Technology and Computer Science
Tata Institute of Fundamental Research, Mumbai.

Indian Conference on Logic and its Applications, 2019

Overview.

- Perfect Graph Theorems
  - Strong Perfect Graph Theorem
  - Weak Perfect Graph Theorem
- Modeling Finite Simple Graphs in Coq
- Constructive proof of Lovász Replication Lemma
- Graph Isomorphism and Graph Constructions
- Conclusions and Future Work

Overview.

- Perfect Graph Theorems
  - Strong Perfect Graph Theorem
  - Weak Perfect Graph Theorem
- Modeling Finite Simple Graphs in Coq
- Constructive proof of Lovász Replication Lemma
- Graph Isomorphism and Graph Constructions
- Conclusions and Future Work

Overview.

- Perfect Graph Theorems
  - Strong Perfect Graph Theorem
  - Weak Perfect Graph Theorem
- Modeling Finite Simple Graphs in Coq
- Constructive proof of Lovász Replication Lemma
- Graph Isomorphism and Graph Constructions
- Conclusions and Future Work

Overview.

- Perfect Graph Theorems
  - Strong Perfect Graph Theorem
  - Weak Perfect Graph Theorem
- Modeling Finite Simple Graphs in Coq
- Constructive proof of Lovász Replication Lemma
- Graph Isomorphism and Graph Constructions
- Conclusions and Future Work

Overview.

- Perfect Graph Theorems
  - Strong Perfect Graph Theorem
  - Weak Perfect Graph Theorem
- Modeling Finite Simple Graphs in Coq
- Constructive proof of Lovász Replication Lemma
- Graph Isomorphism and Graph Constructions
- Conclusions and Future Work

Overview.

- Perfect Graph Theorems
  - Strong Perfect Graph Theorem
  - Weak Perfect Graph Theorem
- Modeling Finite Simple Graphs in Coq
- Constructive proof of Lovász Replication Lemma
- Graph Isomorphism and Graph Constructions
- Conclusions and Future Work

Clique number $\omega(G)$ and Chromatic number $\chi(G)$

- Chromatic number $\chi(G)$: min num of colours to color $V(G)$.
- Clique number $\omega(G)$: size of largest clique in G.

- $\omega(G)$ is an obvious lower bound for $\chi(G)$ (i.e. $\chi(G) \geq \omega(G)$)
- In each of the above cases $\chi(G) = \omega(G)$, i.e. the number of colours needed is the minimum we can hope.
- Can we always hope $\chi(G) = \omega(G)$ for every graph G?

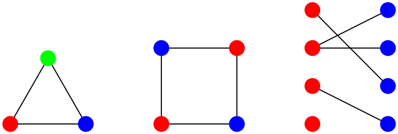
Clique number $\omega(G)$ and Chromatic number $\chi(G)$

- Chromatic number $\chi(G)$: min num of colours to color $V(G)$.
- Clique number $\omega(G)$: size of largest clique in G.

- $\omega(G)$ is an obvious lower bound for $\chi(G)$ (i.e. $\chi(G) \geq \omega(G)$)
- In each of the above cases $\chi(G) = \omega(G)$, i.e. the number of colours needed is the minimum we can hope.
- Can we always hope $\chi(G) = \omega(G)$ for every graph G?

Clique number $\omega(G)$ and Chromatic number $\chi(G)$

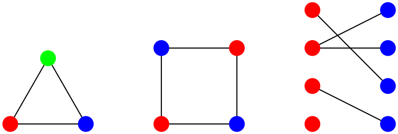
- Chromatic number $\chi(G)$: min num of colours to color $V(G)$.
- Clique number $\omega(G)$: size of largest clique in G.



- $\omega(G)$ is an obvious lower bound for $\chi(G)$ (i.e. $\chi(G) \geq \omega(G)$)
- In each of the above cases $\chi(G) = \omega(G)$, i.e. the number of colours needed is the minimum we can hope.
- Can we always hope $\chi(G) = \omega(G)$ for every graph G?

Clique number $\omega(G)$ and Chromatic number $\chi(G)$

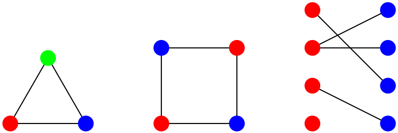
- Chromatic number $\chi(G)$: min num of colours to color $V(G)$.
- Clique number $\omega(G)$: size of largest clique in G.



- $\omega(G)$ is an obvious lower bound for $\chi(G)$ (i.e. $\chi(G) \geq \omega(G)$)
- In each of the above cases $\chi(G) = \omega(G)$, i.e. the number of colours needed is the minimum we can hope.
- Can we always hope $\chi(G) = \omega(G)$ for every graph G?

Clique number $\omega(G)$ and Chromatic number $\chi(G)$

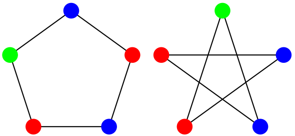
- Chromatic number $\chi(G)$: min num of colours to color $V(G)$.
- Clique number $\omega(G)$: size of largest clique in G.



- $\omega(G)$ is an obvious lower bound for $\chi(G)$ (i.e. $\chi(G) \geq \omega(G)$)
- In each of the above cases $\chi(G) = \omega(G)$, i.e. the number of colours needed is the minimum we can hope.
- Can we always hope $\chi(G) = \omega(G)$ for every graph G?

Odd holes and Odd anti-holes

- Consider the cycle of odd length 5 and its complement. In this case one can see that $\chi(G) = 3$ and $\omega(G) = 2$ (i.e. $\chi(G) > \omega(G)$).

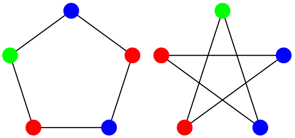


- The gap between $\chi(G)$ and $\omega(G)$ can be made arbitrarily large.

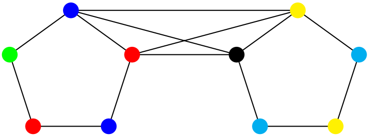
- We have k disjoint 5-cycles with all possible edges between any two copies. In this case one can show [3] that $\chi(G) = 3k$ but $\omega(G) = 2k$.

Odd holes and Odd anti-holes

- Consider the cycle of odd length 5 and its complement. In this case one can see that $\chi(G) = 3$ and $\omega(G) = 2$ (i.e. $\chi(G) > \omega(G)$).



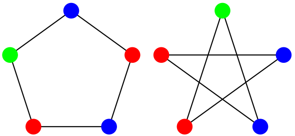
- The gap between $\chi(G)$ and $\omega(G)$ can be made arbitrarily large.



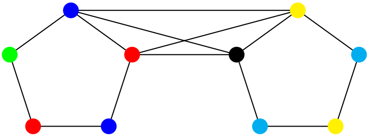
- We have k disjoint 5-cycles with all possible edges between any two copies. In this case one can show [3] that $\chi(G) = 3k$ but $\omega(G) = 2k$.

Odd holes and Odd anti-holes

- Consider the cycle of odd length 5 and its complement. In this case one can see that $\chi(G) = 3$ and $\omega(G) = 2$ (i.e. $\chi(G) > \omega(G)$).



- The gap between $\chi(G)$ and $\omega(G)$ can be made arbitrarily large.



- We have k disjoint 5-cycles with all possible edges between any two copies. In this case one can show [3] that $\chi(G) = 3k$ but $\omega(G) = 2k$.

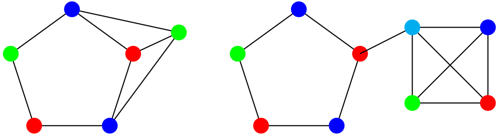
Perfect Graphs

- In 1961, Claude Berge noticed the presence of odd holes (or odd anti-holes) as induced subgraph in all the graphs presented to him that does not have a nice colouring, i.e. $\chi(G) > \omega(G)$.
- He also observed some graphs containing odd holes, where $\chi(G) = \omega(G)$.

- A good way to avoid such artificial construction is to make the notion of nice colouring hereditary.
- A graph G is called a *perfect graph* if $\chi(H) = \omega(H)$ for all of its induced subgraphs H.

Perfect Graphs

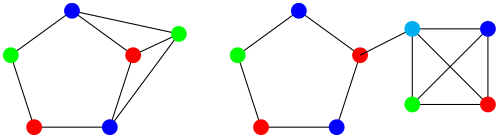
- In 1961, Claude Berge noticed the presence of odd holes (or odd anti-holes) as induced subgraph in all the graphs presented to him that does not have a nice colouring, i.e. $\chi(G) > \omega(G)$.
- He also observed some graphs containing odd holes, where $\chi(G) = \omega(G)$.



- A good way to avoid such artificial construction is to make the notion of nice colouring hereditary.
- A graph G is called a *perfect graph* if $\chi(H) = \omega(H)$ for all of its induced subgraphs H.

Perfect Graphs

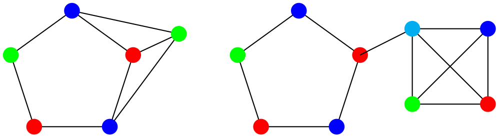
- In 1961, Claude Berge noticed the presence of odd holes (or odd anti-holes) as induced subgraph in all the graphs presented to him that does not have a nice colouring, i.e. $\chi(G) > \omega(G)$.
- He also observed some graphs containing odd holes, where $\chi(G) = \omega(G)$.



- A good way to avoid such artificial construction is to make the notion of nice colouring hereditary.
- A graph G is called a *perfect graph* if $\chi(H) = \omega(H)$ for all of its induced subgraphs H.

Perfect Graphs

- In 1961, Claude Berge noticed the presence of odd holes (or odd anti-holes) as induced subgraph in all the graphs presented to him that does not have a nice colouring, i.e. $\chi(G) > \omega(G)$.
- He also observed some graphs containing odd holes, where $\chi(G) = \omega(G)$.



- A good way to avoid such artificial construction is to make the notion of nice colouring hereditary.
- A graph G is called a *perfect graph* if $\chi(H) = \omega(H)$ for all of its induced subgraphs H.

Perfect Graph Theorems

- (SPGC): A graph is perfect if and only if it does not contain an odd hole (or an odd anti-hole) as its induced subgraph.
- (WPGC): a graph is perfect if and only if its complement is perfect.
- Lovász (in 1972) proved a result [4] known as Lovász Replication Lemma.
- It took however three more decades to come up with a proof for SPGC. The proof of Strong Perfect Graph Conjecture was announced in 2002 by Chudnovsky et al. and published [1] in a 178-page paper in 2006.

Perfect Graph Theorems

- (SPGC): A graph is perfect if and only if it does not contain an odd hole (or an odd anti-hole) as its induced subgraph.
- (WPGC): a graph is perfect if and only if its complement is perfect.
- Lovász (in 1972) proved a result [4] known as Lovász Replication Lemma.
- It took however three more decades to come up with a proof for SPGC. The proof of Strong Perfect Graph Conjecture was announced in 2002 by Chudnovsky et al. and published [1] in a 178-page paper in 2006.

Perfect Graph Theorems

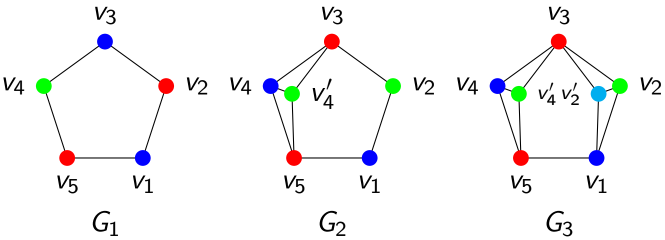
- (SPGC): A graph is perfect if and only if it does not contain an odd hole (or an odd anti-hole) as its induced subgraph.
- (WPGC): a graph is perfect if and only if its complement is perfect.
- Lovász (in 1972) proved a result [4] known as Lovász Replication Lemma.
- It took however three more decades to come up with a proof for SPGC. The proof of Strong Perfect Graph Conjecture was announced in 2002 by Chudnovsky et al. and published [1] in a 178-page paper in 2006.

Perfect Graph Theorems

- (SPGC): A graph is perfect if and only if it does not contain an odd hole (or an odd anti-hole) as its induced subgraph.
- (WPGC): a graph is perfect if and only if its complement is perfect.
- Lovász (in 1972) proved a result [4] known as Lovász Replication Lemma.
- It took however three more decades to come up with a proof for SPGC. The proof of Strong Perfect Graph Conjecture was announced in 2002 by Chudnovsky et al. and published [1] in a 178-page paper in 2006.

Modeling Finite Simple Graphs in Coq

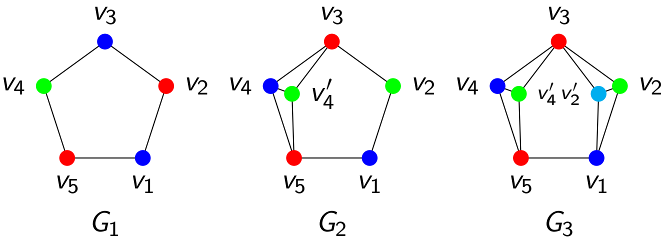
- All the graphs involved are finite simple graphs.



- Vertices as finite sets and edges as binary relation.
- The Mathematical Components library [2] (four color theorem).
- Finite sets using `finType`, `ffun` , and `reflect` predicate.
- Propositions on sets can be represented using computable (boolean) functions. Hence, case analysis on these propositions possible in a constructive way.

Modeling Finite Simple Graphs in Coq

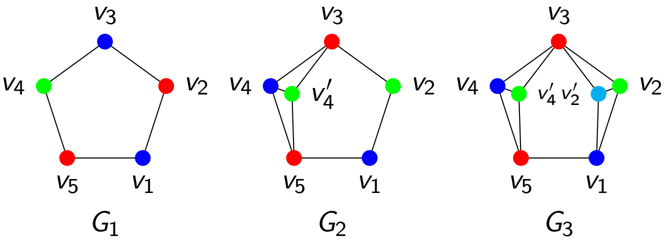
- All the graphs involved are finite simple graphs.



- Vertices as finite sets and edges as binary relation.
- The Mathematical Components library [2] (four color theorem).
- Finite sets using `finType`, `ffun` , and `reflect` predicate.
- Propositions on sets can be represented using computable (boolean) functions. Hence, case analysis on these propositions possible in a constructive way.

Modeling Finite Simple Graphs in Coq

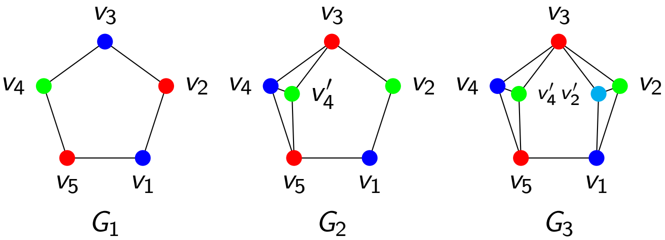
- All the graphs involved are finite simple graphs.



- Vertices as finite sets and edges as binary relation.
- The Mathematical Components library [2] (four color theorem).
- Finite sets using `finType`, `ffun` , and `reflect` predicate.
- Propositions on sets can be represented using computable (boolean) functions. Hence, case analysis on these propositions possible in a constructive way.

Modeling Finite Simple Graphs in Coq

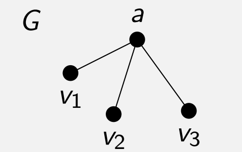
- All the graphs involved are finite simple graphs.

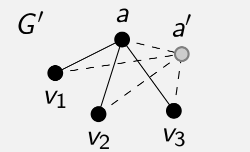


- Vertices as finite sets and edges as binary relation.
- The Mathematical Components library [2] (four color theorem).
- Finite sets using `finType`, `ffun` , and `reflect` predicate.
- Propositions on sets can be represented using computable (boolean) functions. Hence, case analysis on these propositions possible in a constructive way.

Lovász Replication Lemma

- The proof of WPGT involves expansion of graph.





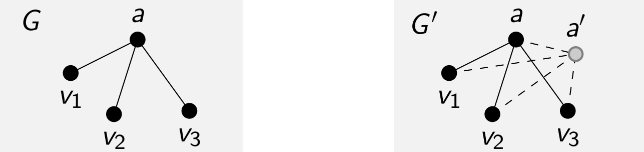
- G' is obtained from G by repeating vertex a
- Can't assume that the vertices of initial graph are sets on `finType`.

Lemma (Lovász Replication Lemma)

If G' is obtained from a perfect graph G by replicating a vertex, then G' is perfect

Lovász Replication Lemma

- The proof of WPGT involves expansion of graph.



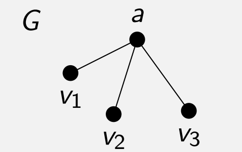
- G' is obtained from G by repeating vertex a
- Can't assume that the vertices of initial graph are sets on `finType`.

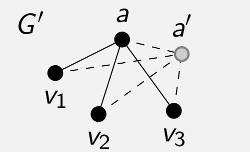
Lemma (Lovász Replication Lemma)

If G' is obtained from a perfect graph G by replicating a vertex, then G' is perfect

Lovász Replication Lemma

- The proof of WPGT involves expansion of graph.





- G' is obtained from G by repeating vertex a
- Can't assume that the vertices of initial graph are sets on `finType`.

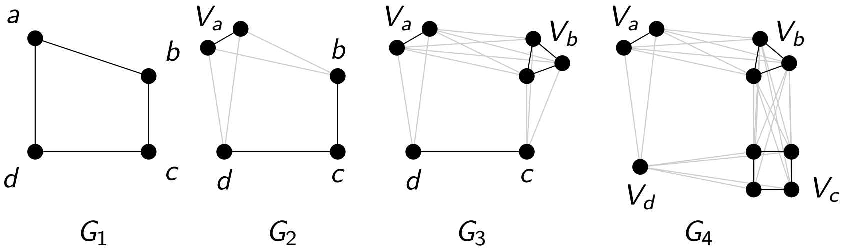
Lemma (Lovász Replication Lemma)

If G' is obtained from a perfect graph G by replicating a vertex, then G' is perfect

Generalised Lovász Replication Lemma

Lemma (Generalised Lovász Replication Lemma)

Let G be a perfect graph and $f : V(G) \to N$. Let G' be the graph obtained by replacing each vertex v_i of the graph G with a complete graph of order $f(v_i)$. Then G' is a perfect graph.



Modeling Finite Simple Graphs in Coq

- We define finite simple graphs as a dependent record with five fields.

```
Record UG (A:ordType) : Type:=  Build_UG {
        nodes :> list A;
        nodes_IsOrd : IsOrd nodes;
        edg: A -> A -> bool;
        edg_irefl: irefl edg;
        edg_sym: sym edg  }.
```

- ordType: type equiped with two boolean functions eqb and ltb.

Lemma (comparing x and y in ordType)

on_comp(x y:T):CompareSpec (x=y)(x <b y)(y <b x)(comp x y)

- Nodes are represented as an ordered list.

Modeling Finite Simple Graphs in Coq

- We define finite simple graphs as a dependent record with five fields.

```
Record UG (A:ordType) : Type:=  Build_UG {
        nodes :> list A;
        nodes_IsOrd : IsOrd nodes;
        edg: A -> A -> bool;
        edg_irefl: irefl edg;
        edg_sym: sym edg  }.
```

- `ordType`: type equiped with two boolean functions eqb and `ltb`.

Lemma (comparing x and y in ordType)

on_comp (x y:T):CompareSpec (x=y)(x <b y)(y <b x)(comp x y)

- Nodes are represented as an ordered list.

Modeling Finite Simple Graphs in Coq

- We define finite simple graphs as a dependent record with five fields.

```
Record UG (A:ordType) : Type:=  Build_UG {
        nodes :> list A;
        nodes_IsOrd : IsOrd nodes;
        edg: A -> A -> bool;
        edg_irefl: irefl edg;
        edg_sym: sym edg  }.
```

- `ordType`: type equiped with two boolean functions eqb and `ltb`.

Lemma (comparing x and y in ordType)

on_comp (x y:T):CompareSpec (x=y)(x <b y)(y <b x)(comp x y)

- Nodes are represented as an ordered list.

Vertices as constructive sets

- Vertices are sets over (A: `ordType`).

```
Record set_on  (A :ordType): Type := {
                 S_of :> list A;
                 IsOrd_S : IsOrd S_of }.
```

Lemma (Element wise equal sets are equal)

set_equal (A: ordType)(l s:set_on A): Equal l s -> l = s.

- Subsets of S are present in the list `pw(S)`.

Lemma (pw(S) is a set)

pw_is_ord (S: list A): IsOrd (pw S).

Vertices as constructive sets

- Vertices are sets over (A: `ordType`).

```
Record set_on  (A :ordType): Type := {
                S_of :> list A;
                IsOrd_S : IsOrd S_of }.
```

Lemma (Element wise equal sets are equal)

set_equal (A: ordType)(l s:set_on A): Equal l s -> l = s.

- Subsets of S are present in the list pw(S).

Lemma (pw(S) is a set)

pw_is_ord (S: list A): IsOrd (pw S).

Vertices as constructive sets

- Vertices are sets over (A: `ordType`).

```
Record set_on  (A :ordType): Type := {
                  S_of :> list A;
                  IsOrd_S : IsOrd S_of }.
```

Lemma (Element wise equal sets are equal)

set_equal (A: ordType)(l s:set_on A): Equal l s -> l = s.

- Subsets of `S` are present in the list `pw(S)`.

Lemma (pw(S) is a set)

pw_is_ord (S: list A): IsOrd (pw S).

Vertices as constructive sets

small scale reflections: reflect

Propositions	Boolean functions	Reflection lemmas
In a l	memb a l	membP
Equal l s	equal l s	equalP
∃ x, (In x l ∧ f x)	existsb f l	existsbP
∀ x, (In x l -> f x)	forallb f l	forallbP

- Reflection lemmas: Propositions connected with Boolean functions.

Lemma (Set membership is decidable)

membP a l: reflect (In a l) (memb a l).

Lemma (Case analysis using reflection lemmas)

reflect_EM (P: Prop)(b:bool): reflect P b -> P ∨ ¬ P.

Vertices as constructive sets

small scale reflections: reflect

Propositions	Boolean functions	Reflection lemmas
In a l	memb a l	membP
Equal l s	equal l s	equalP
∃ x, (In x l ∧ f x)	existsb f l	existsbP
∀ x, (In x l -> f x)	forallb f l	forallbP

- Reflection lemmas: Propositions connected with Boolean functions.

Lemma (Set membership is decidable)

membP a l: reflect (In a l) (memb a l).

Lemma (Case analysis using reflection lemmas)

reflect_EM (P: Prop)(b:bool): reflect P b -> P ∨ ¬ P.

Vertices as constructive sets

small scale reflections: reflect

Propositions	Boolean functions	Reflection lemmas
In a l	memb a l	membP
Equal l s	equal l s	equalP
∃ x, (In x l ∧ f x)	existsb f l	existsbP
∀ x, (In x l -> f x)	forallb f l	forallbP

- Reflection lemmas: Propositions connected with Boolean functions.

Lemma (Set membership is decidable)

membP a l: reflect (In a l) (memb a l).

Lemma (Case analysis using reflection lemmas)

reflect_EM (P: Prop)(b:bool): reflect P b -> P ∨ ¬ P.

Decidable edge relation

Clique, Stable set and Graph colouring

- Edges are represented using a decidable binary relation on the vertices.

Lemma (specification lemma for forall_xyb)

forall_xyP (P:A->A->bool) (l:list A): reflect (forall x y, In x l-> In y l-> P x y) (forall_xyb P l).

```
Definition cliq(G:UG)(K:list A):=
        forall_xyb (fun x y=> (x==y) || edg G x y) K.
Definition Cliq(G:UG)(K:list A):=
        (forall x y,In x K->In y K-> x=y \/ edg G x y).
```

Lemma (Cliq G K is decidable)

cliqP(G: UG)(K: list A): reflect (Cliq G K) (cliq G K).

Decidable edge relation

Clique, Stable set and Graph colouring

- Edges are represented using a decidable binary relation on the vertices.

Lemma (specification lemma for forall_xyb)

forall_xyP (P:A->A->bool) (l:list A): reflect (forall x y, In x l-> In y l-> P x y) (forall_xyb P l).

```
Definition cliq(G:UG)(K:list A):=
        forall_xyb (fun x y=> (x==y) || edg G x y) K.
Definition Cliq(G:UG)(K:list A):=
        (forall x y,In x K->In y K-> x=y \/ edg G x y).
```

Lemma (Cliq G K is decidable)

cliqP(G: UG)(K: list A): reflect (Cliq G K) (cliq G K).

Decidable edge relation

Clique, Stable set and Graph colouring

- Edges are represented using a decidable binary relation on the vertices.

Lemma (specification lemma for forall_xyb)

```
forall_xyP (P:A->A->bool) (l:list A): reflect (forall x y,
In x l-> In y l-> P x y) (forall_xyb P l).
```

```
Definition cliq(G:UG)(K:list A):=
        forall_xyb (fun x y=> (x==y) || edg G x y) K.
Definition Cliq(G:UG)(K:list A):=
        (forall x y,In x K->In y K-> x=y \/ edg G x y).
```

Lemma (Cliq G K is decidable)

```
cliqP(G: UG)(K: list A): reflect (Cliq G K) (cliq G K).
```

Decidable edge relation

Propositions	Boolean functions	Reflection lemmas
Subgraph G1 G2	subgraph G1 G2	subgraphP
Ind_Subgraph G1 G2	ind_subgraph G1 G2	ind_subgraphP
Stable G I	stable G I	stableP
Max_I_in G I	max_I_in G I	max_I_inP
Cliq G K	cliq G K	cliqP
Max_K_in G K	max_K_in G K	max_K_inP
Coloring_of G f	coloring_of G f	coloring_ofP

- Most of the properties are decidable for finite graphs.
- This makes case analysis on these predicates possible even though the Excluded Middle principle is not provable in Coq.

Decidable edge relation

Propositions	Boolean functions	Reflection lemmas
Subgraph G1 G2	subgraph G1 G2	subgraphP
Ind_Subgraph G1 G2	ind_subgraph G1 G2	ind_subgraphP
Stable G I	stable G I	stableP
Max_I_in G I	max_I_in G I	max_I_inP
Cliq G K	cliq G K	cliqP
Max_K_in G K	max_K_in G K	max_K_inP
Coloring_of G f	coloring_of G f	coloring_ofP

- Most of the properties are decidable for finite graphs.
- This makes case analysis on these predicates possible even though the Excluded Middle principle is not provable in Coq.

Decidable edge relation

Propositions	Boolean functions	Reflection lemmas
Subgraph G1 G2	subgraph G1 G2	subgraphP
Ind_Subgraph G1 G2	ind_subgraph G1 G2	ind_subgraphP
Stable G I	stable G I	stableP
Max_I_in G I	max_I_in G I	max_I_inP
Cliq G K	cliq G K	cliqP
Max_K_in G K	max_K_in G K	max_K_inP
Coloring_of G f	coloring_of G f	coloring_ofP

- Most of the properties are decidable for finite graphs.
- This makes case analysis on these predicates possible even though the Excluded Middle principle is not provable in Coq.

Constructive proof of Lovasz Replication Lemma

Lemma (Lovász Replication Lemma)

If G' is obtained from a perfect graph G by replicating a vertex, then G' is perfect

- Let `H'` be an induced subgraph of `G'`, then goal: χ`(H') =` ω`(H')`.
- Ind Hyp : `∀ X, |X|<|G| → Perfect X → Perfect X'`
- Case 1: `H' ≠ G'`
  - Case 1a: `a ∉ H'` : **Reason 1**
  - Case 1b: `a ∈ H'` : **Reason 2**
- Case 2: `H' = G'` (let `P K := max_K_in G K ∧ memb a K.`)
  - Case 2a: exists a clique `K` of size ω`(G)` such that `a ∈ K`.
    - (∃ x, In x (pw G) ∧ P x) : **Reason 3**
  - Case 2b: a does not belong to any clique `K` of size ω`(G)`.
    - (∀ x, In x (pw G) → P x)) : **Reason 4**
- All the cases correspond to predicates on sets and finite graphs.
- We have decidable representations for all of these predicates.

Constructive proof of Lovasz Replication Lemma

Lemma (Lovász Replication Lemma)

If G' is obtained from a perfect graph G by replicating a vertex, then G' is perfect

- Let `H'` be an induced subgraph of `G'`, then goal: χ`(H') = `ω`(H')`.
- `Ind Hyp` : `∀ X, |X|<|G| → Perfect X → Perfect X'`
- Case 1: `H'` $\neq$ `G'`
  - Case 1a: `a` $\notin$ `H'` : **Reason 1**
  - Case 1b: `a` $\in$ `H'` : **Reason 2**
- Case 2: `H' = G'` (let `P K := max_K_in G K ∧ memb a K.`)
  - Case 2a: exists a clique `K` of size ω`(G)` such that `a` $\in$ `K`.
    - (∃ x, In x (pw G) ∧ P x) : **Reason 3**
  - Case 2b: a does not belong to any clique `K` of size ω`(G)`.
    - (∀ x, In x (pw G) → P x)) : **Reason 4**
- All the cases correspond to predicates on sets and finite graphs.
- We have decidable representations for all of these predicates.

Constructive proof of Lovasz Replication Lemma

Lemma (Lovász Replication Lemma)

If G' is obtained from a perfect graph G by replicating a vertex, then G' is perfect

- Let H' be an induced subgraph of G', then goal: χ(H') = ω(H').
- Ind Hyp : $\forall$ X, |X|<|G| $\to$ Perfect X $\to$ Perfect X'
- Case 1: H' $\neq$ G'
  - Case 1a: a $\notin$ H' : **Reason 1**
  - Case 1b: a $\in$ H' : **Reason 2**
- Case 2: H' = G' (let P K := max_K_in G K $\wedge$ memb a K.)
  - Case 2a: exists a clique K of size ω(G) such that a $\in$ K.
    - ($\exists$ x, In x (pw G) $\wedge$ P x) : **Reason 3**
  - Case 2b: a does not belong to any clique K of size ω(G).
    - ($\forall$ x, In x (pw G) $\to$ P x)) : **Reason 4**
- All the cases correspond to predicates on sets and finite graphs.
- We have decidable representations for all of these predicates.

Constructive proof of Lovasz Replication Lemma

Lemma (Lovász Replication Lemma)

If G' is obtained from a perfect graph G by replicating a vertex, then G' is perfect

- Let `H'` be an induced subgraph of `G'`, then goal: χ`(H') =` ω`(H')`.
- Ind Hyp : `∀ X, |X|<|G| → Perfect X → Perfect X'`
- Case 1: `H' ≠ G'`
  - Case 1a: `a ∉ H'` : **Reason 1**
  - Case 1b: `a ∈ H'` : **Reason 2**
- Case 2: `H' = G'` (let `P K := max_K_in G K ∧ memb a K.`)
  - Case 2a: exists a clique `K` of size ω`(G)` such that `a ∈ K.`
    - (∃ x, In x (pw G) ∧ P x) : **Reason 3**
  - Case 2b: a does not belong to any clique `K` of size ω`(G)`.
    - (∀ x, In x (pw G) → P x)) : **Reason 4**
- All the cases correspond to predicates on sets and finite graphs.
- We have decidable representations for all of these predicates.

Constructive proof of Lovasz Replication Lemma

Lemma (Lovász Replication Lemma)

If G' is obtained from a perfect graph G by replicating a vertex, then G' is perfect

- Let `H'` be an induced subgraph of `G'`, then goal: χ`(H') =` ω`(H')`.
- `Ind Hyp` : $\forall$ `X, |X|<|G|` $\rightarrow$ `Perfect X` $\rightarrow$ `Perfect X'`
- Case 1: `H'` $\neq$ `G'`
  - Case 1a: `a` $\notin$ `H'` : **Reason 1**
  - Case 1b: `a` $\in$ `H'` : **Reason 2**
- Case 2: `H' = G'` (let `P K := max_K_in G K` $\wedge$ `memb a K.`)
  - Case 2a: exists a clique `K` of size ω`(G)` such that `a` $\in$ `K`.
    - ($\exists$ x, In x (pw G) $\wedge$ P x) : **Reason 3**
  - Case 2b: a does not belong to any clique `K` of size ω`(G)`.
    - ($\forall$ x, In x (pw G) $\rightarrow$ P x)) : **Reason 4**
- All the cases correspond to predicates on sets and finite graphs.
- We have decidable representations for all of these predicates.

Constructive proof of Lovasz Replication Lemma

Lemma (Lovász Replication Lemma)

If G' is obtained from a perfect graph G by replicating a vertex, then G' is perfect

- Let `H’` be an induced subgraph of `G’`, then goal: χ`(H’) =` ω`(H’)`.
- `Ind Hyp` : $\forall$ `X, |X|<|G|` $\rightarrow$ `Perfect X` $\rightarrow$ `Perfect X’`
- Case 1: `H’` $\neq$ `G’`
  - Case 1a: `a` $\notin$ `H’` : **Reason 1**
  - Case 1b: `a` $\in$ `H’` : **Reason 2**
- Case 2: `H’ = G’` (let `P K := max_K_in G K` $\wedge$ `memb a K.`)
  - Case 2a: exists a clique `K` of size ω`(G)` such that `a` $\in$ `K`.
    - ($\exists$ x, In x (pw G) $\wedge$ P x) : **Reason 3**
  - Case 2b: a does not belong to any clique `K` of size ω`(G)`.
    - ($\forall$ x, In x (pw G) $\rightarrow$ P x)) : **Reason 4**
- All the cases correspond to predicates on sets and finite graphs.
- We have decidable representations for all of these predicates.

Graph Isomorphism

- Proving equality of graph without assuming any axiom.

```
Record UG (A:ordType) : Type:=  Build_UG {
        nodes :> list A;
        nodes_IsOrd : IsOrd nodes;
        edg: A -> A -> bool;
        edg_irefl: irefl edg;
        edg_sym: sym edg  }.
```

```
Record UG (A:ordType) : Type:=  Build_UG {
        nodes :> list A;
        nodes_IsOrd : isOrd nodes;    ------> UIP
        edg: A -> A -> bool;
        edg_irefl: irefl_in nodes edg;   ---> UIP
        edg_sym: sym_in nodes edg  }. ------> UIP
```

Lemma (UIP: Uniqueness of Identity Proofs)

eq_proofs_unicity A (decA : ∀ x y : A, x = y ∨ x <> y) (x y: A) (p1 p2: x=y): p1=p2

Graph Isomorphism

- We need a proper representation for graph isomorphism.

```
Definition iso_using (f: A->A)(G G': @UG A) :=
        (forall x, f (f x) = x) /\
        (nodes G') = (img f G)  /\
        (forall x y, edg G x y = edg G' (f x) (f y)).
Definition iso (G G': @UG A) := exists f, iso_using f G G'.
```

- Self invertible nature of `f` which makes it injective on both `G` and `G'`.

Lemma (invertible nature of isomorphism)

iso_one_one (G G': UG)(f: A-> A): iso_using f G G'-> one_one_on G f.

Lemma (symmetric nature of isomorphism)

iso_sym (G G': UG): iso G G' -> iso G' G.

Graph Isomorphism

- We need a proper representation for graph isomorphism.

```
Definition iso_using (f: A->A)(G G': @UG A) :=
        (forall x, f (f x) = x) /\
        (nodes G') = (img f G)  /\
        (forall x y, edg G x y = edg G' (f x) (f y)).
Definition iso (G G': @UG A) := exists f, iso_using f G G'.
```

- Self invertible nature of `f` which makes it injective on both `G` and `G'`.

Lemma (invertible nature of isomorphism)

iso_one_one (G G': UG)(f: A-> A): iso_using f G G'-> one_one_on G f.

Lemma (symmetric nature of isomorphism)

iso_sym (G G': UG): iso G G' -> iso G' G.

Graph Isomorphism

Lemma (isomorphic counterpart)

```
iso_subgraphs (G G' H :UG) (f: A-> A) : iso_using f G G'->
Ind_subgraph H G -> (∃ H', Ind_subgraph H' G' ∧ iso_using f
H H').
```

- Every induced subgraph `H` of `G` has an isomorphic counterpart `H'` in `G'`.
- Results in isomorphic cliques, stable sets and same chromatic number.

Lemma (isomorphic stable set)

```
iso_stable (G G': UG)(f: A-> A)(I: list A):iso_using f G
G'-> Stable G I-> Stable G' (img f I).
```

Graph Isomorphism

Lemma (isomorphic counterpart)

```
iso_subgraphs (G G' H :UG) (f: A-> A) : iso_using f G G'->
Ind_subgraph H G -> (∃ H', Ind_subgraph H' G' ∧ iso_using f
H H').
```

- Every induced subgraph `H` of `G` has an isomorphic counterpart `H'` in `G'`.
- Results in isomorphic cliques, stable sets and same chromatic number.

Lemma (isomorphic stable set)

```
iso_stable (G G': UG)(f: A-> A)(I: list A):iso_using f G
G'-> Stable G I-> Stable G' (img f I).
```

Graph Isomorphism

Lemma (isomorphic cliques)

```
iso_cliq (G G': UG)(f:A-> A)(K:list A):iso_using f G G'->
Cliq G K -> Cliq G' (img f K).
```

Lemma (coloring of isomorphic graphs)

```
iso_coloring(G G':UG)(f:A->A)(C: A->nat):iso_using f G G'
-> Coloring_of G C -> Coloring_of G' (fun (x:A) => C (f x)).
```

Lemma (perfectness is preserved)

```
perfect_G' (G G':UG ): iso G G'-> Perfect G -> Perfect G'.
```

Graph Isomorphism

Lemma (isomorphic cliques)

```
iso_cliq (G G': UG)(f:A-> A)(K:list A):iso_using f G G'->
Cliq G K -> Cliq G' (img f K).
```

Lemma (coloring of isomorphic graphs)

```
iso_coloring(G G':UG)(f:A->A)(C: A->nat):iso_using f G G'
-> Coloring_of G C -> Coloring_of G' (fun (x:A) => C (f x)).
```

Lemma (perfectness is preserved)

```
perfect_G' (G G':UG ): iso G G'-> Perfect G -> Perfect G'.
```

Graph Isomorphism

Lemma (isomorphic cliques)

```
iso_cliq (G G': UG)(f:A-> A)(K:list A):iso_using f G G'->
Cliq G K -> Cliq G' (img f K).
```

Lemma (coloring of isomorphic graphs)

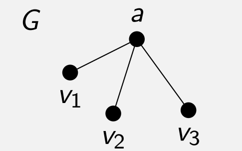
```
iso_coloring(G G':UG)(f:A->A)(C: A->nat):iso_using f G G'
-> Coloring_of G C -> Coloring_of G' (fun (x:A) => C (f x)).
```

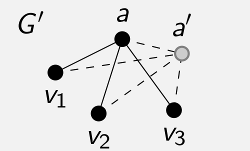
Lemma (perfectness is preserved)

```
perfect_G' (G G':UG ): iso G G'-> Perfect G -> Perfect G'.
```

Graph Constructions

- Adding (or removing) edges in an existing graph.



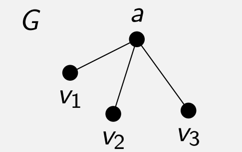


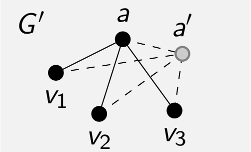

```
Definition nw_edg(G:UG)(a a':A):=
        fun(x y:A) => match (x==a), (y==a') with
                | _ , false => (edg G) x y
                | true, true => true
                | false, true => (edg G) x a
                end.
```

Lemma

nw_edg_xa_xa' (G: UG)(x:A): (edg G) x a -> (edg G') x a'.

Graph Constructions





Lemma

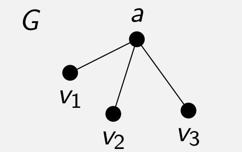
nw_edg_xy_xy (G: UG)(x y:A)(P': ¬ In a' G): (edg G) x y -> (edg G') x y

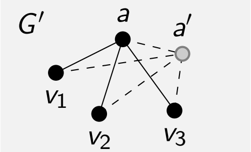
Lemma

nw_edg_xy_xy4 (G: UG)(x y:A)(P: In a G)(P': ¬In a' G): y ≠ a' -> (edg G) x y = (edg G') x y.

- we can't use `nw_edg(G:UG)(a a':A)` for edge relation while declaring `G'` as an instance of `UG`.

Graph Constructions





Lemma

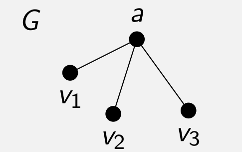
nw_edg_xy_xy (G: UG)(x y:A)(P': ¬ In a' G): (edg G) x y -> (edg G') x y

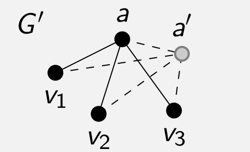
Lemma

nw_edg_xy_xy4 (G: UG)(x y:A)(P: In a G)(P': ¬In a' G): y ≠ a' -> (edg G) x y = (edg G') x y.

- we can't use `nw_edg(G:UG)(a a':A)` for edge relation while declaring `G'` as an instance of `UG`.

Graph Constructions

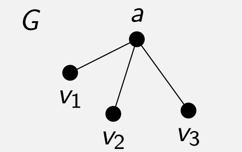


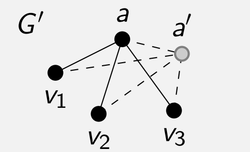



```
Definition nw_edg(G:UG)(a a':A):=
        fun(x y:A) => match (x==a), (y==a') with
                | _ , false => (edg G) x y
                | true, true => true
                | false, true => (edg G) x a
                end.
```

- We can add more branches to the match statement.
- Results in a more complex function and proving even essential properties becomes hard.
- we define functions namely `mk_irefl` and `mk_sym`.

Graph Constructions

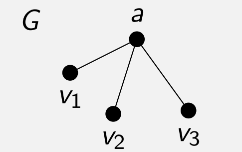


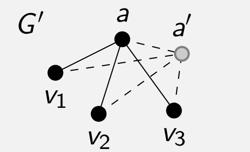



```
Definition nw_edg(G:UG)(a a':A):=
        fun(x y:A) => match (x==a), (y==a') with
                | _ , false => (edg G) x y
                | true, true => true
                | false, true => (edg G) x a
                end.
```

- We can add more branches to the match statement.
- Results in a more complex function and proving even essential properties becomes hard.
- we define functions namely `mk_irefl` and `mk_sym`.

Graph Constructions






```
Definition nw_edg(G:UG)(a a':A):=
        fun(x y:A) => match (x==a), (y==a') with
                | _ , false => (edg G) x y
                | true, true => true
                | false, true => (edg G) x a
                end.
```

- We can add more branches to the match statement.
- Results in a more complex function and proving even essential properties becomes hard.
- we define functions namely `mk_irefl` and `mk_sym`.

Graph Constructions

Lemma (specification of mk_irefl)

```
mk_ireflP (E: A -> A-> bool): irefl (mk_irefl E).
```

Lemma (specification of mk_sym)

```
mk_symP (E: A-> A-> bool): sym (mk_sym E).
```

- These functions do not change the properties ensured by each other.

Lemma (invariance lemma for mk_sym)

```
irefl_inv_for_mk_sym (E: A-> A-> bool): irefl E -> irefl
(mk_sym E).
```

Lemma (invariance lemma for mk_irefl)

```
sym_inv_for_mk_irefl (E: A->A-> bool): sym E -> sym
(mk_irefl E).
```

Graph Constructions

Lemma (specification of mk_irefl)

```
mk_ireflP (E: A -> A-> bool):  irefl (mk_irefl E).
```

Lemma (specification of mk_sym)

```
mk_symP (E: A-> A-> bool):  sym (mk_sym E).
```

- These functions do not change the properties ensured by each other.

Lemma (invariance lemma for mk_sym)

```
irefl_inv_for_mk_sym (E: A-> A-> bool):  irefl E -> irefl
(mk_sym E).
```

Lemma (invariance lemma for mk_irefl)

```
sym_inv_for_mk_irefl (E: A->A-> bool):  sym E -> sym
(mk_irefl E).
```

Graph Constructions

Lemma (specification of mk_irefl)

```
mk_ireflP (E: A -> A-> bool): irefl (mk_irefl E).
```

Lemma (specification of mk_sym)

```
mk_symP (E: A-> A-> bool): sym (mk_sym E).
```

- These functions do not change the properties ensured by each other.

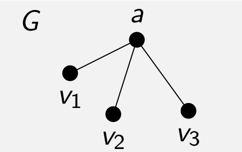
Lemma (invariance lemma for mk_sym)

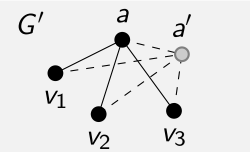
```
irefl_inv_for_mk_sym (E: A-> A-> bool): irefl E -> irefl
(mk_sym E).
```

Lemma (invariance lemma for mk_irefl)

```
sym_inv_for_mk_irefl (E: A->A-> bool): sym E -> sym
(mk_irefl E).
```

Graph Constructions






```
Definition ex_edg(G:UG)(a a':A):= mk_sym(mk_irefl(nw_edg G a a')).

Definition G':= refine({| nodes:= add a' G;
                          edg:= (ex_edg G a a'); |});
                          unfold ex_edg. all: auto. Defined.
```

- These functions can significantly ease the construction of new graphs.
- Tactic (`all:auto`) can discharge all the proof obligations generated while declaring `G'`.

Graph Constructions


```
Definition ex_edg(G:UG)(a a':A):= mk_sym(mk_irefl(nw_edg G a a')).

Definition G':= refine({| nodes:= add a' G;
                          edg:= (ex_edg G a a'); |});
                          unfold ex_edg. all: auto. Defined.
```

- These functions can significantly ease the construction of new graphs.
- Tactic (`all:auto`) can discharge all the proof obligations generated while declaring `G'`.

Conclusions

- Modeling finite graphs in a constructive way.
- Use of small scale reflection to obtain decidable predicates.
- A constructive proof of Lovász Replication Lemma
- Equality of graphs and graph isomorphism
- Functions to automate constructions of new graphs.
- Future Work:
  - An efficient way to represent Graph Expansion.
  - Constructive proof of WPGT.
  - Decompositions involved in the proof of SPGT.

Conclusions

- Modeling finite graphs in a constructive way.
- Use of small scale reflection to obtain decidable predicates.
- A constructive proof of Lovász Replication Lemma
- Equality of graphs and graph isomorphism
- Functions to automate constructions of new graphs.
- Future Work:
  - An efficient way to represent Graph Expansion.
  - Constructive proof of WPGT.
  - Decompositions involved in the proof of SPGT.

Conclusions

- Modeling finite graphs in a constructive way.
- Use of small scale reflection to obtain decidable predicates.
- A constructive proof of Lovász Replication Lemma
- Equality of graphs and graph isomorphism
- Functions to automate constructions of new graphs.
- Future Work:
  - An efficient way to represent Graph Expansion.
  - Constructive proof of WPGT.
  - Decompositions involved in the proof of SPGT.

Conclusions

- Modeling finite graphs in a constructive way.
- Use of small scale reflection to obtain decidable predicates.
- A constructive proof of Lovász Replication Lemma
- Equality of graphs and graph isomorphism
- Functions to automate constructions of new graphs.
- Future Work:
  - An efficient way to represent Graph Expansion.
  - Constructive proof of WPGT.
  - Decompositions involved in the proof of SPGT.

Conclusions

- Modeling finite graphs in a constructive way.
- Use of small scale reflection to obtain decidable predicates.
- A constructive proof of Lovász Replication Lemma
- Equality of graphs and graph isomorphism
- Functions to automate constructions of new graphs.
- Future Work:
  - An efficient way to represent Graph Expansion.
  - Constructive proof of WPGT.
  - Decompositions involved in the proof of SPGT.

Conclusions

- Modeling finite graphs in a constructive way.
- Use of small scale reflection to obtain decidable predicates.
- A constructive proof of Lovász Replication Lemma
- Equality of graphs and graph isomorphism
- Functions to automate constructions of new graphs.
- Future Work:
  - An efficient way to represent Graph Expansion.
  - Constructive proof of WPGT.
  - Decompositions involved in the proof of SPGT.

Conclusions

- Modeling finite graphs in a constructive way.
- Use of small scale reflection to obtain decidable predicates.
- A constructive proof of Lovász Replication Lemma
- Equality of graphs and graph isomorphism
- Functions to automate constructions of new graphs.
- Future Work:
  - An efficient way to represent Graph Expansion.
  - Constructive proof of WPGT.
  - Decompositions involved in the proof of SPGT.

Conclusions

- Modeling finite graphs in a constructive way.
- Use of small scale reflection to obtain decidable predicates.
- A constructive proof of Lovász Replication Lemma
- Equality of graphs and graph isomorphism
- Functions to automate constructions of new graphs.
- Future Work:
  - An efficient way to represent Graph Expansion.
  - Constructive proof of WPGT.
  - Decompositions involved in the proof of SPGT.

Conclusions

- Modeling finite graphs in a constructive way.
- Use of small scale reflection to obtain decidable predicates.
- A constructive proof of Lovász Replication Lemma
- Equality of graphs and graph isomorphism
- Functions to automate constructions of new graphs.
- Future Work:
  - An efficient way to represent Graph Expansion.
  - Constructive proof of WPGT.
  - Decompositions involved in the proof of SPGT.

References

Maria Chudnovsky, Neil Robertson, Paul Seymour, and Robin Thomas.
The strong perfect graph theorem.
ANNALS OF MATHEMATICS, 164:51–229, 2006.

Georges Gonthier and Assia Mahboubi.
An introduction to small scale reflection in Coq.
Journal of Formalized Reasoning, 3(2):95–152, 2010.

Andras Gyarfas, Andras Sebo, and Nicolas Trotignon.
The chromatic gap and its extremes.
Journal of Combinatorial Theory, Series B, 102(5):1155 – 1178, 2012.

L. Lovász.
Normal hypergraphs and the perfect graph conjecture.
Discrete Mathematics, 2(3):253 – 267, 1972.

Thank You !