Верификация доказательства теоремы о нижней оценке хроматического числа плоскости в системе Coq

Выступающий: Е.Б.Анюшева
Руководитель: к.ф.-м.н. Е.В.Дашков

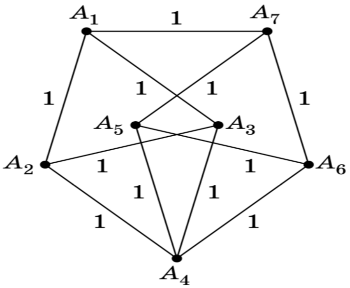
Факультет Инноваций и Высоких Технологий
МФТИ (ГУ)

Москва, 2019

Полный исходный код находится по адресу

https://github.com/LenaAn/deGrey_proofs

Задача о хроматическом числе плоскости



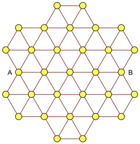
$\chi \geq 4$

1950 год

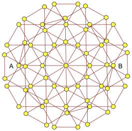
$\chi \leq 7$

1950 год

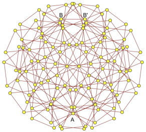
Хроматическое число плоскости не меньше 5



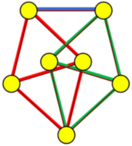
Граф J $\to$



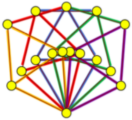
Граф K $\to$



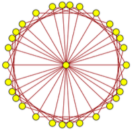
Граф L



Веретено $\to$



Веретена $\to$



Граф V $\to$ Граф N с 20425 вершинами

Текущее состояние задачи: верификация с помощью SAT-solver'а

- Найден меньший пример графа, не раскрашиваемого в 4 цвета, он имеет 1585 вершин
- Утверждение о том, что данный граф не раскрашивается в 4 цвета, записано в виде формулы первого порядка и сведено в задаче SAT
- С помощью SAT-solver'а проверено отсутствие раскраски в 4 цвета

Цели работы

- Разработка методов конструкций графов в системе Coq
- Формализация конструкций графов
- формализация утверждений статьи в системе Coq
- Верификация утверждений статьи

Система Coq

- Система для *верификации* доказательств теорем
- Использует *Исчесление индуктивных конструкций (Calculus of Inductive Constructions)*
- *Соответствие Карри-Ховарда*

Теорема	Терм
Утверждение	Тип
Доказательство	Терм данного типа

Актуальность компьютерной верификации

- Структура данных с несвязным множеством: доказательство корректности в Coq было опубликовано в 2007 году.
- Теорема Фейта – Томпсона: формальное доказательство с использованием Coq было завершено в сентябре 2012 года.
- Теорема о четырех цветах: формальное доказательство с использованием Coq было завершено в 2005 году.

Представление графов в Coq

```
Definition node := PositiveOrderedTypeBits.t.
Definition nodeset := PositiveSet.t.
Definition nodemap: Type -> Type :=
                             PositiveMap.t.
Definition graph := nodemap nodeset.
```

Представление графов в Coq

```
Definition K3 :=
    mk_graph [ (1, 2) ; (2, 3); (1, 3)].

Compute (S.elements (Mdomain K3)).

Function gr_show (g : graph) : list (node * node) :=
  S.fold
  (fun n l => (map (fun y => (n, y)) (S.elements (adj g n))) ++ l)
  (Mdomain g) nil.

Compute gr_show K3.
```

```
K3 is defined
```

Представление графов в Coq

```
Definition K3 :=
    mk_graph [ (1, 2) ; (2, 3); (1, 3)].

Compute (S.elements (Mdomain K3)).

Function gr_show (g : graph) : list (node * node) :=
  S.fold
  (fun n l => (map (fun y => (n, y)) (S.elements (adj g n))) ++ l)
  (Mdomain g) nil.

Compute gr_show K3.
```

```
     = [2; 1; 3]
     : list S.elt
```

Представление графов в Coq

```
Definition K3 :=
    mk_graph [ (1, 2) ; (2, 3); (1, 3)].

Compute (S.elements (Mdomain K3)).

Function gr_show (g : graph) : list (node * node) :=
  S.fold
  (fun n l => (map (fun y => (n, y)) (S.elements (adj g n))) ++ l)
  (Mdomain g) nil.

Compute gr_show K3.
```

```
gr_show is defined
```

Представление графов в Coq

```
Definition K3 :=
    mk_graph [ (1, 2) ; (2, 3); (1, 3)].

Compute (S.elements (Mdomain K3)).

Function gr_show (g : graph) : list (node * node) :=
  S.fold
  (fun n l => (map (fun y => (n, y)) (S.elements (adj g n))) ++ l)
  (Mdomain g) nil.

Compute gr_show K3.
```

```
     = [(3, 2); (3, 1); (1, 2); (1, 3); (2, 1); (2, 3)]
     : list (node * node)
```

Представление графов в Coq

```
Definition add_edge (e: (E.t*E.t)) (g: graph) :
                                            graph :=
 M.add (fst e) (S.add (snd e) (adj g (fst e)))
  (M.add (snd e) (S.add (fst e) (adj g (snd e))) g).

Definition mk_graph (el: list (E.t*E.t)) :=
  fold_right add_edge (M.empty _) el.

Definition K3 :=
    mk_graph [(1, 2); (2, 3); (1, 3)].
```

Представление графов в Coq: функции

Функции для работы с графами с высокой степенью симметрии:

- `l_rng (l : list node) : node * node`
- `gr_rng (g : graph) : node * node`
- `rename_all (f : node -> node) (g : graph) : graph`
- `delete_edge (g: graph) (a b : node) : graph`
- `rename_in_order (g: graph) : graph`
- `mk_cmn_edge (g1 g2 : graph) (a b n m : node) : graph`

Представление графов в Coq: граф H

```
Definition H : graph :=
  let g1 := rename_in_order
    (mk_cmn_edge K3 K3 1 3 1 3) in
  let g2 := rename_in_order
    (mk_cmn_edge g1 K3 1 (snd (gr_rng g1)) 1 3) in
  let g3 := rename_in_order
    (mk_cmn_edge g2 K3 1 (snd (gr_rng g2)) 1 3) in
  let g4 := rename_in_order
    (mk_cmn_edge g3 K3 1 (snd (gr_rng g3)) 1 3) in
  rename_in_order
    (add_edge (2, snd (gr_rng g4)) g4).
```

Представление графов в Coq: граф J

```
Definition J: graph :=
  let HH := mk_cmn_edge H H 2 3 6 7 in
  let HH_H := mk_cmn_edge HH H 7 2 6 7 in
  let HHH := rename_node 14 12 HH_H in
  let HHH_H := mk_cmn_edge HHH H 6 7 6 7 in
  let HHHH := rename_node 19 17 HHH_H in
  let HHHH_H := mk_cmn_edge HHHH H 5 6 6 7 in
  let HHHHH := rename_node 24 22 HHHH_H in
  let HHHHH_H := mk_cmn_edge HHHHH H 4 5 6 7 in
  let HHHHHH := rename_node 29 27 HHHHH_H in
  let HHHHHH_H := mk_cmn_edge HHHHHH H 3 4 6 7 in
  let HHHHHHH :=  rename_node 34 32 HHHHHH_H in
  rename_in_order (rename_node 37 9 HHHHHHH).
```

Представление графов в Coq: графы K и L

```
Definition K: graph :=
  let JJ := mk_art J J 1 1 in
  let JJ := add_edges [
            (9, 9+31); (12, 12+31); (16, 16+31);
            (20, 20+31); (24, 24+31); (28, 28+31)
            ] JJ in
  rename_in_order JJ.

Definition L: graph :=
  let KK := mk_art K K A A in
  let KK := add_edge (B, B+snd(gr_rng K)) KK in
  rename_in_order KK.
```

Доказательства корректности операций над графами

```
Definition undirected (g: graph) :=
   forall i j, S.In j (adj g i) ->
                         S.In i (adj g j).

Definition no_selfloop (g: graph) :=
    forall i, ~ S.In i (adj g i).

Definition graph_ok (g : graph) :=
  undirected g /\ no_selfloop g.
```

Доказательства корректности операций над графами: H_ok

```
Lemma H_ok : graph_ok H.
Proof.
split.
- unfold undirected. intros. remember H as H'.
  clear HeqH'. apply edge_corr_1 in H.
  gr_destr H. gr_destr H'.  reflexivity.
- unfold no_selfloop. repeat intro. remember H as H'.
  clear HeqH'. apply edge_corr_1 in H. gr_destr H; gr_destr H';
  discriminate.
Qed.
```

```
1 subgoal
_____________________________________(1/1)
graph_ok H
```

Доказательства корректности операций над графами: H_ok

```
Lemma H_ok : graph_ok H.
Proof.
split.
- unfold undirected. intros. remember H as H'.
  clear HeqH'. apply edge_corr_1 in H.
  gr_destr H. gr_destr H'.  reflexivity.
- unfold no_selfloop. repeat intro. remember H as H'.
  clear HeqH'. apply edge_corr_1 in H. gr_destr H; gr_destr H';
  discriminate.
Qed.
```

```
2 subgoals
_____________________________________(1/2)
undirected H
_____________________________________(2/2)
no_selfloop H
```

Доказательства корректности операций над графами: H_ok

```
Lemma H_ok : graph_ok H.
Proof.
split.
- unfold undirected. intros. remember H as H'.
  clear HeqH'. apply edge_corr_1 in H.
  gr_destr H. gr_destr H'.  reflexivity.
- unfold no_selfloop. repeat intro. remember H as H'.
  clear HeqH'. apply edge_corr_1 in H. gr_destr H; gr_destr H';
  discriminate.
Qed.
```

```
1 subgoal
_____________________________________(1/1)
undirected H
```

Доказательства корректности операций над графами: H_ok

```
Lemma H_ok : graph_ok H.
Proof.
split.
- unfold undirected. intros. remember H as H'.
  clear HeqH'. apply edge_corr_1 in H.
  gr_destr H. gr_destr H'.  reflexivity.
- unfold no_selfloop. repeat intro. remember H as H'.
  clear HeqH'. apply edge_corr_1 in H. gr_destr H; gr_destr H';
  discriminate.
Qed.
```

```
1 subgoal
______________________________________(1/1)
forall (i : node) (j : S.elt), S.In j (adj H i) -> S.In i (adj H j)
```

Доказательства корректности операций над графами: H_ok

```
Lemma H_ok : graph_ok H.
Proof.
split.
- unfold undirected. intros. remember H as H'.
  clear HeqH'. apply edge_corr_1 in H.
  gr_destr H. gr_destr H'.  reflexivity.
- unfold no_selfloop. repeat intro. remember H as H'.
  clear HeqH'. apply edge_corr_1 in H. gr_destr H; gr_destr H';
  discriminate.
Qed.
```

```
1 subgoal
i : node
j : S.elt
H : S.In j (adj H i)
_____________________________________(1/1)
S.In i (adj myGraphs.H j)
```

Доказательства корректности операций над графами: H_ok

```
Lemma H_ok : graph_ok H.
Proof.
split.
- unfold undirected. intros. remember H as H'.
  clear HeqH'. apply edge_corr_1 in H.
  gr_destr H. gr_destr H'.  reflexivity.
- unfold no_selfloop. repeat intro. remember H as H'.
  clear HeqH'. apply edge_corr_1 in H. gr_destr H; gr_destr H';
  discriminate.
Qed.
```

```
1 subgoal
i : node
j : S.elt
H, H' : S.In j (adj H i)
HeqH' : H' = H
______________________________________(1/1)
S.In i (adj myGraphs.H j)
```

Доказательства корректности операций над графами: H_ok

```
Lemma H_ok : graph_ok H.
Proof.
split.
- unfold undirected. intros. remember H as H'.
  clear HeqH'. apply edge_corr_1 in H.
  gr_destr H. gr_destr H'.  reflexivity.
- unfold no_selfloop. repeat intro. remember H as H'.
  clear HeqH'. apply edge_corr_1 in H. gr_destr H; gr_destr H';
  discriminate.
Qed.
```

```
1 subgoal
i : node
j : S.elt
H, H' : S.In j (adj H i)
______________________________________(1/1)
S.In i (adj myGraphs.H j)
```

Доказательства корректности операций над графами: H_ok

```
Lemma H_ok : graph_ok H.
Proof.
split.
- unfold undirected. intros. remember H as H'.
  clear HeqH'. apply edge_corr_1 in H.
  gr_destr H. gr_destr H'.  reflexivity.
- unfold no_selfloop. repeat intro. remember H as H'.
  clear HeqH'. apply edge_corr_1 in H. gr_destr H; gr_destr H';
  discriminate.
Qed.
```

```
1 subgoal
i : node
j : S.elt
H : S.In i (nodes H)
H' : S.In j (adj myGraphs.H i)
______________________________________(1/1)
S.In i (adj myGraphs.H j)
```

Доказательства корректности операций над графами: H_ok

```
Lemma H_ok : graph_ok H.
Proof.
split.
- unfold undirected. intros. remember H as H'.
  clear HeqH'. apply edge_corr_1 in H.
  gr_destr H. gr_destr H'.  reflexivity.
- unfold no_selfloop. repeat intro. remember H as H'.
  clear HeqH'. apply edge_corr_1 in H. gr_destr H; gr_destr H';
  discriminate.
Qed.
```

```
7 subgoals
j : S.elt
H0 : 4 = 4
H' : S.In j (adj H 4)
______________________________________(1/7)
S.In 4 (adj H j)
______________________________________(2/7)
S.In 2 (adj H j)
______________________________________(3/7)
S.In 6 (adj H j)
______________________________________(4/7)
S.In 1 (adj H j)
______________________________________(5/7)
S.In 5 (adj H j)
______________________________________(6/7)
S.In 3 (adj H j)
```

Доказательства корректности операций над графами: H_ok

```
Lemma H_ok : graph_ok H.
Proof.
split.
- unfold undirected. intros. remember H as H'.
  clear HeqH'. apply edge_corr_1 in H.
  gr_destr H. gr_destr H'.  reflexivity.
- unfold no_selfloop. repeat intro. remember H as H'.
  clear HeqH'. apply edge_corr_1 in H. gr_destr H; gr_destr H';
  discriminate.
Qed.
```

```
9 subgoals
H0 : 4 = 4
__________________________________________(1/9)
S.In 4 (adj H 1)
__________________________________________(2/9)
S.In 4 (adj H 5)
__________________________________________(3/9)
S.In 4 (adj H 3)
__________________________________________(4/9)
S.In 2 (adj H j)
__________________________________________(5/9)
S.In 6 (adj H j)
__________________________________________(6/9)
S.In 1 (adj H j)
__________________________________________(7/9)
S.In 5 (adj H j)
```

Доказательства корректности операций над графами: H_ok

```
Lemma H_ok : graph_ok H.
Proof.
split.
- unfold undirected. intros. remember H as H'.
  clear HeqH'. apply edge_corr_1 in H.
  gr_destr H. gr_destr H'.  reflexivity.
- unfold no_selfloop. repeat intro. remember H as H'.
  clear HeqH'. apply edge_corr_1 in H. gr_destr H; gr_destr H';
  discriminate.
Qed.
```

```
8 subgoals
H0 : 4 = 4
______________________________________________(1/8)
S.In 4 (adj H 5)
______________________________________________(2/8)
S.In 4 (adj H 3)
______________________________________________(3/8)
S.In 2 (adj H j)
______________________________________________(4/8)
S.In 6 (adj H j)
______________________________________________(5/8)
S.In 1 (adj H j)
______________________________________________(6/8)
S.In 5 (adj H j)
______________________________________________(7/8)
S.In 3 (adj H j)
```

Доказательства корректности операций над графами: тактика gr_destr

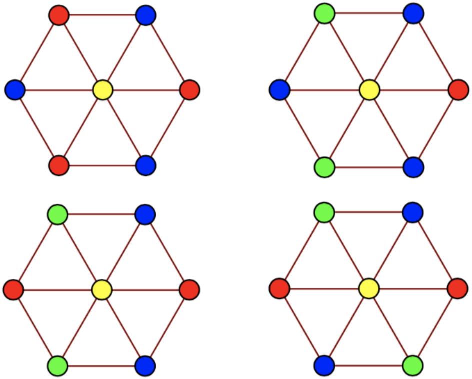
```
Ltac gr_destr h :=
  apply S.elements_1 in h; compute in h;
  repeat rewrite InA_cons in h;
  rewrite InA_nil in h;
  repeat destruct h as [? | h];
  try inversion h; subst.
```

Теорема о корректности добавления ребра

```
Lemma add_edge_corr' : forall g x y a b,
  edge (add_edge (a, b) g) x y <-> edge g x y \/
  (x = a /\ y = b) \/ (x = b /\ y = a).

Lemma add_edge_corr : forall g a b, graph_ok g ->
  a <> b -> graph_ok (add_edge (a, b) g).
```

Типы возможных правильных раскрасок графа T в не более чем 4 цвета



Типы возможных правильных раскрасок графа T в не более чем 4 цвета

```
Definition Coloring := positive -> positive.

Inductive is_color : positive -> Prop :=
| c1: is_color 1
| c2: is_color 2
| c3: is_color 3
| c4: is_color 4.

Definition is_coloring (c : Coloring) :=
    forall x : positive, is_color (c x).

Definition is_good_coloring (c : Coloring) (g : graph) :=
  is_coloring c /\ forall x y : positive,
    S.In y (adj g x) -> c x <> c y.
```

Типы возможных правильных раскрасок графа T в не более чем 4 цвета

```
Lemma coloring_H:
  forall c: Coloring, is_good_coloring c H ->
  type1_H c \/ type2_H c \/ type3_H c \/ type4_H c.
Proof.
  intros. unfold is_good_coloring in H.
  unfold is_coloring in H. destruct H.
  color_next H 1;
    color_next H 2; try find_contr H0 c;
      color_next H 3; try find_contr H0 c;
        color_next H 4; try find_contr H0 c;
          color_next H 5; try find_contr H0 c;
            color_next H 6; try find_contr H0 c;
              color_next H 7; try find_contr H0 c;
                find_type H3 H5 H7 H9 H11 H13 H15 c.
Qed.
```

Результаты и выводы

- Разработаны эффективные методы конструирования графов, обладающих высокой степенью симметрии
- Эти методы применены к графам H, J, K и L
- Разработаны методы работы с раскрасками графов
- Верифицировано доказательство того, что существует ровно 4 существенно различные раскраски графа H в не более чем 4 цвета

Планы будущей работы

- Формализация алгоритма раскраски графа из статьи де Грея
- Верификация указанного алгоритма
- Формализация конструкций графов из главы 4, основывающихся на реализации на плоскости

Спасибо за внимание!