TARTU ÜLIKOOL
Arvutiteaduse instituut
Informaatika õppekava

Raul Redpap

Kahni algoritmi tõestamine Coq raamistikus

Bakalaureusetöö (9 EAP)

Juhendaja: Kalmer Apinis

Tartu 2019

Kahni algoritmi tõestamine Coq raamistikus

Lühikokkuvõte:

Coq on funktsionaalne programmeerimiskeel, mille abil saab kirjutada teoreemide tõestusi. Töö eesmärk on uurida ja anda ülevaade Coqist ning tõestada Coqis omadus, et Kahni algoritm lõpetab töö iga sisendi puhul ja et tagastatud järjestus on tõepoolest topoloogiline järjestus. Peamiselt keskendutame formaalse tõestuse loomisele ja tõestustes kasutatavate taktikate kirjeldamisele, kuid antakse ka lühiülevaade Kahni algoritmist ja topoloogilisest järjestusest.

Võtmesõnad:

Algoritmid ja andmestruktuurid, Kahni algoritm, Coq, formaalne tõestus

CERCS: P175 Informaatika, süsteemiteooria

Proof of Kahn’s algorithm in Coq

Abstract:

Coq is a functional programming language that can be used to write proofs of theorems. The purpose of the work is to investigate and provide an overview of Coq and to prove the property that Kahn’s algorithm terminates for every input and that the returned sequence is indeed a topological sorting. We mainly focus on the creation of formal proof and the description of tactics used in proofs, but also provides a brief overview of Kahn's algorithm and topological sorting.

Keywords:

Algorithms and data structures, Kahn’s algorithm, Coq, formal proof

CERCS: P175 Informatics, systems theory

Sisukord

1. Sissejuhatus 4

2. Coq raamistik 5

2.1 Taktikad, mis lahendavad lihtsamaid eesmärke 6

2.2 Taktikad, mis muudavad eesmärki 7

2.3 Taktikad, mis jagavad eesmärgi või eelduse mitmeks erinevaks osaks 8

3. Kahni algoritm 11

3.1 Graafidega seotud mõisted 11

3.2 Algoritmi kirjeldus ja topoloogiline järjestus 11

4. Algoritmi tõestusest 13

5. Kokkuvõte 17

Viidatud kirjandus 18

Lisad 19

I. Kahni algoritmi tõestus Coqis 19

II. Litsents 20

1. Sissejuhatus

Tartu Ülikooli informaatika eriala üks kohustuslikke mooduleid on matemaatika alusmoodul. Selle mooduli kursuste raames puutub tudeng kokku teoreemide ning nende tõestustega. Seetõttu võiks informaatikatudengil olla ettekujutus, kuidas neid tõestusi on võimalik kirjutada ka programmeerimiskeeltes. Tõestamiseks kasutatavaid keeli on loodud mitmeid, kuid selles töös keskendutakse vaid Coqile. See keel sai valitud selle järgi, et Coq on kõige populaarsem interaktiivne teoreemide tõestaja [1]. Coqi ja interaktiivselt teoreemide tõestamist toetavate programmide kohta puuduvad seni eestikeelsed materjalid. Neid keeli saab tarkvaraarenduses kasutada selleks, et tõestada loodud algoritmide korrektsust.

Töö eesmärgiks on Kahni algoritmi erinevate omaduste tõestuste kirjutamine eelmainitud keeles. Lisaks antakse töös eestikeelne ülevaade Coqist, tõestustes kasutatavatest taktikatest, Kahni algoritmist ja topoloogilisest järjestusest.

Töö koosneb kolmest suuremast osast. Esimeses osas antakse ülevaade raamistikust ja mõningatest taktikatest, mille abil tõestusi kirjutada. Järgmises osas kirjeldatakse Kahni algoritmi, topoloogilist järjestust ning nendega seotud mõisteid. Viimane osa jaguneb kaheks: esiteks formuleeritakse ja tõestatakse Coqis Kahni algoritmi omadused ning seejärel kirjeldatakse ning selgitatakse formaalse tõestuse tähtsamaid osasid.

2. Coq raamistik

Coq on süsteem, mille üks osa on funktsionaalne programmeerimiskeel. Selle keele südameks on CIC (*Calculus of Inductive Constructions*), mida tänu Curry-Howard isomorfismile saab matemaatiliste teoreemide, lausete ja lemmade tõestamiseks kasutada. Coqis on sisse ehitatud funktsioonide hulk väga väike, kuid tarkvara võimaldab kõiki andmetüüpe (sõned, naturaalarvud, tõeväärtused jne) ise defineerida. Siiski standardteekide abil on defineeritud näiteks tõeväärtused, naturaalarvud, listid ja paisktabelid. Teine tähtis Coq-i komponent on Ltac. See on skriptimiskeel, millega on tõestusi hõlpsam kirjutada. Coq ei tõesta automaatselt lauseid, kuid raamistik sisaldab taktikaid, milles mingi osa tõestusest saab Coq ise automaatselt ära teha. Lisaks saab Coqis tõestusi ja definitsioone teisendada sellistesse keeltesse nagu OCaml, Haskell ja Scheme. [2, 3]

Coqis on lausete ülesehitus kindla struktuuriga. Kõik laused algavad lausetüübiga, milleks võib olla näiteks definitsioon, funktsioon, teoreem, lemma jne. Lausetüübile järgneb lause nimi ning siis lause ise. Kõik koodi näited ja ka Kahni algoritmi omaduste tõestused on tehtud Coq versiooni 8.9.0.

Näide 1. Teoreemi tõestuse ülesehitus

```
Theorem plus_O_n : forall n : nat, 0 + n = n.
Proof.
  intros.|simpl. reflexivity.
Qed.
```

```
1 subgoal
n : nat
_____
0 + n = n
```

Näites 1 on vasakul kasutaja poolt kirjapandud lause. Selle lause tüübiks on *Theorem* ja lause nimeks on *plus_O_n*. Lause väidab, et iga naturaalarvu n korral kehtib võrdus 0 + n = n. Tõestus algab sõnaga *Proof* ja kui lause on tõestatud, siis lõppeb lühendiga *Qed*. *Proof* ja *Qed* vahele jäävad taktikad, mille abil lause tõestatakse. Taktikad teisendavad tõestuste seisundit, mis asub näites 1 paremal pool. Tõestuse seisund koosneb eesmärkidest ja eeldustest. Lause loetakse tõestatuks, kui kõik eesmärgid on tõestatud. Antud näites on välja toodud tõestuse seisund tõestuses oleva kursori kohal - peale *intros*t, aga enne *simpl*it. Paremal pool on joone all eesmärk, mida kasutaja peab tõestama. Lausest sõltuvalt võib eesmärke olla ka mitu. Joone kohal näeb kasutaja, mitu eesmärki on vaja tõestada ja samuti ka muutujat koos andmetüübiga.

Järgnevalt tuuakse välja mõned näiteid taktikatest.

2.1 Taktikad, mis lahendavad lihtsamaid eesmärke

Reflexivity: selle taktika abil tõestatakse väiteid kujul x = x. Vajadusel lihtsustab see taktika avaldist automaatselt, kuigi Coqis on avaldiste lihtsustamiseks olemas ka taktika *simpl*. Näidetes 2 ja 3 on näha, et eesmärk sai tõestatud, ilma et avaldist oleks eelnevalt lihtsustatud. [4]

Näide 2. Eesmärk enne *reflexivity* kasutamist

```
Theorem plus_O_n : forall n : nat, 0 + n = n.
Proof.
  intros.|reflexivity.
Qed.
```

```
1 subgoal
n : nat
_____________
0 + n = n
```

Näide 3. Eesmärk tõestatud pärast *reflexivity* kasutamist

```
Theorem plus_O_n : forall n : nat, 0 + n = n.
Proof.
  intros. reflexivity.|
Qed.
```

```
No more subgoals.
```

Discriminate: seda taktikat kasutatakse selleks, et tõestada lauseid, kus eelduses tekib vastuolu. Vastuolu võib eeldusel olla mõne eelnevalt defineeritud konstruktoriga või mingid kaks eeldust võivad omavahel olla vastuolus. Vastuolu esinemise korral jäetakse tõestuse vastuoluline osa läbi vaatamata ja eemaldatakse eesmärkide hulgast. Näidetes 4 ja 5 on näha, et eeldus H on väär ja seega lause on tõestatud. Vastuolu tuleneb sellest, et naturaalarvud on defineeritud nii, et S n on nullist erinev naturaalarv (vt näide 6). [4]

Näide 4. Tõestuse olukord enne *discriminate* kasutamist

```
Theorem discriminate_ex1 : forall (n : nat),
  S n = 0 -> 2 + 2 = 5.
Proof.
  intros.|discriminate.
Qed.
```

```
1 subgoal
n : nat
H : S n = 0
_____________
2 + 2 = 5
```

Näide 5. Väite eeldus on väär ja *discriminate*i abil on lause tõestatud

```
Theorem discriminate_ex1 : forall (n : nat),
  S n = 0 -> 2 + 2 = 5.
Proof.
  intros. discriminate.|
Qed.
```

```
No more subgoals.
```

Näide 6. Naturaalarvu definitsioon

```
Inductive nat : Type :=
 | O
 | S (n : nat).
```

2.2 Taktikad, mis muudavad eesmärki

Intro ja ***intros***: neid taktikaid kasutatakse, et välja sorteerida väites olevad muutujad ja eeldused (vt näide 7 ja näide 4).

Näide 7. Seis enne *introse* kasutamist

```
Theorem discriminate_ex1 : forall (n : nat),
 S n = 0 -> 2 + 2 = 5.
Proof.
 intros. discriminate.
Qed.
```

```
1 subgoal
_____________________________________
forall n : nat, S n = 0 -> 2 + 2 = 5
```

Coq leiab muutujad ja eeldused väitest ise üles, kuid tegelikult on võimalik ka ise määrata uued muutujanimed. Muutujanimesid on soovituslik ette anda, sest raamistiku versiooni uuenedes võivad tekkida tõrked. *Intro* ja *intros*e vahe seisneb selles, et *intro* lõpetab töö esimese muutuja või eelduse esinemisel väitest, kuid *intros* otsib muutujaid ja väiteid kuni kõik on leitud. *Intros*ele muutujanimesid ette andes leiab see taktika nii mitu muutujat ja eeldust kui mitu muutujanime on kaasa antud. [4]

Apply: kui väite tõestamisel taandub väide samale kujule nagu eeldus või mõni varasemalt tõestatud teoreem, siis selle taktika abil saame eeldust või teoreemi rakendades enda tõestuse tõestatud (vt näide 8 ja näide 9). [4]

Näide 8. Enne *apply* rakendamist on eesmärk samal kujul nagu eeldus

```
Theorem minus_diag : forall n,
 minus n n = 0.
Proof.
 intros n. induction n as [| n' IHn'].
  simpl. reflexivity.
  simpl. apply IHn'.  Qed.
```

```
1 subgoal
n' : nat
IHn' : n' - n' = 0
_____________________
n' - n' = 0
```

Näide 9. Teoreem tõestatud peale *apply* rakendamist

```
Theorem minus_diag : forall n,
  minus n n = 0.
Proof.
  intros n. induction n as [| n' IHn'].
    simpl. reflexivity.
    simpl. apply IHn'. Qed.
```

```
No more subgoals.
```

Lisaks vastab apply lausearvutuses *modus ponens* reeglile, mis tähendab, et kui A-st järeldub B ja on vaja tõestada B, siis piisab kui tõestame A. [4]

Näide 10. Eesmärk enne *apply eq2* rakendamist

```
Theorem apply_test : forall (n m o p : nat),
    n = m  ->
    (forall (q r : nat), q = r -> [q;o] = [r;p]) ->
    [n;o] = [m;p].
Proof.
  intros n m o p eq1 eq2.
  apply eq2. apply eq1.  Qed.
```

```
1 subgoal
n, m, o, p : nat
eq1 : n = m
eq2 : forall q r : nat, q = r -> [q; o] = [r; p]
______________________________________
[n; o] = [m; p]
```

Näide 11. Eesmärk peale *apply eq2* rakendamist

```
Theorem apply_test : forall (n m o p : nat),
    n = m  ->
    (forall (q r : nat), q = r -> [q;o] = [r;p]) ->
    [n;o] = [m;p].
Proof.
  intros n m o p eq1 eq2.
  apply eq2.apply eq1.  Qed.
```

```
1 subgoal
n, m, o, p : nat
eq1 : n = m
eq2 : forall q r : nat, q = r -> [q; o] = [r; p]
______________________________________
n = m
```

Näites 10 on näha, et väide B on [n;o] = [m;p]. Näites 11 näeme, et piisab kui tõestame väite A, milleks on n = m.

2.3 Taktikad, mis jagavad eesmärgi või eelduse mitmeks erinevaks osaks

Destruct: induktiivsete andmetüüpide korral kasutatakse seda taktikat juhtumite eraldamiseks. Juhtumite arv sõltub eelnevalt defineeritud andmetüübist. Tõeväärtus on defineeritud järgnevalt :

```
Inductive bool : Set :=
  | true : bool
  | false : bool.
```

Näites 12 ja näites 13 on näha, et *destruct*i kasutamisel tekib kaks juhtumit (sest tõeväärtusel saab definitsiooni järgi olla ainult kaks väärtust), kus esimesel korral saab muutuja b väärtuseks tõene ja teisel korral väär. [4]

Näide 12. Enne *destruct*i on üks eesmärk

```
Theorem negb_involutive : forall b : bool,
  negb (negb b) = b.
Proof.
  intros.| destruct b.
```

```
1 subgoal
b : bool
______________________
negb (negb b) = b
```

Näide 13. *Destruct*i kasutamise järel on kaks eesmärki

```
Theorem negb_involutive : forall b : bool,
  negb (negb b) = b.
Proof.
  intros. destruct b.|
```

```
2 subgoals
______________________
negb (negb true) = true
______________________
negb (negb false) = false
```

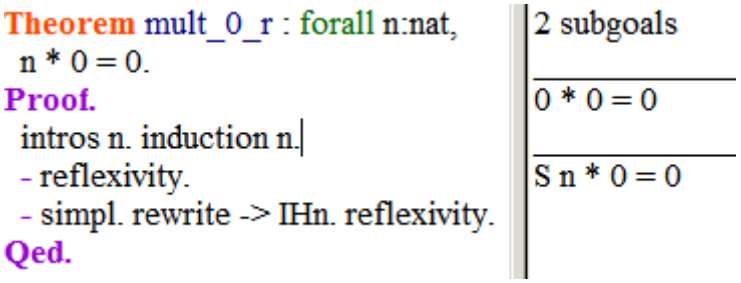
Induction: see taktika põhineb struktuurse induktsiooni meetodil. Naturaalarvude puhul tuleb tõestada baas ja samm. See taktika on väga sarnane *destruct*iga, kuid oluline erinevus seisneb selles, et sammu tõestamisel saab kasutada induktsiooni hüpoteesi. Näites 14 on vaja tõestada, et iga naturaalarvu n korral kehtib, et n * 0 = 0. Näites 15 on tehtud induktsioon üle naturaalarvu n ning on vaja näidata, et väide kehtib kui n = 0 ja et sellest järeldub, et kehtib ka n = S n' korral (vt näide 6). Näites 16 on näha, et peale esimese juhtumi tõestust on eelduste hulka lisatud induktsiooni hüpotees, mis ütleb, et väide kehtib naturaalarvu n korral. Rekursiivseid funktsioone ja definitsioone on mõistlikum tõestada kasutades *induction*it kui *destruct*i. [4]

Näide 14. Olukord enne *induction*i kasutamist

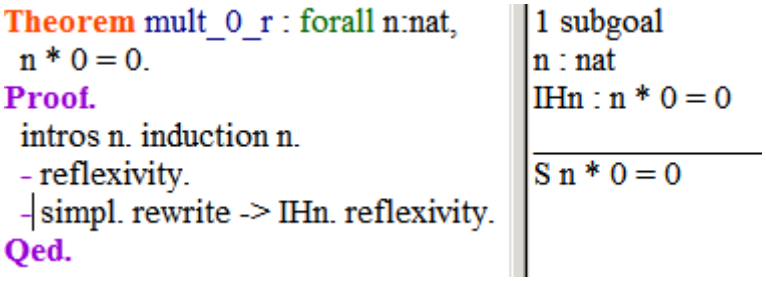
```
Theorem mult_0_r : forall n:nat,
  n * 0 = 0.
Proof.
  intros n.| induction n.
  - reflexivity.
  - simpl. rewrite -> IHn. reflexivity.
Qed.
```

```
1 subgoal
n : nat
______________________
n * 0 = 0
```

Näide 15. *Induction*i kasutamise järel tekib kaks eesmärki



Näide 16. Peale esimese eesmärgi tõestust lisatakse eelduste hulka induktsiooni hüpotees



3. Kahni algoritm

3.1 Graafidega seotud mõisted

Orienteeritud graafiks nimetatakse graafi G = (V, E), kus V on mittetühi tippude hulk ning E on kaarte hulk, mis koosneb hulga V järjestatud paaridest. **Ahel** on selline tippude järjend v_0, v_1, …, v_k, kus iga kaks järjestikku tippu on servaga ühendatud. **Tsükliks** nimetatakse ahelat, mis algab ja lõppeb ühe ja sama tipuga. **Väljundastmeks** nimetatakse tipust väljuvate kaarte arvu. **Sisendastmeks** nimetatakse tippu sisenevate kaarte arvu. **Väljundiks** nimetatakse tippu, kuhu kaared ainult sisenevad. **Sisendiks** nimetatakse tippu, kust kaared ainult väljuvad. Graafi nimetatakse **sidusaks**, kui iga kahe tipu korral leidub neid tippe ühendav ahel. Kui graaf ei ole sidus, siis jaguneb ta eraldiseisvateks osadeks, millest igaüks omaette on sidus graaf. Neid osi nimetatakse **sidusateks komponentideks**. Orienteeritud graafi tippude **topoloogiliseks järjestuseks** nimetatakse järjestust, kus komponentideks on parajasti graafi kõik tipud, iga tipp üks kord, ja graafi iga kaare korral selle algustipp paikneb lõpptipust eespool. Vaid tsükliteta graafi tippe saab järjestada topoloogiliselt. Kahni algoritmi kasutatakse orienteeritud ehk suunatud tsükliteta graafi tippude topoloogilise järjestuse leidmiseks. [5-7]

3.2 Algoritmi kirjeldus ja topoloogiline järjestus

Sisend: Graaf (E,V)

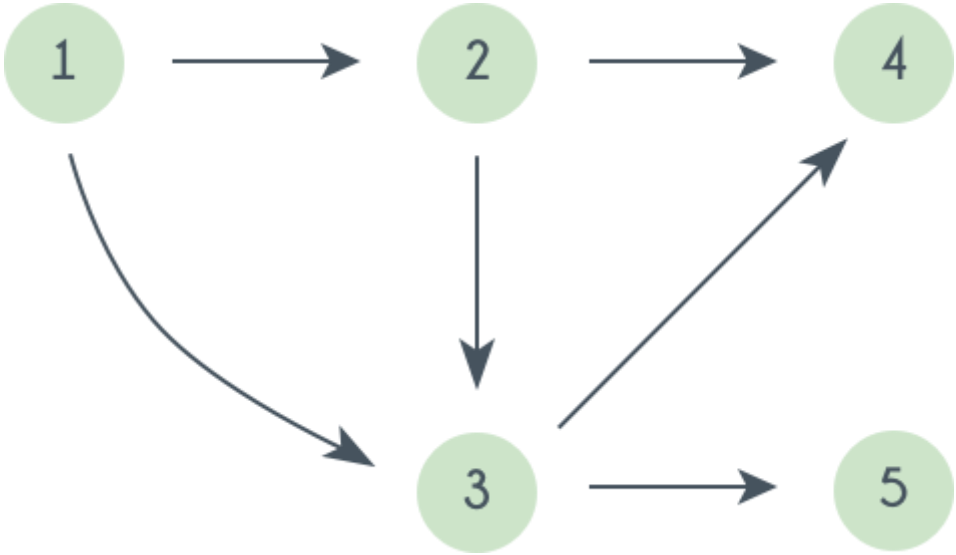
Väljund: Järjestus L

Kahni algoritm:

1) Leitakse iga tipu sisendaste
2) Luuakse järjend Q, mis sisaldab graafi tippe, mille sisendaste on null
3) Tsükkel
    - Töö lõpetatakse, kui järjend Q on tühi
    - Kui järjend Q pole tühi:
        a. Valitakse järjendist suvaline tipp *v*
        b. Eemaldame graafist (E,V) tipu v ja kõik servad (v,u)
        c. Tipp v lisatakse topoloogilise järjestuse L lõppu
        d. Lõpptipud, mille sisendaste graafis (E,V) muutub nulliks, lisatakse järjendisse Q

4) Kui E on tühi, siis väljastatakse L, muidu veateade. [7]

Joonis 1. Orienteeritud tsükliteta graaf



Topoloogilise järjestuse leidmiseks ei pea graaf olema sidus, vaid peavad leiduma sidusad komponendid. Topoloogiline järjestus pole ilmtingimata üheselt määratud. Orienteeritud tsükliteta graafil võib olla mitu topoloogilist järjestust. Joonisel 1 on näide orienteeritud tsükliteta graafist, millel on kaks erinevat topoloogilist järjestust:

1) 1 2 3 4 5
2) 1 2 3 5 4. [6, 8]

4. Algoritmi tõestusest

Coqis algoritmi kirjeldamiseks on vaja kasutada erinevaid loendeid, mis on järgnevalt loetletud:

- SV – algsete tippude hulk graafis
- SE – algsete servade hulk graafis
- E – jooksev servade hulk
- Q – töötlemata sisendastmega null tippude hulk
- P – paari (E, Q)
- L – tulemusjärjend

Koodis on Kahni algoritm koos abifunktsioonidega teostatud järgmiselt:

```
Fixpoint remove_edges_from (E: edges) (v: node) (X: nodes) : edges*nodes :=
  match E with
  | nil => (nil,X)
  | (x,y)::E' =>
    if equalN v x then
     remove_edges_from E' v (y::X)
    else
     let '(E'', Q') := remove_edges_from E' v X in
     ((x,y) :: E'', Q')
  end.
 Fixpoint add_to_Q (E: edges) (X:nodes) (Q:nodes) : nodes :=
  match X with
  | nil => Q
  | y::X' =>
    if noIncoming E y then
     add_to_Q E X' (y::Q)
    else
     add_to_Q E X' Q
  end.
 Definition remove_E_add_Q (E: edges) (v: node) (Q: nodes) : edges*nodes :=
  let '(E', X) := remove_edges_from E v nil in
  (E', add_to_Q E' X Q).
```

```
Function kahn_loop (P:edges * nodes)  (L: nodes) {measure length_pair P}: option nodes :=
  match P with
  | (E, nil) =>
   match E with
     | nil         => Some L
     | cons _ _ => None
   end
  | (E, v :: Q') =>
     kahn_loop (remove_E_add_Q E v Q')  (v::L)
  end.
Definition kahn (V:nodes) (E:edges): option nodes :=
  kahn_loop (E, (filter (noIncoming E) V)) [].
```

Definitsioon *kahn* kutsub välja *kahn_loop* funktsiooni. Selleks sorteeritakse graafist välja kõik sisendastmega null tipud. Funktsioonis *kahn_loop* vaadatakse kõigepealt, kas Q-s on veel töötlemata tippe. Kui pole töötlemata tippe ja E-s ei leidu enam servi, siis tagastatakse topoloogiline järjestus. Kui E-s leidub servi, siis järelikult leidub graafis tsükkel ja tagastatakse *None*. Kui Q-s on veel töötlemata tippe, rakendatakse abifunktsiooni *remove_E_add_Q*, mis omakorda kasutab kahte abifunktsiooni *remove_edges_from* ja *add_to_Q*. Funktsioon *remove_edges_from* eemaldab kõik servad, mis väljuvad tipust v, ning lisab kõik tipud, kuhu siseneb serv tipust v, X-i. Funktsiooni *noIncoming* abil kontrollitakse, kas tippu x leidub sissetulevaid servi. Nii *remove_edges_from* kui ka *noIncoming* kasutavad ka definitsiooni *equalN*, millega vaadatakse, kas mingid kaks tippu on võrdsed. *Add_to_Q* lisab kõik X-is olevad tipud, millesse pole sisenevaid servi, Q-sse. Abifunktsioon *remove_E_add_Q* eemaldab E-st kõik tipust v väljuvad servad ja lisab kõik sisendastmega null tekkivad tipud Q-sse. Kuna elemente lisatakse listi algusse konstantse ajaga, on arvutuse efektiivsuse huvides L-is topoloogiline järjestus tagurpidises järjestuses. Paari P on vaja selleks, et lihtsam oleks tõestada algoritmi termineeruvust.

Topoloogilise järjestuse jaoks on vaja defineerida induktiivne andmetüüp *First_Then*:

```
Inductive First_Then {X:Type}: X -> X -> list X -> Prop :=
  | FT_delay x y z xs : First_Then x y xs ->  First_Then x y (z::xs)
  | FT_start x y xs : In y xs ->  First_Then x y (x::xs).
```

FT_start ütleb, et kui tipp y on xs-is, siis tipu x lisamisel xs-i asub tipp x tipust y eespool. *FT_delay* ütleb, et kui tipp x on tipust y eespool xs-is, siis tipp x on tipust y eepool ka peale tipu z lisamist xs-i.

Topoloogiline järjestus Coqis:

```
Definition TopoOrder (E:edges) (xs:nodes) :=

  forall (x y: node), In (x,y) E -> First_Then y x xs.
```

Topoloogilise järjestuse definitsiooni kohaselt iga serva (x, y) korral kehtib, et tipp y asub tipust x eespool järjendis xs. Nii *kahn_loop* kui ka *TopoOrder* lubab ka korduvaid tippe tulemusjärjendis. Selline olukord võib tekkida, kui tipust x tippu y on mitu kaart ja tippu y teistest tippudest sisenevaid kaari pole. Selle vea avastasin liiga hilja ja kuna lõputöö maht oli ületatud, ei hakanud enam parandusi tegema. Paranduse jaoks oleks pidanud tõestustes kaasas kandma tingimust, et L-s esinevad elemendid ühekordselt.

Programmis on põhiteoreemiks:

```
Theorem main: forall (V: nodes) (E: edges) (l: nodes),
    forall n m, In (n,m) E -> In n V /\ In m V) -> kahn V E = Some l -> TopoOrder E l .
```

Põhiteoreemi kohaselt kui serv (n, m) on E-s ja tipud n ja m on V-s ning Kahni algoritm tagastab tulemusjärjendi, siis järeldub, et tulemusjärjend on topoloogiline järjestus.

Järgnevalt on toodud peamised tõestuse lemmad: *kahn_loop*, *startProp*, *main_loop*, *loopProp* ja *PropResultSound*. *Kahn_loop*is kontrollitakse algoritmi termineeruvust. Selleks on vaja tõestada, et kehtib `length_pair (remove_E_add_Q E v Q) < length_pair (E, v :: Q)`. See tähendab, et tsükli igal sammul paari (E, Q) pikkuste summa väheneb ja programm ei jää lõpmatult käima ning põhiteoreemi kohaselt tagastatakse korrektne järjestus. *StartProp*is tõestatakse, et kehtib `AProp SV SE (SE, (filter (noIncoming SE) SV)) []`. *AProp* on funktsiooni *kahn* väljakutse tsükli invariant, mis defineeritakse allpool. *Main_loop*is tõestatakse, et kehtib `kahn_loop (E, Q) L = Some L' -> AProp SV SE (E, Q) L -> TopoOrder SE L'` ehk kui tsükkel tagastab tulemusjärjendi L' ja kehtib tsükli invariant, siis järeldub, et L' on topoloogiline järjestus. *LoopProp*is tõestatakse, et kehtib `AProp SV SE (E, v :: Q') L -> AProp SV SE (remove_E_add_Q E v Q') (v::L)` ehk kui tsükli invariant kehtib algseisundis, siis kehtib ka tsükli järgneval sammul. *PropResultSound*is tõestatakse, et `AProp SV SE ([], []) L -> forall m n, In (m,n) SE -> First_Then n m L` ehk kui tsükli invariant kehtib ja kui E ja Q on saanud tühjaks ning serv (m, n) leidub esialgses graafis, siis järeldub, et L-s asub tipp n eespool tipust m.

Tsükli invariant *AProp* on defineeritud järgnevalt:

```
Definition AProp (SV:nodes) (SE:edges) (P:edges*nodes) (L: nodes) :=
    PropQl P /\ PropLl P L /\ PropQLr SV P L /\ PropL SE P L /\ PropLOrigin SE P L .
```

AProp koosneb viiest komponendist:

```
Definition PropQl (P:edges*nodes) := forall n, In n (snd P) -> noIncoming (fst P) n = true.,
Definition PropLl (P:edges*nodes) (L: nodes) := forall n, In n L -> noIncoming (fst P) n = true.,
Definition PropQLr (SV:nodes) (P:edges*nodes) (L: nodes) :=
    forall n, In n SV -> noIncoming (fst P) n = true -> In n (snd P) \/ In n L.,
Definition PropL (SE:edges) (P:edges*nodes) (L: nodes) :=
    forall m n, In (m,n) SE -> ~(In (m,n) (fst P)) -> In m L.,
Definition PropLOrigin (SE:edges) (P:edges*nodes) (L: nodes) :=
    forall m n, In (m,n) SE -> noIncoming (fst P) n = true -> ~ In n (snd P) -> First_Then n m L.
```

PropQl ütleb, et kui tipp n on Q-s, siis tippu n ei sisene servi E-s. *PropLl* ütleb, et kui tipp n on L-s, siis tippu n ei sisene servi E-s. *PropQLr* ütleb, et kui tipp n oli graafi tipp ja temasse ei sisene servi E-s, siis tipp n on kas Q-s või L-s. *PropL* ütleb, et kui serv (m, n) oli SE-s ja on E-st eemaldatud, siis tipp m asub L-s. *PropLOrigin* ütleb, et kui serv (m, n) oli SE-s, tippu n pole sisenevaid servi E-s ja tipp n ei ole Q-s, siis L-s asub tipp n tipust m eespool. Lisaks mainitutele lemmadele on töös defineeritud ja tõestatud ka mitmeid teisi abilemmasid, mis aitasid mainituid lemmasid tõestada. Täpsemalt saab näha tõestuse programmi failis „*Kahn.v*“ (vt lisa I).

5. Kokkuvõte

Bakalaureusetöö eesmärk oli tutvustada Coqi ning kirjutada Coqis Kahni algoritmi omaduste tõestus. Töö peamiseks saavutuseks oli eelmainitud programmeerimiskeeles Kahni algoritmi formaalse tõestuse kirjutamine. See eesmärk saavutati kahe mööndusega: esiteks väljastatakse topoloogiline järjestus tagurpidises järjekorras ja teiseks lubatakse korduvaid tippe topoloogilises järjestuses.

Veel üheks saavutuseks oli eestikeelse materjali loomine Coqi ja tõestuses kasutatavate taktikate kohta. Samuti anti ülevaade Kahni algoritmist ja topoloogilisest järjestusest. Töös kirjeldati ka tõestuse struktuuri, ülesehitust ning tõestuse tähtsamaid osasid.

Ühe võimaliku edasiarendusena on võimalik töö praktilises osas parandada puudused. Teoreetilises osas saab kirjeldada põhjalikumalt ja detailsemalt Coqis pakutavaid võimalusi ning saab luua sisukama ülevaate selles keeles kasutatavatest taktikatest.

Viidatud kirjandus

[1] J. Harrison, J. Urban and F. Wiedijk, "History of interactive theoremproving". https://www.cl.cam.ac.uk/~jrh13/papers/joerg.pdf (02.08.2019)

[2] COQ, "What is Coq?". https://coq.inria.fr/about-coq (28.04.2019)

[3] B. C. Pierce, A. Azevedo de Amorim, C. Casinghino, M. Gaboardi, M. Greenberg, C. Hriţcu, V. Sjöberg and B. Yoregy, "Logical Foundations," jaanuar 2019. https://softwarefoundations.cis.upenn.edu/lf-current/index.html (05.05.2019)

[4] "3110 Coq Tactics Cheatsheet". https://www.cs.cornell.edu/courses/cs3110/2018sp/a5/coq-tactics-cheatsheet.html (12.05.2019)

[5] R. Palm, Diskreetse matemaatika elemendid, Tartu: Tartu Ülikooli Kirjastus, 2009.

[6] A. Peder, Tartu Ülikooli õppeaine MTAT.03.133 „Algoritmid ja andmestruktuurid“ 13. loeng, 27 november 2018. https://moodle.ut.ee/pluginfile.php/193384/mod_resource/content/6/loeng13_nov2018.pdf (13.04.2019)

[7] J. Kiho, Algoritmid ja andmestruktuurid, Tartu: Tartu Ülikooli Kirjastus, 2003.

[8] V. Jaimini, "Topological Sort". https://www.hackerearth.com/practice/algorithms/graphs/topological-sort/tutorial/ (15.04.2019)

Lisad

I. Kahni algoritmi tõestus Coqis

Tõestus asub tööga kaasas olevas failis „*Kahn.v*“.

II. Litsents

Lihtlitsents lõputöö reprodutseerimiseks ja üldsusele kättesaadavaks tegemiseks

Mina, Raul Redpap,

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) minu loodud teose Kahni algoritmi tõestamine Coq raamistikus, mille juhendaja on Kalmer Apinis, reprodutseerimiseks eesmärgiga seda säilitada, sealhulgas lisada digitaalarhiivi DSpace kuni autoriõiguse kehtivuse lõppemiseni.
2. Annan Tartu Ülikoolile loa teha punktis 1 nimetatud teos üldsusele kättesaadavaks Tartu Ülikooli veebikeskkonna, sealhulgas digitaalarhiivi DSpace kaudu Creative Commonsi litsentsiga CC BY NC ND 3.0, mis lubab autorile viidates teost reprodutseerida, levitada ja üldsusele suunata ning keelab luua tuletatud teost ja kasutada teost ärieesmärgil, kuni autoriõiguse kehtivuse lõppemiseni.
3. Olen teadlik, et punktides 1 ja 2 nimetatud õigused jäävad alles ka autorile.
4. Kinnitan, et lihtlitsentsi andmisega ei riku ma teiste isikute intellektuaalomandi ega isikuandmete kaitse õigusaktidest tulenevaid õigusi.

Raul Redpap
12.08.2019