VŠB TECHNICKÁ UNIVERZITA OSTRAVA
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Formalizace konstrukcí na konečných automatech v programu COQ

Formalization of structures on finite state machines in the COQ program

Bc. Petr Kožušník

Vedoucí práce: doc. Ing. Zdeněk Sawa, Ph.D.

Ostrava, 2022

Obsah

Kapitola 1

Úvod

Coq je softwarový nástroj pro vytváření a ověřování formálních důkazů. Dokazovaná tvrzení a jednotlivé kroky důkazu se zapisují způsobem, který je v mnoha ohledech podobný zápisu programů v programovacím jazyce.

Cílem této práce je tento nástroj představit a ukázat způsob jeho použití. V tomto případě je použit pro formalizaci základních konstrukcí na konečných automatech. Pro pochopení způsobu formalizace, je nezbytná základní znalost jazyka Coq a jeho principů. Druhá kapitola je tak věnována právě úvodu do fungování tohoto dokazovacího systému.

Kapitola *Formalizce automatu* rozebírá samotný formální zápis konečného deterministického automatu v jazyce Coq. Jedná se o ukázku přepisu matematické definice do podoby dokazovacího jazyka. Kapitola dále obsahuje sekci s definicemi základních pojmů jako je abeceda nebo slovo.

V kapitole *Operace nad automaty*, je popsán zápis teorémů, které vyjadřují definice pro matematické struktury jako je sjednocení automatů nebo jejich průnik.

Tento text je vhodný jako podklad pro navazující práce se souvisejícím tématem.

Kapitola 2

Coq

Coq je imperativní programovací jazyk, který slouží zejména pro dokazování matematických tvrzení. Coq je rovněž označován jako takzvaný *proof assistant*, je tak zřejmé, že tento jazyk nutně obsahuje konstrukce pro zápis výroků. Pro zachování korektnosti matematického dokazování nelze jakoukoliv formuli libovolně prohlásit za platnou, ale je nutné ji tímto nástrojem dokázat.

Samotný Coq *de facto* pouze tvrzení vhodným způsobem transformuje do podoby, která jsou konsistentní s již dříve dokázanými větami a tím je prohlásí za platné. Z tohoto rovněž vyplývá, že je nutné postupovat od důkazů a definic základních pojmů, které jsou nezbytné pro konstrukce komplexnější.

Z těchto důvodů jsou nezbytné definice zahrnuty již ve standardní knihovně.

```
Definition not (A:Prop) := A -> False.
```

Listing 1: Definice záporu ze standardní knihovny

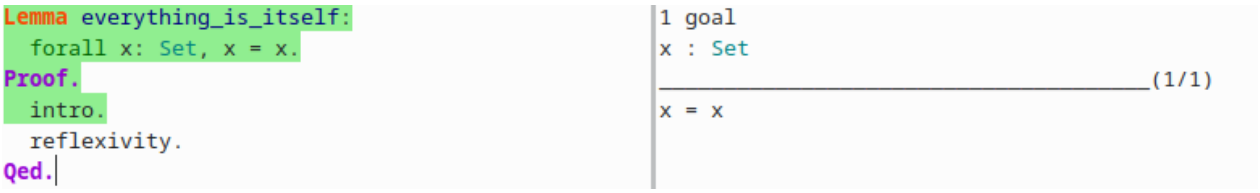
```
Inductive nat : Set :=
| O : nat
| S : nat -> nat.
```

Listing 2: Definice přirozených čísel ze standardní knihovny

Důkazy se provádějí pomocí takzvaných *taktik*. Taktika slouží pro transformaci výroku. Úkolem dokazovatele je výrok postupně transformovat do takové podoby, kdy je z kontextu zaručena jeho pravdivost. Dále je uvedeno několik příkladů základních *taktik*.

- *reflexivity*

  Reflexivitu je vhodné použít, pokud je cílem dokázat, že se něco rovná samo sobě.



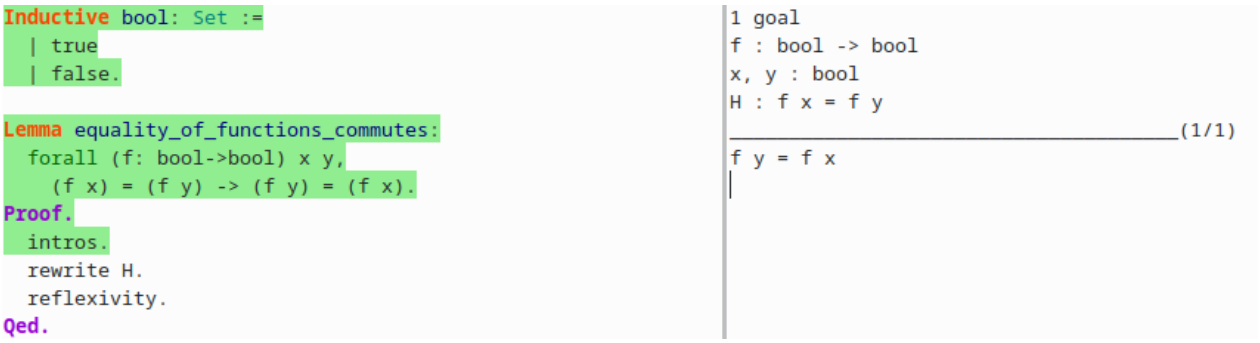
Obrázek 2.1: Použití taktiky *reflexivity*

V obrázku 2.1 dokazujeme, že jakýkoli člen x typu *Set* je roven sám sobě.

- *rewrite*

  Pokud jsou si dva výrazy rovny, můžeme přepsat jeden výraz na druhý pomocí *rewrite*.

  Mějme funkci f takovou, že pokud $(f\ x) = (f\ y)$, tak pak $(f\ y) = (f\ x)$. *Rewrite* lze použít k transformaci $(f\ x)$ v cíli na $(f\ y)$ (a následně dokončit důkaz pomocí *reflexivity*).



Obrázek 2.2: Použití taktiky *rewrite*

- *destruct*

  Pokud máme výraz nějakého typu, můžeme použít *destruct* k jeho rozložení podle konstruktoru. Vygenerují se dílčí cíle pro každý možný konstruktor, který mohl být pro konstrukci termínu použit. Poté je nutné dokázat každý dílčí cíl.

  Příklad ukazuje, že pokud negujeme *boolean* dvakrát, dostaneme stejný *boolean* zpět. Výraz rozložíme pomocí *destruct*, abychom jej dokázali pro jakoukoli možnou hodnotu (pravda nebo nepravda).

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

Definition not (b: bool) : bool :=
  match b with
    | true => false
    | false => true
  end.

Lemma not_not_x_equals_x:
  forall b, not (not b) = b.
Proof.
  intro.
  destruct b.
  - reflexivity.
  - reflexivity.
Qed.
```

```
2 goals
_____________________________________(1/2)
not (not true) = true
_____________________________________(2/2)
not (not false) = false
```

- *induction*

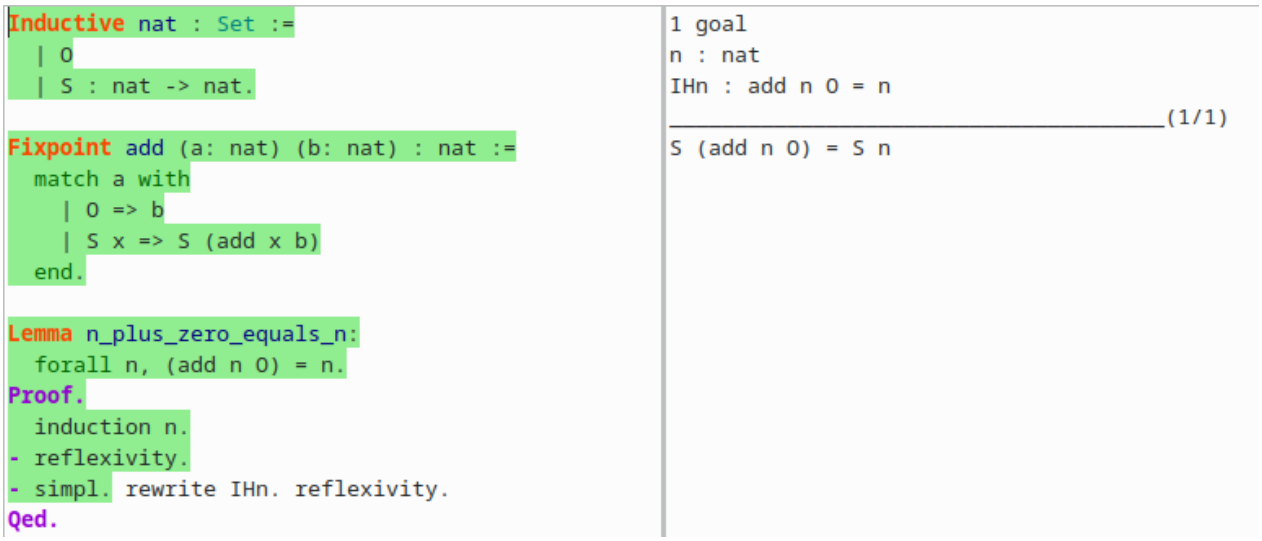
  Při použití *indukce*, Coq generuje dílčí cíle pro každý možný konstruktor výrazu podobně jako *destruct*, avšak navíc vytvoří indukční hypotézu.

  V příkladu 2.3 dokazujeme, že přičtením libovolného čísla k nule získáme stejné číslo. Provedeme indukci na n a dostaneme dva případy.

  Pokud n je O, pak víme, že *(add O O)* je O, toto je základní případ.

  Pro induktivní případ předpokládáme, že vlastnost platí pro n a musíme ji dokázat pro *(S n)* (nebo-li *n+1*).

  Abychom to dokázali, spustíme funkci *add* pro jeden krok pomocí *simpl*. Tím se S dostane mimo funkci *add* a nyní můžeme přepsat cíl pomocí induktivní hypotézy. Následně použijeme reflexivitu k dokončení důkazu.



Obrázek 2.3: Použití taktiky *induction*

Kapitola 3

Formalizace automatu

Tato kapitola ukazuje možný způsob formálního zápisu deterministického konečného automatu v jazyce Coq. Jelikož jsou složitější věty v tomto systému tvořeny z dílčích definic, je právě definice této struktury nezbytná. Současně je nutné definovat také pojmy související s touto doménou jako je abeceda nebo jazyk.

3.1 Deterministický konečný automat

Automat je abstraktní výpočetní stroj, který je využíván v oblasti teoretické informatiky a diskrétní matematiky. Zjednodušeně se jedná o množinu prvků (stavů), kdy je vždy právě jeden z nich označen jako *aktivní*, a zákonitostí pro změnu *aktivního* prvku.

Deterministický konečný automat je běžně definován jako následující uspořádaná pětice.

$(Q, \Sigma, q0, F, \delta)$

- Q - konečná neprázdná množina stavů
- Σ - konečná neprázdná množina vstupních symbolů - abeceda
- $q0$ - počáteční stav, $q0 \in Q$
- F - množina koncových stavů, $F \subseteq Q$
- δ - přechodová funkce, $Q \times \Sigma \mapsto Q$

Takovouto pětici v Coqu deklarujeme pomocí struktury.

```
(* Deterministic Finite Automaton *)
Structure DFA  (Sigma:Alphabet) : Type := createDFA {
                                  Q:      Set;
                                  delta:  Q -> Sigma -> Q;
                                  q0:     Q;
                                  F:      Q -> bool
}.
```

Abeceda je přijímána jako parametr při konstrukci automatu a následně použita pro deklaraci přechodové funkce.

Dále zde vidíme, že množina koncových stavů vzniká jako zobrazení množiny stavů na *boolean*, právě tímto způsobem lze vhodně vyjádřit podmnožina.

3.2 Definice základních pojmů

Pro práci v kontextu automatů je nutné definovat veškeré základní pojmy jako je jazyk nebo slovo. Coq implicitně tyto pojmy nezná, definujeme je následovně.

```
(* Alphabet is Set of characters *)
Definition Alphabet := Set.

(* Word is list of characters from Alphabet *)
Definition Word (Sigma : Alphabet) : Set := list Sigma.

(* Concatenation of words of alphabet *)
Definition ConcatWord (Sigma : Alphabet)(w1:Word Sigma)(w2: Word Sigma) : Word Sigma :=
  app w1 w2.

(* Language is  set of accepted words *)
Definition Language (Sigma : Alphabet):= Word Sigma -> Prop.

(* Intersection of languages *)
Definition LangIntersection (Sigma : Alphabet) (L1 L2:Language Sigma):=
  fun w: Word Sigma => L1 w /\ L2 w.

(* Unification of languages *)
Definition LangUnification (Sigma : Alphabet) (L1 L2: Language Sigma):=
  fun w: Word Sigma => L1 w \/ L2 w.
```

Kapitola 4

Operace nad automaty

Tato sekce ukazuje možný formální zápis množinových operací nad automaty.

4.1 Sjednocení automatů

Automat, který vznikl sjednocením automatů, přijímá taková slova, jež jsou přijímána alespoň jedním z nich. Tedy sjednocením automatu *A1* rozpoznávající jazyk *L1* a automatu*A2*, rozpoznávající jazyk *L2*, vznikne automat *A*, který rozpoznává slova jež patří do jazyka *L1* anebo *L2*.

Konstrukce takového automatu je následující. Počátečním stavem *A* je uspořádaná dvojice počátečních stavů *A1*, *A2*.

```
Let q0_a := q0 Sigma Automaton_a.
Let q0_b := q0 Sigma Automaton_b.
Let q0 := pair q0_a q0_b.
```

Následuje přechodová funkce.

```
Definition delta_union (q: Q)(a: Sigma) : Q :=
  let (q1, q2) := q in
  let q1':= delta_a q1 a in
  let q2':= delta_b q2 a in
    pair q1' q2'.
```

Koncový stav tvoří takové uspořádané dvojice, kdy jednotlivé prvky jsou koncovými stavy vstupních automatů, tedy:

```
Definition qend_union (q: Q) : bool :=
  let (q1, q2) := q in
  orb (qend_a q1)(qend_b q2).
```

Samotný sjednocený pak vytvoříme zavoláním koknstruktoru.

```
Definition Union_automata: DFA Sigma :=
  createDFA Sigma Q delta_union q0 qend_union.
```

Konstrukce automatu průniku by nebyla příliš odlišná. Ostatně jediným rozdílem je množina konečných stavů. Kdy oba prvky z uspořádané dvojice musí být koncovým stavem.

```
Definition Intersection_automata (a: DFA Sigma)(b: DFA Sigma) : DFA Sigma :=
  createDFA Sigma Q delta_union q0 qend_intersection.
```

Příkladem složitějšího tvrzení může být následující formule: $A1$ přejde slovem w ze stavu $q1$ do stavu $q1'$ a zároveň $A2$ přejde slovem w ze stavu $q2$ do stavu $q2'$ právě tehdy a jen tehdy, když $A3$ vznikající sjednocením $A1 \cup A2$ přejde slovem w ze stavu $(q1, q2)$ do stavu $(q1', q2')$.
Pomocí zjednodušeného matematického zápisu lze tvrzení vyjádřit takto.

$$(q1q2)w = (q1', q2') \equiv (q1w = q1' \wedge q2w = q2')$$

Je zřejmé, že pro přesnější zápis pomocí Coq je nutné blíže specifikovat přechod - *transition* při přijetí slova.

```
Inductive transition: Q -> Word Sigma -> Q -> Prop :=
|trans_empty: forall q, transition q nil q
|trans_ind : forall q q' q'' h t, delta q h = q' -> transition q' t q'' -> transition q (h::t)
```

Ekvivalentní zápis tvrzení v Coqu má pak následující podobu.

```
Lemma Union_transition : forall Sigma (Automaton_a Automaton_b : DFA Sigma),
  let automaton_union := Union_automata Sigma Automaton_a Automaton_b in
  let Q1 := Q Sigma Automaton_a in
  let Q2 := Q Sigma Automaton_b in
      forall (w: Word Sigma)(q1 q1': Q1)(q2 q2': Q2),
        transition Sigma automaton_union (pair q1 q2) w (pair q1' q2') <->
        (transition Sigma Automaton_a q1 w q1' /\ transition Sigma Automaton_b q2 w q2').
```

Kapitola 5

Závěr

Práce přibližuje základní práci v prostředí dokazovacího nástroje Coq a ukazuje možnosti transformace výrazů při dokazování.

Následně je tato znalost aplikována na doménu konečných automatů, kdy je výstupem formalizace deterministického konečného automatu a následná ukázka konstrukce souvisejících tvrzení.

Práce ukazuje postup pro formalizaci v Coqu a může sloužit jako vhodný podklad pro práce navazující. Vedle sjednocení by bylo vhodné práci rozšířit o další množinové operace, jako je průnik nebo doplněk, dalším zjevným pokračováním by mohly být konstrukce nad automatem nedeterministkým, jako je například převod na DKA.

Práce celkově slouží jako příklad možného využití Coqu jako nápomocného systému pro dokazování tvrzení se zaručenou správností.