CSE203 FINAL PROJECT - REGEXP

Formalizing Regular Expressions Using Coq

Remy Seassau, Nazila Sharifi Amina

November 27, 2022

WHAT WE LEARNED FROM THE PROJECT

REGEX

This project was our first in-depth encounter with regex. We were rigorously introduced to the following concepts:

- Languages
- Kleene closures
- Regular expressions
- Brzozowski's derivative

We were able to learn the formal definitions and to get some intuition on these notions.

USING COQ

This project was instructive in terms of teaching us how to use a proof-solver in a general context.
We used a variety of concepts such as:

- Definitions
- Checking definitions
- Checking practical applications

USING COQ

This project was instructive in terms of teaching us how to use a proof-solver in a general context.
We used a variety of concepts such as:

- Definitions
- Checking definitions
- Checking practical applications

```
Lemma CheckPattern : forall a b,eqR
(RE_K (RE_Disj (RE_Atom a) (RE_Atom b)))
(RE_K (RE_Conc (RE_K (RE_Atom a)) (RE_K (RE_Atom b)))).
```

SPECIFICS

The fact that the concatenation of the empty language with any language is empty.

The concept of the Brzozowski derivative was quite interesting, specifically because of the name and the parallels that can be drawn with numerical derivatives.

The inductive definition of regexp reminded us that inductive definitions can be completely abstract in the sense that the only difference between the disjunction and the concatenation is how we interpret them.

OUR MAIN DIFFICULTIES

PRONOUNCING BRZOZOWSKI

Upon encountering the name, we were first baffled as to how to pronounce it correctly. Thankfully, our Slovak roommate was able to help us and we managed to continue the project with the correct pronunciation in mind.

THE CASE OF LANGK

One of the first hurdles we encoutered in the project was the definition of the Kleene closure of a language:

- The definition wasn't straightforward to impliment
- The first definition we used made proving langKK quite difficult

THE CASE OF LANGK

One of the first hurdles we encoutered in the project was the definition of the Kleene closure of a language:

- The definition wasn't straightforward to impliment
- The first definition we used made proving langKK quite difficult

```
Definition langK L : language :=
  fun w => exists (ws : list word),
  (forall w1, In w1 ws -> L w1) /\ concat ws = w.
```

THE CASE OF LANGK

One of the first hurdles we encoutered in the project was the definition of the Kleene closure of a language:

- The definition wasn't straightforward to impliment
- The first definition we used made proving langKK quite difficult

```
Definition langK L : language :=
  fun w => exists (ws : list word),
  (forall w1, In w1 ws -> L w1) /\ concat ws = w.
```

We ended up changing the definition of langK to an inductive one and had to rewrite some of the Lemmas we had already proven.

TRICKY INDUCTIONS

One problem we encountered several times over the course of the project was inductions that we were not able to prove in all cases.

This was solved by a trick that we remembered from one of the TDs: moving something down before the induction in order to use it later.

```
(* Q16. show that the `Brzozowski` function is correct.

Lemma Brzozowski_correct (x : A) (w : word) (r : regexp) :
  interp (Brzozowski x r) w -> interp r (x :: w).
Proof.
move: w.
induction r; simpl.
+ done.
+ move => w h. simpl in h. done.
```

Figure: Moving w down in Q16

USING THE COQ LIST LIBRARY

Using the predefined lemmas given by the standard list library did not come naturally for us.

In fact, it was only later in the project that we realised that it was possible to use such lemmas, even causing us to write our own versions before doing so.

USING THE COQ LIST LIBRARY

Using the predefined lemmas given by the standard list library did not come naturally for us.

In fact, it was only later in the project that we realised that it was possible to use such lemmas, even causing us to write our own versions before doing so.

```
Theorem app_nil: forall A (l: list A), app l nil = l.
Proof.
move => A l.
induction l.
+ simpl. reflexivity.
+ simpl. rewrite IHl. reflexivity.
Qed.
```

Figure: Our implementation of app_nil_r

THANK YOU

```
Lemma nat_eq_dec : forall (n m : nat), {n = m} +
induction n.
 destruct m as [|m].
  left.
  reflexivity.
 right.
 discriminate.
destruct m as [|m].
 right; discriminate.
destruct (IHn m) as [Hm|Hm].
 left.
 rewrite Hm.
 reflexivity.
right.
intros Hnm.
apply Hm.
injection Hnm.
tauto.
Defined.

Eval compute in (nat_eq_dec 2 2).
Eval compute in (nat_eq_dec 2 1).
Definition pred (n:nat) : option nat :=
match n with
```

```
n : nat
IHn : forall m : nat, {n = m} + {n <> m}
m : nat
Hm : n = m
_____________________________________(1/2)
S m = S m


_____________________________________(2/2)
{S n = S m} + {S n <> S m}
```

Final Words

We can honestly say that of all our final projects this is the one we enjoyed working on the most. Thank you for making a project this interesting and informative.