SEQUENCES

RAUL H.C.LOPES

1. Sequences

A draft on **sequences**, that should one day deal with

- **prepend** and **append**
- definitional axioms
- fundamental operations
- properties: theorems
- formal presentation with **PVS** (maybe **Isabelle**)
- Haskell representations
- lists, stacks, queues, deques
- C matters

2. Introduction

Definition 1. $x. = .y \triangleq (x) = (y)$

3. *Prepend* sequences: axioms

Sequences are usually seen as inductive classes obtained from an initial element, say NIL, and constructors that add new elements to the beginning or the the end of the sequence. When construction of new sequences from existing ones is restricted to the addition of elements to the beginning of the existing sequence, they are frequently called **cons** sequences.[1] Gries and Schneider [1] call them **prepend** sequences. The axioms for the inductive class of a sequence over A (denoted $seq\{A\}$) are stated bellow and they are based on those presented by them. First there are the axioms defining the inductive class, then there is extensionality, stating equality/difference, and at last a *Principle of Induction.*

Definition 2. *The set of sequences over a set A, denoted* $seq\{A\}$*, is given by the following axioms:*

[1]The term **cons** was first coined by Lisp community that used it as the name of the function that would build a new **list** (a **Lisp** name for sequence) from a pair of elements of the language.

- *The empty sequence* ε *is a* $seq\{A\}$.

$$\varepsilon \in seq\{A\} \tag{1}$$

- *Prepending (consing) an element from* A *to an existing* $seq\{A\}$ *produces a* $seq\{A\}$.

$$c \in A \wedge x \in seq\{A\} \Rightarrow c \lhd x \tag{2}$$

- *Closure*

In addition, there is extensionality (used establishing equality between different sequences, and the difference between the empty sequence and any nonempty sequence.

Definition 3. *Extensionality for sequences.*

- *The empty sequence is different from any nonempty sequence.*

$$c \lhd x \neq \varepsilon \tag{3}$$

- *Two nonempty sequences are equal when their components are respectively equal.*

$$(b \lhd x = c \lhd y) = (b = c \wedge x = y) \tag{4}$$

As always with inductive classes there is *induction over sequences.*

Definition 4. *The* ***Principle of Induction*** *for* $seq\{A\}$.
In order to prove that $\forall x : x \in seq\{A\} : P.x$ *holds, prove that:*

- P *holds for the empty sequence*
- *if* P *holds for* x*, for all* $x \in seq\{A\}$ *then* P *holds for* $c \lhd x$*, for all* $c \in A$.

That gives:

(5)

$$(\forall x : x \in seq\{A\} : P.x) = (P.\varepsilon \wedge \forall c \in A : x \in seq\{A\} : P.x \Rightarrow P(c \lhd x))$$

Exercise 1. *Prove the following theorems:*

- $x = \varepsilon \vee (\exists b \in A, y \in seq\{A\} : x = b \lhd y)$
- $c \lhd x \neq x$

3.1. **Head, tail and member.** The next two axioms establish the *head*, *car* in **Lisp** parlance, of a sequence and its *tail*

- The head of $a \lhd x$ is a.

$$head(a \lhd x) = a \tag{6}$$

- The tail of $a \lhd x$ is x.

$$tail(a \lhd x) = x \tag{7}$$

Membership states that nothing is a member of the empty sequence and that for something to be a member of a sequence it must either be its head or be in its tail.

- a thing cannot be a member of an empty sequence.

$$a \notin \varepsilon \tag{8}$$

- a thing occurs in the head of a sequence or in its tail.

$$(a \in c \lhd x) = (a = c \vee a \in x) \tag{9}$$

Exercise 2. *Define a set of clauses for representing the sequences axioms. Assume that your sequences are all built over the same set: you don't have to parameterize over A, the set of elements in the sequence.*

3.2. **Haskell consing.** An inductive data type for sequences is easly built in Haskell with the construction **data**. The **data** constructor introduces the constructor for sequences over A, **NIL** for ε and **:<|:** for $\lhd$. The associated **deriving** clause introduces a principle of extensionality.

```
data Seq a   = NIL  |    a :<|: (Seq a)
                      deriving (Eq, Read, Show)
```

Functions to retrieve *head, tail* and test for membership mirror exactly their logical counterparts.

```
head (a :<|: x) = a

tail (a :<|: x) = x

member a NIL = False
member a (b :<|: x) = a == b or member a x
```

Unfortunately, or maybe not, Haskell will not generate a Principle of Induction. But you can easily forge one and prove a few properties for the function of *catenation* of sequences.

```
NIL :^: y = y
(a :<|: x) :^: y = a :<|: (x :^: y)
```

Exercise 3. (1) *Give logical axioms for **cat**.*

(2) *Prove*
   (a) $x :\hat{}: NIL = NIL$
   (b) $x :\hat{}: (y :\hat{}: z) = (x :\hat{}: y) :\hat{}: z$
   (c) $b \in x :\hat{}: y = (b \in x \vee b \in y)$

4. Append sequences

Sequences get more interesting when adding to the end (appending) is allowed. The following axioms define the operation:

Definition 5. ***Appending to sequences***

- *appending to empty sequence is the same consing to it.*

$$\varepsilon \triangleright c = c \triangleleft \varepsilon \tag{10}$$

- *Appending to a nonempty sequence gives a the head of the original sequence followed by the appending to its tail.*

$$(a \triangleleft x) \triangleright c = b \triangleleft (x \triangleright c) \tag{11}$$

Exercise 4. *Prove:*

(1) $x \triangleright c \neq \varepsilon$
(2) $(x \triangleright b = y \triangleright c) = (x = y \wedge b = c)$
(3) $a \in x \triangleright c = (a \in x \vee a = c)$

4.1. **Haskell append sequence.** A new type is introduced below that admits append and prepend constructors.

```haskell
data PrepApSeq a = NIL |
                   a :<|: PrepApSeq a |
                   PrepApSeq a :|>: a
    deriving (Eq, Show, Read)
```

Exercise 5. *Write a Haskell function that converts a general* ***PrepApSeq*** *to one containing just* **:<|:** *operators.*

Exercise 6. *Write Haskell functions to:*

(1) *retrieve the head of* ***PrepApSeq****.*
(2) *retrieve the tail of* ***PrepApSeq****.*
(3) *test membership in* ***PrepApSeq****.*

4.2. **New operations.**

Definition 6. x ***is a subsequence of*** y *($x \sqsubseteq y$)* $\triangleq$ x *is obtained from* y *by eliminating some of its elements.*

Exercise 7. (1) *Give axioms to formally define the subsequence relation.*

Solution.

The axioms for subsequences are:

- *Empty subsequence*

$$\varepsilon \sqsubseteq x \tag{12}$$

- *Nonempty cannot be subsequence of empty*

$$\neg(a \triangleleft x \sqsubseteq \varepsilon) \tag{13}$$

- *Subsequence starting at the beginning of sequence*

$$c \triangleleft x \sqsubseteq c \triangleleft y . = . x \sqsubseteq y \tag{14}$$

- *Subsequence of another sequence*

(15) $$b \neq c \Rightarrow (b \triangleleft x \sqsubseteq c \triangleleft y. = .b \triangleleft x \sqsubseteq y)$$

□

(2) *Prove the reflexivity of* $\sqsubseteq$.
(3) *Prove* $x \sqsubseteq \varepsilon = (x = \varepsilon)$
(4) *Prove* $c \triangleleft x \sqsubseteq y. = .y \neq \varepsilon \wedge ((c = head.y \wedge x \sqsubseteq tail.y) \vee (c \triangleleft x \sqsubseteq tail.y))$

Solution.

There are two things to prove:

- *That* $y \neq \varepsilon$
  This one follows from axiom 13.
- *that* $((c = head.y \wedge x \sqsubseteq tail.y) \vee (c \triangleleft x \sqsubseteq tail.y))$
  This demands proving that either:
  - $(c = head.y \wedge x \sqsubseteq tail.y)$
    Proved using axiom 14, and definitions of tail and head.
  - *or* $(c \triangleleft x \sqsubseteq tail.y)$
    Follows from axiom 15, and definition of tail.

□

(5) *Give a **Haskell** function that implements the subsequence relation.*

Consider the following definitions: a segment of a sequence x is a subsequence of adjacent elements of x; a prefix of a x is an initial segment of x; a suffix of x is a final segment of x.

Exercise 8. (1) *Formalize segment, prefix and suffix as relations between sequences.*
(2) *Give **Haskell** functions for those relations.*

Consider the following definition: x is a permutation of y if x is a rearregement in the order of the elements of y.

Exercise 9. (1) *Give the axioms defining the relation* $perm(x, y)$*, meaning* x *is a permutation of* y.

Solution.

Use the following axioms, where a.del.y is meant to be the sequence resulting from the removal of the first occurrence of a from y.

$$a.del.\varepsilon = \varepsilon$$
$$a.del.a \triangleleft y = y$$
$$b \neq a \Rightarrow a.del.b \triangleleft y = b \triangleleft a.del.y$$

Notice that x is permutation of y when each element of x has one occurence in y: in other words, removing any element of x from x and y should produce respectively x' and y' that are still permutaions of each other.

□

(2) *Prove perm is reflexive.*
(3) *Prove perm is symmetric.*
(4) *Prove perm is transitive.*
(5) *Give a **Haskell** functions implementing perm. Prove its correctness.*

5. Stacks, queues, and deques

Consider the definitions:

- **Stack**: a sequence where items can be added or retrieved only from the beginning of the structure.
- **Queue**: a sequence where items can be added at end and retrieved from the beginning of the structure.
- **Deque**: a sequence where items can be both added to and retrieved from the beginning and end of the structure.

Exercise 10. *Define the axioms for a queue.*

Exercise 11. *Define a queue in Haskell. Can you define one that take time $(O(1)$ for either inserting an item at the end or retrieving the head?*

6. Haskell lists

...more to come...

7. Linked Lists

... one day ...

References

[1] David Gries and Fred B.Schneider, *A logical approach to discrete mathematics*, Springer-Verlag, 1993.