CMPSCI 250: Introduction to Computation

Lecture #20: Strings and String Operations
David Mix Barrington
18 October 2013

Strings and String Operations

- Peano Axioms for Strings
- Pseudo-Java for the string Class
- Defining the String Operations
- Proof By Induction For Strings
- Concatenating Strings Adds Lengths
- Concatenation is Associative
- Reversal of a Concatenation

Peano Axioms for Strings

- We define our string data type for any fixed alphabet Σ by induction, just as we defined the naturals.
- The basic way to make new strings from old is by appending a letter to a string.
- We can define five "Peano axioms" for strings, which are much like the Peano axioms for the naturals.

Peano Axioms for Strings

- 1. λ is a string.
- 2. If w is a string and a is a letter in Σ, then wa is a string.
- 3. If wa and vb are the same string, then w = v and a = b (i.e., no string is formed by appending in two different ways).
- 4. Any string other than λ is equal to wa for some string w and letter a.
- 5. The only strings are those made from λ by the second axiom.

The Pseudo-Java `string` class

- We can think of our string operations as being built up from basic string methods in our pseudo-Java programming language.
- Remember that unlike real Java `String` objects, pseudo-Java `string` values are primitives.
- We have a method to test whether a string is empty, a method to append a letter, and two "inverses" for the append operation.

The Pseudo-Java `string` class

- The inverse methods throw an exception if called on an empty string. If called on a string wa, `last` returns a, the last letter, and `allButLast` returns the string w.

```
public static boolean isEmpty( ){...}

public static string append (string w, char
                             a) {...}

public static char last (string w) {...}

public static string allButLast (string w)
{...}
```

Clicker Question #1

```
public static char foo (string w) {
   if (isEmpty(w)) return '!';
   w = allButLast(w);
   if (isEmpty(w)) return '!';
   return last(w);}
```

- What does this method return on input "abca"?
- (a) '!'
- (b) 'a'
- (c) 'b'
- (d) 'c'

Answer #1

```
public static char foo (string w) {
   if (isEmpty(w)) return '!';
   w = allButLast(w);
   if (isEmpty(w)) return '!';
   return last(w);}
```

- What does this method return on input "abca"?
- (a) '!'
- (b) 'a'
- (c) 'b'
- (d) *'c'*

Defining String Operations

- We defined operations on naturals recursively, first saying what the operation does with argument 0 and then defining what argument n+1 does based on what argument n does.
- Here we can do much the same thing for strings.
- Each operation comes from a simple recursive definition.

Length and Concatenation

- The code for these methods follows fairly directly from the inductive definitions.

```
public static natural length (string w) {
   if (isEmpty(w)) return 0;
   return
      successor(length(allButLast(w)));}

public static string cat (string w, string
                          x) {
   if (isEmpty(x)) return w;
   return append
      (cat(w, allButLast(x)), last(x));}
```

The Code for Reversal

- There's two interesting wrinkles in this code for the reversal operation.
- We need the `cat` operation to be defined.
- Since `cat` takes two `string` arguments, we have an implicit type cast from the character `last(w)` to a `string`.

```
public static string rev (string w) {
   if (isEmpty(w)) return w;
   return cat(last(w),
              rev(allButLast(w)));}
```

Clicker Question #2

```
public static string mix (string w) {
   return cat(allButLast(rev(w)),
              rev(allButLast(w)));}
```

- What is the output of this method on input "dog"?
- (a) "gd"
- (b) "good"
- (c) "doggod"
- (d) no output, exception is thrown

Answer #2

```
public static string mix (string w) {
   return cat(allButLast(rev(w)),
              rev(allButLast(w)));}
```

- What is the output of this method on input "dog"?
- (a) "gd"
- (b) *"good"*
- (c) "doggod"
- (d) no output, exception is thrown

Proof by Induction for Strings

- As we noted above, an alternate version of the fifth Peano Axiom for strings allows us to prove statements of the form $\forall x$: P(x), where x is of type `string`, by induction on all strings.
- We need a base case of P(λ), and then an inductive case for each letter a in Σ, of the form $\forall w$: P(w) $\rightarrow$ P(wa).
- With binary strings we must prove P(w) $\rightarrow$ P(w0) and P(w) $\rightarrow$ P(w1) for arbitrary w (or just prove P(w) $\rightarrow$ (P(w0) $\wedge$ P(w1)).

Proof By Induction for Strings

- Each of our recursive definitions defines f(wa), for example, in terms of f(w).
- So if we can phrase our statement P(w) so that it talks about f(w), then information about f(w) should be useful in talking about f(wa) when we prove P(wa).
- We'll finish the lecture by doing three such inductive proofs.

Concatenation Adds Lengths

- Our first proof relates a string operation to an operation on naturals.
- When we concatenate two strings, we add their lengths. Let's prove the statement ∀u: ∀v: |uv| = |u| + |v|, where we use "|u|" to mean the length of u.
- We let u be an arbitrary string and use string induction on v.
- The statement P(v) is "|uv| = |u| + |v|", or "length(cat(u, v)) == plus(length(u), length(v))".

Concatenation Adds Lengths

- The base case P(λ) says that $|u\lambda| = |u| + |\lambda|$, which is true because the definitions tell us that $u\lambda = u$, $|\lambda| = 0$, and $|u| = |u| + 0$.
- We assume P(v) and look at P(va), which says $|u(va)| = |u| + |va|$.
- To prove this we will need to use the inductive clauses of two recursive definitions, that of concatenation and that of length.

Concatenation Adds Lengths

- The definition of concatenation says that u(va) = (uv)a, and the definition of length then says that |u(va)| = |(uv)a| = successor(|uv|).
- The definition of length says that |va| = successor(|v|), and the definition of addition says that |u| + successor(|v|) = successor(|u| + |v|).
- We finish by using the IH to replace |uv| by |u| + |v|. This completes the inductive step for arbitrary v and a.

Clicker Question #3

- Let Σ = {a, b,..., z}. Define a function cardie(w) from Σ^* to Σ^* by the rules cardie(λ) = λ and for any string w and any letter a, cardie(wa) = cat("z", cardie(w)). What does this function do?
- (a) returns a string of z's with the same length as w
- (b) returns the reversal of w
- (c) returns λ for any w
- (d) returns "z" for any w

Answer #3

- Let Σ = {a, b,..., z}. Define a function cardie(w) from Σ^* to Σ^* by the rules cardie(λ) = λ and for any string w and any letter a, cardie(wa) = cat("z", cardie(w)). What does this function do?
- (a) *returns a string of z's with the same length as w*
- (b) returns the reversal of w
- (c) returns λ for any w
- (d) returns "z" for any w

Concatenation is Associative

- Now we prove ∀u:∀v:∀w: (uv)w = u(vw), where we use parentheses to indicate the order of operations. We let u and v be arbitrary, and use string induction on w with P(w) as "(uv)w = u(vw)" or "cat(cat(u, v), w) == cat(u, cat(v, w))".
- The base case P(λ) is "(uv)λ = u(vλ)", which reduces to uv = uv by the definition of concatenating with λ.

Concatenation is Associative

- We assume P(w) and try to prove P(wa), which says "(uv)(wa) = u(v(wa))". (Again we must be careful of notation, as we are using the same notation for appending and concatenation.)
- The LHS is ((uv)w)a, and the RHS is u((vw)a) which we can convert to (u(vw))a, each time using the definition of concatenation.
- The IH of "(uv)w = u(vw)" now lets us prove that the LHS equals the RHS, by appending an a to each side of this equation.

Reversal of a Concatenation

- Finally we prove the rule relating reversal and concatenation, the statement $\forall u{:}\forall v{:}(uv)^R = (v^R)(u^R)$. (For example, ("bulldog")^R = ("dog")^R("bull")^R = "godllub".) We'll let u be arbitrary and use string induction on v.
- The base case P(λ) is "$(u\lambda)^R = \lambda^R u^R$". We can prove this with the rules $u\lambda = u$ and $\lambda^R = \lambda$, and the theorem $\lambda u = u$, which is easy to prove by induction on u.

Reversal of a Concatenation

- So we assume P(v), "$(uv)^R = v^R u^R$", and try to prove P(va), "$(u(va))^R = (va)^R u^R$".
- The LHS is $((uv)a)^R$ by the definition of concatenation, and $a(uv)^R$ by the definition of reversal. (Note that this last is the concatenation of the two strings a and $(uv)^R$.)
- The RHS is $(av^R)u^R$ by the definition of reversal, and then $a(v^R u^R)$ by associativity of concatenation from the previous slide. We can now equate these forms of the LHS and the RHS by using the IH once. This completes the inductive step and thus also the proof.