Lecture 15: Recursive Data Types, Definitions, and Structural Induction

Date: October 4, 2019.

Strings

Recursive definition of Strings: Let A be a non-empty set of *characters* (or *letters, symbols*). A is called an *alphabet.* The set of *strings* over alphabet A, denoted A^* is defined as follows.

- **Base Case:** The empty string λ is in A^*.
- **Constructor Case:** If $a \in A$ and $s \in A^*$ then $\langle a, s\rangle \in A^*$.

$A=\{0,1\}$ $\quad$ 10101 $\quad$ seq. $(1,0,1,0,1) \in A^5$.

$\langle 1, \langle 0, \langle 1, \langle 0, \langle 1, \lambda\rangle\rangle\rangle\rangle\rangle$ ← representation according to the recursive definition

Length of Strings: Length $|s|$ of a string s is defined recursively as

- **Base Case:** $|\lambda|$ is defined to be 0
- **Constructor Case:** $|\langle a, s\rangle|$ is $1 + |s|$.

$|\langle 1,\langle 0,\langle 1,\langle 0,\langle 1,\lambda\rangle\rangle\rangle\rangle\rangle| = 1+|\langle 0,\langle 1,\langle 0,\langle 1,\lambda\rangle\cdots\rangle| = 1+1+|\langle 1,\langle 0,\langle 1,\lambda\rangle\cdots\rangle| = \cdots = 1+1+1+1+1+|\lambda| = 1+1+1+1+1+0 = 5$

Concatenation: The concatenation of string s with t, denoted $s \cdot t$ is recursively defined as

- **Base Case:** $\lambda \cdot t$ is t
- **Constructor Case:** $\langle a, s\rangle \cdot t$ is $\langle a, s \cdot t\rangle$.

$\langle 1, \langle 0, \lambda\rangle\rangle \cdot \langle 1, \langle 1, \lambda\rangle\rangle = \langle 1, \langle 0, \lambda\rangle \cdot \langle 1, \langle 1, \lambda\rangle\rangle\rangle = \langle 1, \langle 0, \lambda \cdot \langle 1, \langle 1, \lambda\rangle\rangle\rangle\rangle = \langle 1, \langle 0, \langle 1, \langle 1, \lambda\rangle\rangle\rangle\rangle$

obvious statement $\forall s.\ \lambda \cdot s = s$.

Proposition 1. *$s \cdot \lambda = s$ for all $s \in A^*$.*

Induction predicate : $P(s)$: $s \cdot \lambda = s$.

Base Case : When $s = \lambda$. $\lambda \cdot \lambda = \lambda$ (by defn of $\cdot$)

Ind hyp : Assume $P(s)$.

Ind Stp : $P(\langle a, s\rangle)$: $\langle a, s\rangle \cdot \lambda = \langle a, s \cdot \lambda\rangle = \langle a, s\rangle$ (ind hyp)

Proposition 2. *For all $s, t \in A^*$, $|s \cdot t| = |s| + |t|$.*

Induction predicate : $P(s) : \forall t.\ |s \cdot t| = |s| + |t|$

Base Case: $s = \lambda$. $|\lambda \cdot t| = |t| = |t| + 0 = |t| + |\lambda|$ ✓

Ind Hyp : Assume $P(s)$ holds

Ind stp: $|\langle a, s\rangle \cdot t| = |\langle a, s \cdot t\rangle| = 1 + |s \cdot t| = 1 + |s| + |t|$ (ind hyp)
$= |\langle a, s\rangle| + |t|$

$P(t)$: $\forall s\ |s \cdot t| = |s| + |t|$

Ind Stp : $|s \cdot \langle a, t\rangle|$

Structural Induction: Let P be a predicate on a recursively defined data type R. If

- $P(b)$ is true for each base case element $b \in R$, and
- for all k-argument constructors $\mathbf{c}$

$$[P(r_1) \text{ AND } P(r_2) \text{ AND } \cdots \text{ AND } P(r_k)] \text{ IMPLIES } P(\mathbf{c}(r_1, r_2, \ldots r_k))$$

for all $r_1, r_2, \ldots r_k \in R$

then $P(r)$ is true for all $r \in R$.

Well matched Brackets [[ ]] [ ]]] [

Definition: The set of well-match strings, RecMatch, can be defined as

- **Base Case:** $\lambda \in$ RecMatch
- **Constructor Case:** If $s, t \in$ RecMatch then $\langle [, \lambda \rangle \cdot s \cdot \langle], \lambda \rangle \cdot t \in$ RecMatch.

If $s, t \in$ RecMatch then $[s]t \in$ RecMatch

Number of characters: $\#_c(s)$ is the number of occurences of c in s, and can be defined recursively as

- **Base Case:** $\#_c(\lambda) = 0$
- **Constructor Case:** $\#_c(\langle a, s \rangle) = \#_c(s)$ if $a \neq c$, and $\#_c(\langle a, s \rangle) = 1 + \#_c(s)$ if $a = c$.

Exercise

$\#_c(s \cdot t)$
$= \#_c(s) + \#_c(t)$

Proposition 3. *Every string in* RecMatch *has an equal number of* [*and*] *symbols.*

Induction Predicate: $P(s)$: $\#_[(s) = \#_](s)$.

Base Case: $s = \lambda$: $\#_[(\lambda) = 0 = \#_](\lambda)$. $P(\lambda)$ holds.

Ind hyp: Assume $P(s)$ and $P(t)$.

Ind Step: $\#_[([s]t) = \#_[(\langle [, s]t \rangle) = 1 + \#_[(s]t)$
$= 1 + \#_[(s) + \#_[(]) + \#_[(t)$
$= 1 + \#_[(s) + \#_[(t)$

$\#_](]s]t) = \#_](\langle [, s]t \rangle)$
$= \#_](s]t) = \#_](s) + \#_](]) + \#_](t)$
$= \#_](s) + 1 + \#_](t)$