Abstract Domain Theory: STRINGS

Functions

is-empty[x]	test for the empty string
is-char[x]	test for valid character
is-string[x]	test for valid string
prefix[u, x]	attach character to front of string

Decomposition

if not[x = empty-string], then x = prefix[head[x], tail[x]]

Induction

if F[empty-string] and
    forall x: if not[x = empty-string], then if F[tail[x]] then F[x]
then forall x: F[x]

Recursion

base: F[empty-string]
general: F[prefix[u,x]] = prefix[F[u], F[x] }

Facts

is-string[empty-string]
is-string[u] for all characters u
is-string[prefix[u, x]]

Rules

not[prefix[u, x]] = empty-string
prefix[u, x] = prefix[v, y] iff u=v and x=y
prefix[u, empty-string] = u

if x=y then prefix[x,z] = prefix[y,z]
if x=y then prefix[u,x] = prefix[u,y]

head[prefix[u, x] = u
tail[prefix[u, x] = x

concatenate[empty-string, y] = y
concatenate[prefix[u,x], y] = prefix[u, concatenate[x,y]]
is-string[concatenate[x,y]]
concatenate[concatenate[x,y], z] = concatenate[x, concatenate[y,z]]
reverse[empty-string] = empty-string
reverse[prefix[u,x]] = concatenate[reverse[x], u]
reverse[concatenate[x,y]] = concatenate[reverse[y], reverse[x]]
reverse[reverse[x]] = x

reverse-accum[empty-string, res] = res
reverse-accum[prefix[u,x], res] = reverse-accum[x, prefix[u,res]]

length[empty-string] = 0
length[prefix[u,x]] = length[x] + 1
length[concatenate[x,y]] = length[x] + length[y]

A symbolic proof by induction

To prove: rev[rev[x]] = x x is of type STRING

Base case:	Rule applied:
rev[rev[empty-str]] =?= empty-str	1. problem
rev[empty-str] =?= empty-str	2. rev[empty-str] = empty-str
empty-str =?= empty-str	3. rev[empty-str] = empty-str
QED	4. identity

Inductive case:

rev[rev[x]] =?= x	1. problem
rev[rev[u • x]] = u • x	2. assume by induction rule
rev[rev[x] * u] = u • x	3. rev[a•b] = rev[b]*a
rev[u] * rev[rev[x]] = u • x	4. rev[a*b] = rev[b]*rev[a]
u * rev[rev[x]] = u • x	5. rev[a] = a a is a char
u • rev[rev[x]] = u • x	6. lemma a*b=a•b a is a char
rev[rev[x]] = x	7. a•b = a•c iff b=c
QED	

Lemma:

u * x =?= u • x	1. problem
(u • x) * y = u • (x * y)	2. prefix/concatenate distribution
(u • empty-str) * y = u • (empty-str * y)	3. let x=empty-str
u * y = u • (empty-str * y)	4. a • empty-str = a
u * y = u • y	5. empty-str * a = a
QED	