Topics in Automated Deduction (CS 576)

Elsa L. Gunter
2112 Siebel Center
egunter@illinois.edu
http://www.cs.illinois.edu/class/sp10/cs576/

General Recursive Functions: `fun`

Example:

```
fun fib :: "nat ⇒ nat" where
    "fib 0 = 0" |
    "fib 1 = 1" |
    "fib (Suc(Suc x)) = (fib x + fib (Suc x))"
```

Not primitive recursive because of `fib(Suc(Suc x))` on left, and because of `fib(Suc x)` on right.

`fun`: Rules of Use

Compared to `primrec`, very few restrictions:

- Can be used to define functions over any type
- Clauses in `fun` must be equations
- Left-hand side is function being defined applied to terms built from data constructors, distinct variables and wildcards
- Right-hand side is a expression made from the function being defined, the variables in the argument on the left, and previously defined terms
- If clauses overlap, first takes precedence.
- Calculates a measure from lexicographic ordering of some collection of arguments

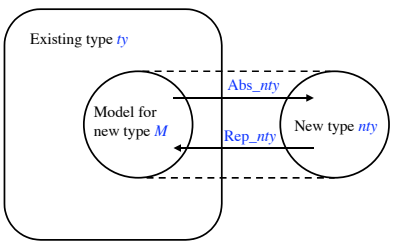
Example: `sep`

Define a function for putting a separator between all adjacent elements in a list:

```
fun sep ::  "'a * 'a list => 'a list" where
 "sep(a, []) = []" |
 "sep(a, [x]) = [x]" |
 "sep(a, x#y#zs) = x # a # sep(a,y#zs)"
```


Demo: General Recursive Functions

Introducing new types



Properties of a Defined Type

Syntax for `typedef`:

`typedef` *nty* = "*modeling_set*"
Proof of $\exists x.x \in$ *modeling_set*.

introduces a new type named *nty*, and functions

`Abs_`*nty* :: *ty* ⇒ *new_ty*
`Rep_`*nty* :: *new_ty* ⇒ *ty*

where *modeling_set*::*ty*

Properties of a Defined Type

Theorems automatically provided:

Rep_*nty*: `Rep_`*nty* $x \in$ *modeling_set*
Abs_*nty*_inverse: `Abs_`*nty*(`Rep_`*nty* x) $= x$
Rep_*nty*_inverse:
$y \in$ *modeling_set* $\Longrightarrow$ `Rep_`*nty*(`Abs_`*nty* y) $= y$
Abs_*nty*_inject:
$[|x \in$ *modeling_set* $: y \in$ *modeling_set*$|] \Longrightarrow$
(`Abs_`*nty* x = `Abs_`*nty* y) $= (x = y)$
Rep_*nty*_inject: (`Rep_`*nty* x = `Rep_`*nty* y) $= (x = y)$