Topics in Automated Deduction (CS 576)

Elsa L. Gunter
2112 Siebel Center
egunter@cs.uiuc.edu
http://www.cs.uiuc.edu/class/sp08/cs576/

Format for Inductive Relations Definitions

```
inductive R :: "τ ⟶ bool" where
  ⟦R(a1,1); ...; R(a1,n); A1,1; ...; A1,k⟧ ⟹ R(a1) |
    ... |
  ⟦R(am,1); ...; R(am,l); Am,1; ...; Am,j⟧ ⟹ R(am)
```

where $A_{i,j}$ are side conditions not involving R.

Format for Mutual Inductive Relations Definitions

```
inductive
R1 :: "τ1 ⟶ bool" and
...
Rn :: "τn ⟶ bool" where
  ⟦Ri(a1,1); ...; Rj(a1,n); A1,1; ...; A1,k⟧ ⟹ Rk(a1) |
    ... |
  ⟦Rm(am,1); ...; Rn(am,l); Am,1; ...; Am,j⟧ ⟹ Rp(am)
```

where $A_{i,j}$ are side conditions not involving any R_k.

Example with Mutual Recursion

```
inductive
Even :: "nat ⇒ bool" and
Odd :: "nat ⇒ bool" where
ZeroEven [intro!]: "Even 0" |
OddOne [intro!]: "Odd (Suc 0)" |
OddSucEven [intro]: "Odd n ⟹ Even (Suc n)" |
EvenSucOdd [intro]: "Even n ⟹ Odd (Suc n)"
```


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)"
```


Introducing new types