John Hughes
Greg Witt

List and Set Proofs in Guru

A Brief Introduction to Guru Lists

- User-declared inductive datatype
- Incrementally and uniquely built up

```
Inductive list : Fun(A:type).type :=
      nil : Fun(A:type).<list A>
    | cons : Fun(A:type)(a:A)(l:<list A>). <list A>.
```

(cons nat three (cons nat two (cons nat one (nil nat))))

Anatomy of a Guru Proof

```
Define not_not : Forall(b:bool). {(not (not b)) = b }:=
  foralli(b:bool).
  case b with
 ff => trans cong (not (not *)) b_eq
       trans join (not (not ff)) ff
       symm b_eq
| tt => transs cong (not (not *)) b_eq
                  join (not (not tt)) tt
                  symm b_eq  end
       end.
```

~ (~b) = ~ (~ff)

~(~ff) = ff

ff = b

A Simple List Example

The goal:

- Convert a list to a vector
- Convert the vector back to a list
- Prove that the converted list == the original

Converting a List to a Vector

```
Define list_to_vec : Fun( A : type )( L : < list A > ).
                          <vec A (length A L)> :=
fun list_to_vec( A : type )( L : < list A > ) :
                          <vec A (length A L)>.
 match L with
          nil _  =>  cast (vecn A) by
                     cong <vec A *>
                     symm trans cong (length A *) L_eq
                     join (length A nil) Z

 | cons _ a L'   =>
                cast (vecc A (length A L')(list_to_vec A L')) by
                     cong <vec A *>
                     symm trans cong (length *) L_eq
                     join (length (cons a L)) (S (length L'))
 end.
```

Proving Our List Hasn't Changed

```
Define list_vec_list:
   Forall (A:type)(L:<list A>).
  {(vec_to_list (list_to_vec L)) = L} :=
   foralli (A:type).
   induction (L:<list A>)
   return {(vec_to_list (list_to_vec L)) = L} with
   ...
```

A More Complex Example

The Goal

- Prove that a set is a subset of itself
- Define *subset* with code, not a formula
- Define subset functions on lists

- Define list_subset : Fun(A: type)
  (eqA: Fun(a b:A). bool) (l1 l2 :<list A>) . bool

This Required a Few Helper Lemmas

Lemmas

- eqlist_total
- list_subset_total
- list_seteq_total
- seteq_symm

Functions

- list_setmbr
- set_equals

The Role of the Total Lemmas

- Total lemmas prove that a function terminates
- Knowing a function terminates, allows us to exhaust all cases in proofs
- In Guru, we must prove that even the most trivial functions terminate
  - For example: if b is a Boolean, b must be true or false; there is no non-terminating case

The "Subset of Self" Code

```
Define list_SubsetOfSelf :
Forall(l': <list A>)
        (l : <list A>)
                .
                .
                .
                .
{(list_subset A eqA l (append l' l)) = tt} :=
```

Conclusion

- Now, you've seen a few different examples, including:
  - How lists are represented in Guru
  - How sets are implemented as lists
  - How lists can be cast as vectors
  - The use of helper lemmas in a complex proof