Formalizing the Theory Concept in Nuprl

Jason Hickey
March 15, 1994

Outline

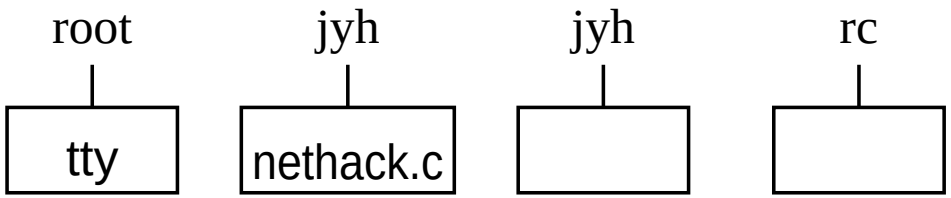
Goals of formalization

Goal generfization

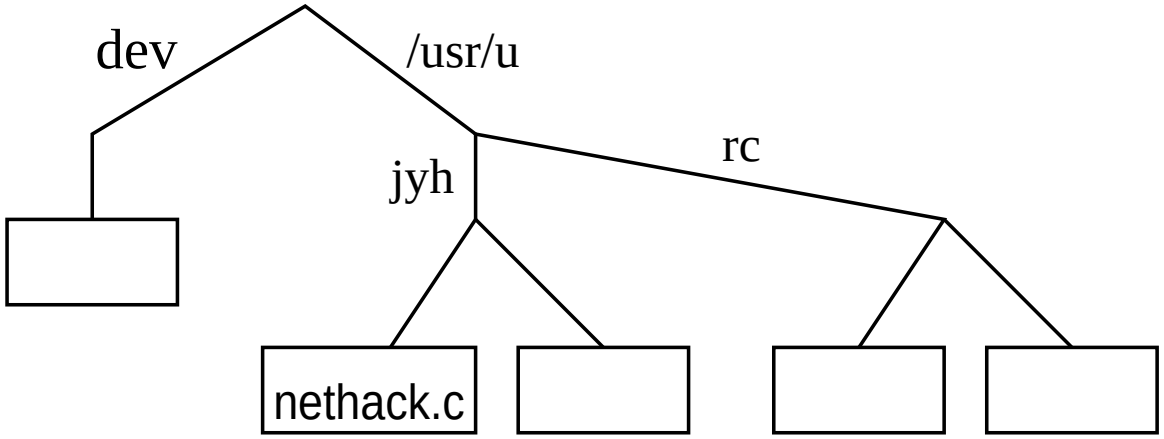
- Operate within a context of Theorems, Axioms, Definitions
- Primary goal is to extend the context
- Problems
  - flat name space
  - unrestricted access
  - generfl form of inheritance

Flat name space

Flat namespace



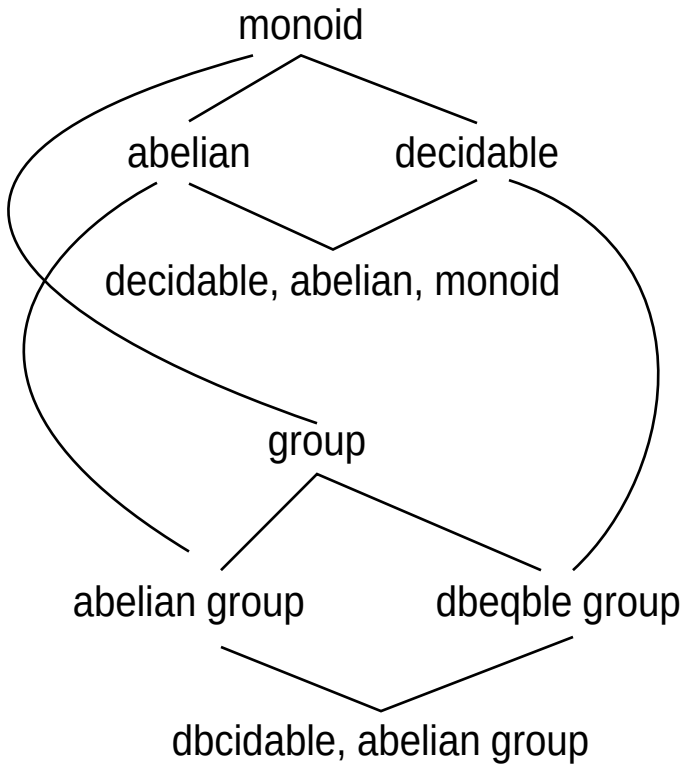
Hierarchy



Unrestricted Access

Inheritance

Running example



Reflection

- Want to reason about
  - proofs
  - theories
  - tactics

Object Orientation

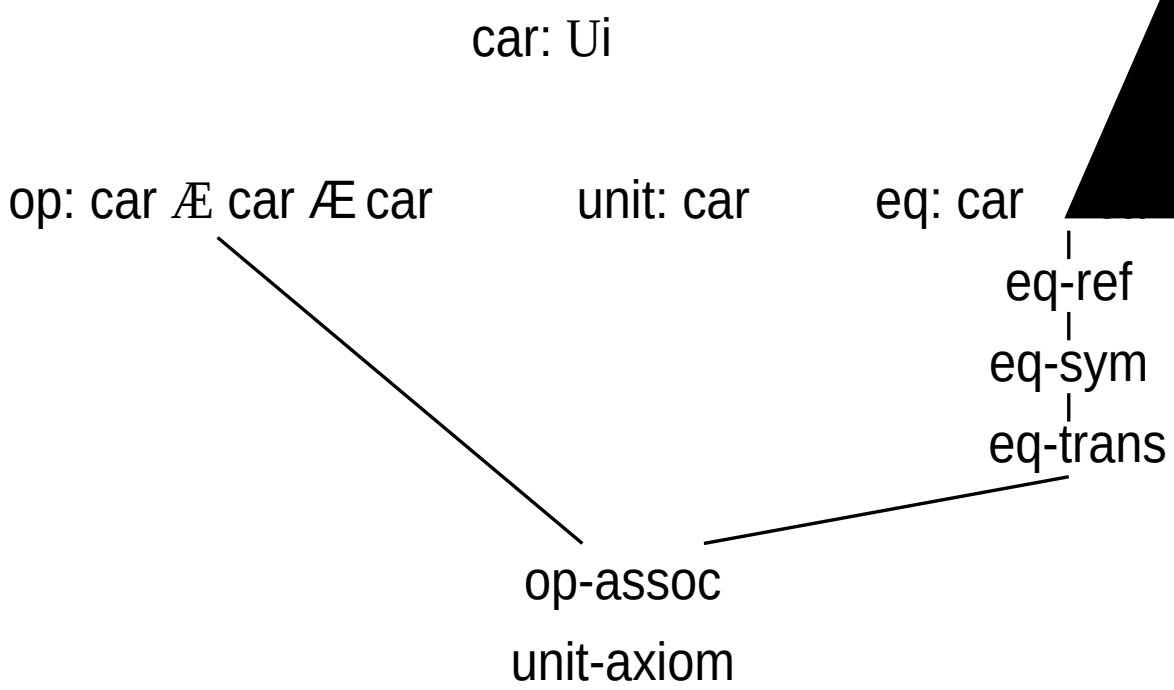
- Abstract Data Typing
- Inheritance

```
* ABSmonoid
Monoid[fig] == car:U[fig]
               unit:car
            £ op:(car ! car ! car)
            £ eq:(car ! car ! P[fig])
            £ eq-ref:(8a:car. a eq a)
            £ eq-sym:(
```

Generalize the dependent product

- Number of elements is arbitrary

Least constraining ordering



TheoryItem

- Reference this axiom by name
- Use inhabitants of previous axioms by looking thr ancestors
- What is an axiom?

```
* ABS theory
Theory[fig] == rec(Theory.name:IDList
                  £ label:Atom
                  £ flags:List of Z
                  £ lib:List of TheoryLibType
                  £ preds:List of Theory
                  £ Axiom?
```

Axiom type

- Define lookup function
  - Ld0up(Type) name in preds
    Axiom: (name: ID $\rightarrow$ L
    $\rightarrow$ Ui

Example

- MonoidCar: <"car", [ ], lparents, lookup. Ui>
  MonoidUnit:
  <"unit",
  [MonoidCar],
  λparents, lookup.
  Lookup "car" in parents using MonoidCar>q7

Real Example

```
* ABS monoid car
MonoidCarfig = fig carfig
```

```
* ABS monoid car thyMonoidCarThyfig== Axiom:
        Name = car/ ,
        Flags = CNil,
        Lib = CNil,
        Preds = CNil,
        Axiom(par, pre) = MonoidCar
```

MonoidUnit

```
* ABS monoid unit preds
MonoidUnitPreds⟨fig⟩ == MonoidCarThy⟨fig⟩::CNil

* ABS monoid unit
MonoidUnit(parents, preds) == TheoryEnv[parents, preds]
                                  car = car
                              in⟨funit⟩
                                  car

* ABS monoid unit thy
MonoidUnitThy⟨fig⟩ == Axiom:
                          Name = unit/ ,
                          Flags = CNil,
```

MonoidOpAssoc

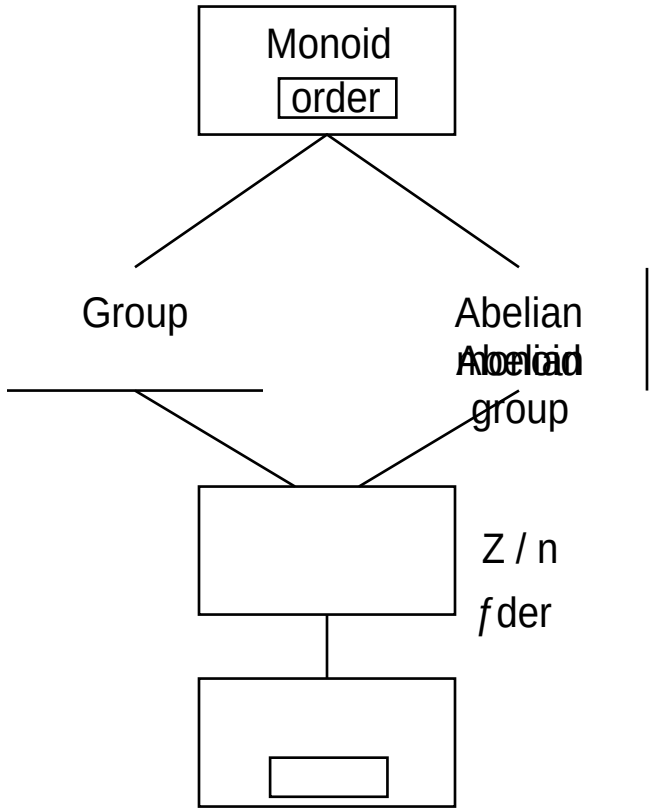
```
* ABS monoid unit preds
ds              fig                 fig::CNil

* ABS monoid unit
parents, preds) == TheoryEnv[parents, preds]
                                        car = car
                                   infunitg
                                        car

* ABS m1id unit thy
M1idUnitThy  fig == Axiom:
                              Name = unit/
```

Theorems

- Axiom: "every monoid has an order"
- Theorem extract: function that computes the order
- May want several proofs of the theorem depending on the particular monoid



Why do we want Theorems?

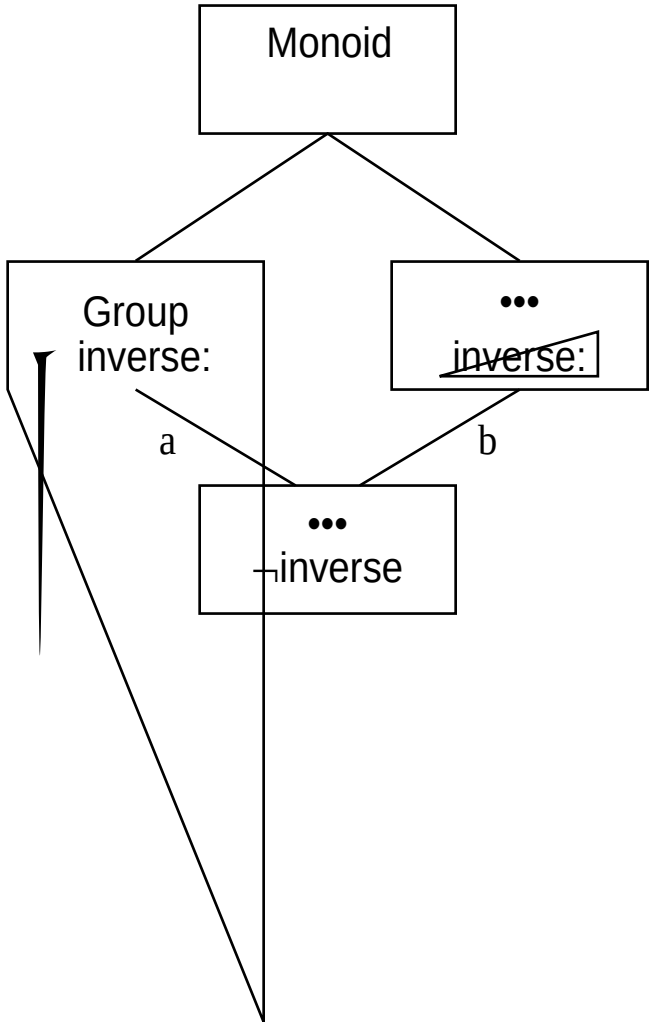
- To show something is true
  - "Every monoid of prime order is cyclic"
- To compute
  - "Every monoid has an order"
- Don't handle extracts!

  A Theorem is just a link from an axiom to an object in the

Naming considerations

Can pull in any theory

Must inherit all axioms as well



Problems

- rec(Theory. •••) does not work!
  - Typing rule is too simple.
- Build type on top of Y-combinator

Y-combinator

- Can hide all instances of bounds by using derived rules
- All proofs are by (normal) induction
- No fixed-pdbnt semantics

Data structure

- In a functional language where equality is undecidable, how can we represent a DAG? (Would like an equality over strings)
- My solution
  - need an arbitrary total order.
  - the DAG are integer indices into the total order
  - Not elegant!

Results

- Can reasd•about theories
  A theory is a normal object, reasd•ing is typically by inductid•
- Examples
  - Kndwledge mdndtd•icaly increases
  - Equivalence of theories
  - "Lifting" of theorems: any theorem can be lifted to be beneath its immediate predecessors

Results II

- Naming is better

Problems

Future work

- Package theories in a hierarchy
  - Conceptual blocks
    Assist in naming

TheoryType

```
* ML theory type switch ml
% Givena  TheoryItemType fig(n) , construct a product of all the axioms
    in the theory.  This effectively all the junk out of a theory
    tdV4(provide)-501(a)-507(type)-516(that)-516(can)-537(be)-510(quantified)-510(over.)]TJ-1.032 -1.2 TD
        case
        of inl( ) =>
            parents:TheoryTypch(inr       , item.preds)
            £ Switch x = item.term of
              Axiom  !  x parents (,name.Lookup(preds,f name in parents using
              Theorem  !  Unit  !  Unit| inr( ) =>
                                      Case item of
                                      CNil => Unit
                                      hd::tl => TheoryTypeSwitch(     , hd.theory) £ TheoryTypeSwitch(
```

TheoryLookup

```
* ML theory lookup switch ml
% Given a  TheoryItemTypefig(n) , and a term of  TheoryType(item) ,
    and a name of type  ID , return the term inhabinting the axiom by that name.
  %
  add rec def  Lookup[switch] name in term using item
       case switch
       of inl( ) =>
           if name 2 item.name
           then inr term.2
           elsewiLookup[inr ] name in term.1 using item.preds
           fi
         inr( ) =>
           Case item of
           CNil => inl
           hd::tl => casewiLookup[inl ] name in term.1 using hd.theory
                     of inl( ) =>
                         Lookup[inri] name in term.2 using tl
            inr (x)  =>
             inr x;;
```

```
* ML theory lookup type switch ml
% Give a  TheoryItemTypefig(n) , and a term of  TheoryType(item) ,
    and a name  ID , return the axiom of that name.
  %
  add rec def  Lookup(type)[switch] name in term using item

  of inl( ) =>
      if name 2 item.name
      Axiom(term.1) else Lookup(type)[switch] name in term.1 using item.preds
                         Else ! Unit
                              fi
                          | inr( ) =>
                              Case item of
                              CNil => inl Unit
                              hd::tl => case Lookup(type)[inl ] name in term.1 using hd.theory
                                        of inl( ) =>
                                            Lookup(type)[inr ] name in term.2 using tl
```