Abstract Programming in Nuprl

"The Majority Vote Algorithm"

- Jason Hickey

- November 23, 1993

Outline

- **Abstraction**
- **Majority vote algorithm**
- **Abstract implementation**
- **Solution to majority vote**
- **Concrete implementation**
- **Compare with other languages**

Themes of high-level programming

- **Abstraction**
  - Hides underlying implementation
    Packages structure into a modules for reuse

Transformation-real abstraction

- **We should be able to specify an abstraction without an implem bHntation**
- **Virtual program is transformed to an executable program by replacing the virtualfLst raction with a concrete impleotat ion.**

Majo

Remove all duplic

- **Given a bag X of size n and the majority k**
  - If b has at least k distinct values, delete them
  - Perform at most n ÷ k times
  - Every value occurring more than(t)2´ ÷ k in´ X is in´25(ev)19(ery)19(red

{B is a reduced ba} {or b[0 ; 1]}

od

:= |+ 1; B [{b[ ]};

Delete k disti\ct values {rom B ; i possible

Multiset

- **Multiset is the major data type–target for abstraction**
- **Follow presentation given in Manna and Waldinger**
- **Basic elements and operations**
  - empty set
  -
  -

Multiset Signature

car:U*f*i*g*

- £ empty:car
- £ gen:(car ! Atom ! car)
- £ member:(car ! Atom ! P*f*i*g*)
- £ equal:(car ! car ! P*f*i*g*)
- £ syneq:(car ! car ! P*f*i*g*)
- £ atomeq:(Atom ! Atom ! P*f*i*g*)
- £ ind:MSetIndType*fg*(Atom, car, empty, gen, syneq)

£ mem p:MSetMemType(Atom, car, empty, member, gen, equal, atomeq)

£ unique p:MSetUniqType(Atom, car, empty, gen, equal, syneq, atomeq)

£ equality p:MSetEqType(Atom, car, gen, equal)

£ atomeq p:MSetAEqType(Atom, atomeq)

£ syneq p:MSetSeqType(car, syneq)

£ B

Multiset Axioms

- induction principle
- uniqueness (free generation)
- equality axioms
- membership axioms

Multiset Theorems

```
* THM mset ind thm
8Atom:Ufig
   8m:MSetSigfig(Atom)
    8P:m.car  !  Pfig
     (8            · m.empty 2 m  )  P[b])
      )  (8b:m.car. 8             )  P[b [m x])
      )  f8b:m.car. P[b]g

* THM mset mem empty
8Atom:Ufig. 8m:MSetSigfig(Atom). 8x:Atom. :(x 2m m.empty)

* THM mset mem thm
8Atom:Ufig
   8m:MSetSig
```

Other functions and Theorems

- **Decomposition**
- **Equality Theorems**
- **MSet union**
- **MSet difference**
- **etc.**

Return to Majority Vote

- **Define a predicate that is true when an element has the majory vote:**
  - MajVoteProp[k](Atom, m, b, x)
  - x has a majority in b
- **Define aedicate tha isrue when a mise is a set of k distinct elements**
  - MajDistinctProp[k](Atom, m, b)
  - b is a set, and it has k distinct elements
- **Dfine a-step reduction**
  - ReducedBag[k](A, m, b $\rightarrow$c)
  - There is a set d with k distinct elements and c » d = b

Majority Vote II

- **Define a predicate fdPr a reduced bag**
  - MajRedBag[k](Atom, m, b $\rightarrow$ c)

    c has fewer than k distinct elements

    all elements x having a majority vote in b have a majority vote in c

Thed•ms

- **The elements with a majority vote are in every reduced bag**
- **Normalizat**
  - if b - c = d and d $\rightarrow$ e and c isE st of k distinct elements, then b $\rightarrow$ e
- **Main Result:**

  Fd• every bag, there is a reduced bag
  - Proof is by induction on the siz of the bag

    If Ethe bag has fewer than k distinct elements, done

    Otherwise, •move k distinct elements. The Eresult Eholds by induction.

Comparison of the two developments

- **Gries logic separates computation from verification**
  - Logic is classical
  - Every proof is for a particular program
-
  - Better--why do more than necessary?
    Worse--computation is hidden

ThorVgorithm is accompanied by a proof

Transformation

- **It would be nice to try out this algorithm**
- **Brief presentation of an implementation based on lists.**
  - empty: []
  - add: cons
  - member: member

Transformation II

- **All the prdHperties dH14(f)7(t)7(h)14(e mu)14(l)14(t)7(i)14**
- **Other options:**

  - function that returns the multiplicity df an element in a bag
- **Cdmputation of majority ve**
  - ReducedBag([1;1;2]) = [1]

What I learned in this process

- Time to learn the system--its huge!
- Existing documentation is good, but its incomplete
- Power of abstraction
- Power of transformation
- Delayed/lazy development

  All these things are poss

Proofs

- **These proofs are tedious!**
  Many of the proofs are small, and conceptually

  - " b: Bag " x: Atom b » x þ 0
  - Three of four steps to prove this

- **Many such trivial proofs**
- **Many well-formedness proofs**
- **Auto tactic is limited to a fixed set of rules**

Incremental Development

- **Small modifications wipe out an entire theorem or library**
  - Save the proof tree?
- **Hypothesis numbering**
- **Libraries as first class objects**
  - Axioms as assumptions
  - Naming

Advantages

- **Higher level of abstraction than any other language I know**
- **Transformation is natural**
- **The problem with tedious proofs can be solved with appropriate ATP's**

Other languages

- **Most languages do not have specifications**

Example in C++

```
class MSet f
private:
        LinkedList car;
public:
        Bool aeq(int; int);
        Bool eq(MSet x);
        Bool member(Atom x);
        void add(Atom x);
        Atom head
        Atom tail
        MSet extract(Atom xg;
```

Features of Object Orientation

Inheritance II

Modify features

Conclusion

- **Powerful abstraction and transformation mechanism**
- **Majority vote algorithm has a simple solution**
- **Proofs sn be tedious, but san be fixed with tastis help**
- **We can provide some features of an objest oriented language, do we want them all?**
- **What is the sorrect model for a theory?**