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
  - Provides high-level features
  - Nuprl provides pretty syntax
  - Useful for hypertext
- **Inheritance**
  - Add features to an existing object
  - Customize features of an existing object

Transformation-real abstraction

- **We should be able to specify an abstraction without an implementation**
- **Virtual program is transformed to an executable program by replacing the virtual abstraction with a concrete implementation.**

Majority Vote

- **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 n ÷ k in X is in every reduced bag for X

$P \quad 0 \leq \jmath \leq \natural \wedge \quad B$ is a reduced ba″ –or $b[0\jmath \quad -1]$
var $\jmath, B \ = 0, ¶\diamondsuit$
¶i“varia“t P; bou“d –u“ctio“ $\natural - \jmath\diamondsuit$
do $\jmath \neq \natural \rightarrow \jmath, B \ = \jmath + 1, B \cup ¶b[\jmath]\diamondsuit$;
Delete k disti“ct values –rom B, i–possible
od
¶B is a reduced ba″ –or $b[0\natural \quad -1]\diamondsuit$
Remove all duplicates –rom B

Multiset

- **Multiset is the major data type–target for abstraction**
- **Follow presentation given in Manna and Waldinger**
- **Basic elements and operations**
  - empty set
  - add element
  - membership
  - equality
  - atom equality
  - well-formedness

Multiset Signature

```
car:U{i}
  × empty:car
  × gen:(car → Atom → car)
  × member:(car → Atom → P{i})
  × equal:(car → car → P{i})
  × syneq:(car → car → P{i})
  × atomeq:(Atom → Atom → P{i})
  × ind:MSetIndType{i}(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
∀Atom:U{i}
   ∀m:MSetSig{i}(Atom)
    ∀P:m.car → P{i}
     (∀b:m.car. b ≡ m.empty ∈ m ⇒ P[b])
     ⇒ (∀b:m.car. ∀x:Atom. P[b] ⇒ P[b ∪m x])
     ⇒ {∀b:m.car. P[b]}

* THM mset mem empty
∀Atom:U{i}. ∀m:MSetSig{i}(Atom). ∀x:Atom. ¬(x ∈m m.empty)

* THM mset mem thm
∀Atom:U{i}
   ∀m:MSetSig{i}(Atom)
    ∀b:m.car. ∀x:Atom. ∀y:Atom. (y ∈m b ∪m x)⇐⇒
        (x = y ∈ m.Atom) ∨ (y ∈m b)

* THM mset mem eq
∀Atom:U{i}
   ∀m:MSetSig{i}(Atom). ∀b:m.car. ∀c:m.car. (b = c in m) ⇒
      (∀x:Atom. (x ∈m b)⇐⇒ (x ∈m c))
```

Two ways to define a size function

```
* THM mset size exists
∀Atom:U{i}
   ∀m:MSetSig{i}(Atom)
    ∃sizef:m.car ⟶ N
     (∀c:m.car. ∀x:Atom. sizef c < sizef c ∪m x)
     ∧ (∀b:m.car. sizef b = 0⇐⇒ (b = m.empty in m))

* THM mset size exists 1
∀Atom:U{i}. ∀m:MSetSig{i}(Atom). ∀b:m.car.
    ∃i:N. MSetSizeProp(Atom, m, b, i)

* ML mset size prop ml
% Test the number of elements in the multiset %
  add rec def  MSetSizeProp(Atom, m, b, i)
          ((b = m.empty in m) ∧ i = 0)
               ∨ (¬(b = m.empty in m)
                 ∧ (∀c:m.car
                     ∀x:Atom
                      (b = c ∪m x ∈ m.car)
                      ⇒ MSetSizeProp(Atom, m, c, i - 1))) ;;
```

Other functions and Theorems

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

Abstraction

- **These definition are totally abstract**
- **Set of operations described by their types and properties**
- **Any implementation inhabiting the type can be used as a multiset**
- **If we are only interested in the logical part of the theory, we don't have to provide an implementation.**
- **(Axioms may be inconsistent).**

Return to Majority Vote

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

Majority Vote II

- **Define a predicate for 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

Theorems

- **The elements with a majority vote are in every reduced bag**
- **Normalization:**
  - if b - c = d and d $\rightarrow$ e and c is a set of k distinct elements, then b $\rightarrow$ e
- **Main Result:**
  - For every bag, there is a reduced bag
  - Proof is by induction on the size of the bag

    If the bag has fewer than k distinct elements, done

    Otherwise, remove k distinct elements. The result holds by induction.
- **Gries's algorithm has two bags for efficiency**

Comparison of the two developments

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

    Its hard to understand the complexity of an algorithm

    The algorithm 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
  - equality: there is a bijection f: N Æ N such that b[i] = c[f i]
    uses nth_element function

Transformation II

- **All the properties of the multiset are properties of this simple implementation**
- **Its not a very efficient implementation**
- **Other options:**
  - function that returns the multiplicity of an element in a bag
- **Computation of majority vote**
  - 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 possible, and they work!**
- **Is this the new way to program?**

Proofs

- **These proofs are tedious!**
- **Many of the proofs are small, and conceptually trivial**
  - " 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**
- **Display forms are wonderful!**
- **Algorithms must obey their specification**

Other languages

- **Most languages do not have specifications**
- **Polya--how do you constrain the transformation?**
- **SPL--we can't deal with constraint management**

Example in C++

```
class MSet{
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 x);
};
```

Features of Object Orientation

- **Abstraction**
- **Inheritance**
  - Add features
  - For example, add a size function to MSet
  - Can do this, possibly a little awkward

$$\begin{array}{l}\textbf{class}\ MSetWithSize : \textbf{public}\ MSet\{ \\ \quad\quad \textbf{int}\ size(); \\ \}; \end{array}$$

Inheritance II

Modify features

- **Improve efficiency**
- **Modify behavior**
- **Implement a mset as a subclass of a list**
- **Reuse similar class in a different way**

$$\begin{array}{l}\textbf{class}\ MSetLinkedList : \textbf{private}\ LinkedList\{ \\ \qquad Bool\ eq(Atom\ b, Atom\ c); \\ \};\end{array}$$

Conclusion

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