# A Solution to the Rockwell Challenge

Hanbing Liu

June 30, 2003

# Outline

- The Rockwell “challenge”
- Key observations and my approach
- Proof sketch
- Generalization

# The Rockwell Challenge

- Data structure represented as memory cells
  - Two kinds of information encoded:
    - Relations between nodes
    - Information in data fields of nodes
- Reasoning about “dynamic” updates
- The bigger picture: getting abstraction back

In particular:

- Data structure: a "tree" of A-type node

  "A" nodes have 4 words. Words 0 and 3 are scalars. Words 1 and 2 point to "A" nodes.

- Operation a-collect, which collect all cells.
- We need to show:

```
(defthm rd-over-a-mark-objects
  (let ((list (a-collect ptr n ram)))
    (implies (and (not (member addr list))
                  (unique list))
             (equal (g addr (a-mark-objects ptr n ram))
                    (g addr ram)))))
```

- Other properties similar to above

# Key Observations

- “Link” cells vs. “data” cells
- Data structure: “shape” decided by “link” cells

  “Shape” vs. values of “data” fields
- Update during travesal “shape” may change:

  Imagine ram' is after updating ram

  (a-collect ptr n ram') not equal (a-collect ptr n ram)
- However, given unique condition, “shape” should not change.

# Proof Sketch and Key Lemmas

- Main goal:

```
(defthm rd-over-a-mark-objects
  (let ((list (a-collect ptr n ram)))
    (implies (and (not (member addr list))
                  (unique list))
             (equal (g addr (a-mark-objects ptr n ram))
                    (g addr ram)))))
```

where a-mark-objects is

```
(defun a-mark-objects (addr n ram)
  (if (zp n) ram
    (if (zp addr) ram
      (let ((ram’ (s addr *somevalue* ram)))
      (a-mark-objects (g (+ addr 2) ram’) (1- n) ram’)))))
```

# First Attempt: Direct Proof by Induction

Obvious choice of induction hint is (a-mark-objects ptr n ram)

Let

ram' be (s addr *the-value* ram)

ptr' be (g (+ ptr 2) ram')

and n' be (- n 1)

We assume:

```
(let ((list' (a-collect ptr' n' ram')))
  (implies (and (not (member addr list'))
                (unique list'))
           (equal (g addr (a-mark-objects ptr' n' ram'))
                  (g addr ram'))))
```

- Complications:
  - No obvious relation between (a-collect ptr' n' ram') and (a-collect ptr n ram)
  - This theorem is not “strong” enough!

    Only about cells outside the structure do not change. We also know (and need the fact) that “link” cells do not change!
  - Without knowing “shape” not change, recursion pattern in (a-collect ptr' n' ram') can be different from (a-collect ptr n ram)
- Attempt failed!

# *Nth* Attempt: Distinguish “Link” and “Data” Cells

N: somewhere between 3-5.

- (unique (a-collect ptr n ram))

  “Link” cells are not overlapping with “data” cells

- Update to any non “link” cell

  “Shape” does not change. Classification of cells do not change.

- (a-mark-objects ptr n ram)

  The *first* update is to the “data” cell.

- Subsequent updates are also to original “data” cells
- “Data” cells are subset of cells used to represent the object
- Final goal proved.

# Variation in Actual Proof

- Group ptr, n, ram into one entity RC, *RAM configuration*
- Reduce a-mark-objects to (apply-A-updates *certain-sequence* RC)
- Prove *certain-sequence* is a subset of “data” cells from the original structure, where *certain-sequence* is (collect-a-updates-dynamic rc)

  To prove the third point above:

  - (collect-a-updates-static rc) is a subset.
  - unique implies non-intersect between “data” and “link” cells
  - Relate (collect-a-updates-static rc) and (collect-a-updates-dynamic rc)

# Key Lemmas

- a-mark-objects-alt-definition

```
(defthm a-mark-objects-alt-definition
  (equal (a-mark-objects addr n ram)
         (apply-a-updates (collect-a-updates-dynamic (make-ram-config addr n ram))
                          ram))
  :rule-classes :definition)
```

- “Shape” remain unchanged, if ...

```
(defthm set-non-link-cells-collect-equal
  (implies (not (member x (a-collect-link-cells-static rc)))
           (struct-equiv-A-ram-config (rc-s x v rc) rc)))
```

- First updated cell is not a link cell under certain hypothesis

```
(defthm addr-not-a-member-a-collect-link-cells-static
  (let ((n (n rc))
        (addr (addr rc)))
    (implies (and (not (zp n))
                  (not (zp addr))
                  (not (overlap (a-collect-data-cells-static rc)
                                (a-collect-link-cells-static rc))))
             (not (member addr (a-collect-link-cells-static rc))))))
```

- More ...

# Other Challenge Problems

- Operations on independent objects

```
(defthm read-over-bab
  (implies
   (let ((list (append (b-collect ptr1 n1 ram)
                       (a-collect ptr2 n2 ram)
                       (b-collect ptr3 n3 ram)
                       )))
     (and
      (not (member addr list))
      (unique list)))
   (equal
    (g addr (compose-bab ptr1 n1 ptr2 n2 ptr3 n3 ram))
    (g addr ram))))
```

- Permutation of operations

```
(defthm a-mark-over-b-mark
  (implies
   (let ((list (append (a-collect ptr1 n1 ram)
                       (b-collect ptr2 n2 ram))))
     (unique list))
   (equal
    (a-mark-objects ptr1 n1 (b-mark-objects ptr2 n2 ram))
    (b-mark-objects ptr2 n2 (a-mark-objects ptr1 n1 ram)))))
```

# Generalization

- The generalized concept of *structurally equivalent* memory configuration
- More data types: theorems like read-over-bab

  J's map idea: introduce a map from type of node to structure of a node.

  Generalize "update" (a-mark-object) and "crawl" (a-collet) operations to work on objects of different type.
- Arbitary composition of different operations

  Generalize update and "crawl" operations to work on sequence of "independent" objects.

  Prove permutation does not matter, if objects do not share structures.
- Operations that changes the "link" cells

# Summary

- Two kinds of information are encoded by a complex data structure.
- First kind is captured by a structural equivalence.
- We reduce dynamic updates of “data” fields to apply a corresponding sequence of updates.
- The sequence can be decided by statically for certain dynamic update operations.
- The approach is being generalized.