Memory Taggings and Dynamic Data Structures

J Strother Moore

Department of Computer Sciences
The University of Texas at Austin

Dynamic Datastructures in ACL2: A Challenge

David Greve and Matt Wilding

Nov. 2002

Background

Arrays, structures, and stacks are common data structures that can be found in some form in nearly every useful software system. The utility that such structures provide in implementing complex systems, however, hinges on the often overlooked assumption of independence. What the programmer often takes for granted - that modifying structure X will not change the value of structure Y - becomes a significant burden when the formal correctness of the final implementation is considered in the context of a linear address space.

The Records Book

```
(defthm g-of-s-redux
  (equal (g a (s b v r))
         (if (equal a b)
             v
             (g a r))))
```

Greve-Wilding Theorem 1

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

```
(defun a-collect (ptr n ram)
  ; ptr     +1     +2     +3      ...
  ; [ ]     [*]    [*]    [ ]     ...
  ;          a-node a-node
 (if (zp n)
     nil
  (if (zp ptr)
    nil
    (append
     (seq-int ptr 4)
     (a-collect (g (+ 1 ptr) ram) (1- n) ram)
     (a-collect (g (+ 2 ptr) ram) (1- n) ram)))))
```

```
(defun a-mark-objects (addr n ram)
  ; ptr     +1     +2     +3     ...
  ; [ ]     [*]    [*]    [ ]    ...
 (if (zp n) ram
   (if (zp addr) ram
     (let ((ram (s addr
                   (+ (g addr ram)
                      (g (+ 2 addr) ram))
                   ram)))
     (a-mark-objects (g (+ addr 2) ram)
                     (1- n)
                     ram)))))
```

Greve-Wilding Theorem 3

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

```
; b-nodes look like this:

; ptr     +1     +2      ...
; [*]     [*]    [ ]     ...
;  b-node  b-node
```

My Generalization

- arbitrary number of data types,
- each field declared to be either (mutable) data or (immutable) pointer to a fixed data type.

A typical “declaration”

```
((A . ("int" A A "double"))
 (B . (B B "int"))
 (C . ("char4" A "int" B C)))
```

Generic Marking Algorithm

```
(mutual-recursion

(defun mark (typ ptr n ram dcl)
  (let ((descriptor (cdr (assoc typ dcl))))
    (cond ((zp n) ram)
          ((zp ptr) ram)
          ((atom descriptor) ram)
          (t (let ((ram (s* typ ptr 0 ram dcl)))
               (mark-lst typ ptr 0
                         (- n 1)
                         ram dcl))))))
```

```
(defun mark-lst (typ ptr i n ram dcl)
 (let* ((descriptor (cdr (assoc typ dcl)))
        (slot-typ (nth i descriptor))
        (i (nfix i)))
  (cond ((zp ptr) ram)
        ((<= (len descriptor) i) ram)
        (t (let ((ram (if (symbolp slot-typ)
                          (mark slot-typ
                                (g (+ ptr i) ram)
                                n ram dcl)
                          ram)))
            (mark-lst typ ptr (+ 1 i) n ram dcl))))))
```

```
(defun s* (typ ptr i ram dcl)
 (let* ((descriptor (cdr (assoc typ dcl)))
        (i (nfix i))
        (slot-typ (nth i descriptor)))
  (cond
    ((zp ptr) ram)
    ((< i (len descriptor))
     (cond
      ((symbolp slot-typ)
       (s* typ ptr (+ 1 i) ram dcl))
      (t (let ((ram (s (+ ptr i)
                       (new-field-value typ ptr i ram dcl)
                       ram)))
          (s* typ ptr (+ 1 i) ram dcl)))))
    (t ram))))
```

```
(encapsulate
 (((new-field-value * * * * *) => *))

 (local (defun new-field-value (typ ptr i ram dcl)
          (declare (ignore typ ptr i ram dcl))
          0))

 (defthm new-field-value-s-commutes
  (implies
   (not (member addr
              (seq-int ptr (len (cdr (assoc typ dcl))))))
   (equal
    (new-field-value typ ptr i (s addr val ram) dcl)
    (new-field-value typ ptr i ram              dcl)))))
```

Generic Collection Algorithm

`(collect typ ptr n ram dcl)` returns a list of all the addresses involved in the representation of data structure of type `typ` at address `ptr` (to depth `n`) in `ram`.

`collect` and `collect-lst` are defined mutually recursively analogously to `mark` and `mark-lst`.

```
(mutual-recursion

(defun collect (typ ptr n ram dcl)
  (let ((descriptor (cdr (assoc typ dcl))))
    (cond ((zp n) nil)
          ((zp ptr) nil)
          ((atom descriptor) nil)
          (t (append (seq-int ptr (len descriptor))
                     (collect-lst typ ptr 0
                                  (- n 1)
                                  ram dcl))))))
```

```
(defun collect-lst (typ ptr i n ram dcl)
  (let* ((descriptor (cdr (assoc typ dcl)))
         (slot-typ (nth i descriptor))
         (i (nfix i)))
    (cond ((zp ptr) nil)
          ((<= (len descriptor) i) nil)
          (t (append (if (symbolp slot-typ)
                         (collect slot-typ
                                  (g (+ ptr i) ram)
                                  n ram dcl)
                       nil)
                     (collect-lst typ ptr (+ 1 i)
                                  n ram dcl))))))
```

Challenge Theorem 1

```
(defthm challenge-theorem-1
 (implies
   (and
    (not (member addr (collect typ ptr n ram dcl)))
    (unique (collect typ ptr n ram dcl)))
   (equal (g addr (mark typ ptr n ram dcl))
          (g addr ram))))
```

Challenge Theorem 3

```
(defthm challenge-theorem-3
 (implies
   (unique (append (collect typ1 ptr1 n1 ram dcl)
                   (collect typ2 ptr2 n2 ram dcl)))
   (equal
    (mark typ1 ptr1 n1 (mark typ2 ptr2 n2 ram dcl) dcl)
    (mark typ2 ptr2 n2 (mark typ1 ptr1 n1 ram dcl) dcl)))
  :hints ...)
```

Statistics

This problem required approximately

- 20 `defuns`
- 90 `defthms`

Obstacles

- Obstacle 1: mutual recursion
- Obstacle 2: mutable pointers
- Obstacle 3: reflexivity

Obstacle 1: Mutual Recursion

```
(mutual-recursion
 (defun mark (typ ptr n ram dcl) ...)
 (defun mark-lst (typ ptr i n ram dcl) ...))

(defun mark-fn (fn typ ptr i n ram dcl) ...)

(defthm collect-is-collect-fn
  (and (equal (collect typ ptr n ram dcl)
              (collect-fn :ONE typ ptr i n ram dcl))
       (equal (collect-lst typ ptr i n ram dcl)
              (collect-fn :ALL typ ptr i n ram dcl))))
```

Obstacle 2: Mutable Pointers

Suppose `mark` mutates and then chases a pointer. Then it may mark addresses not visited by `collect`.

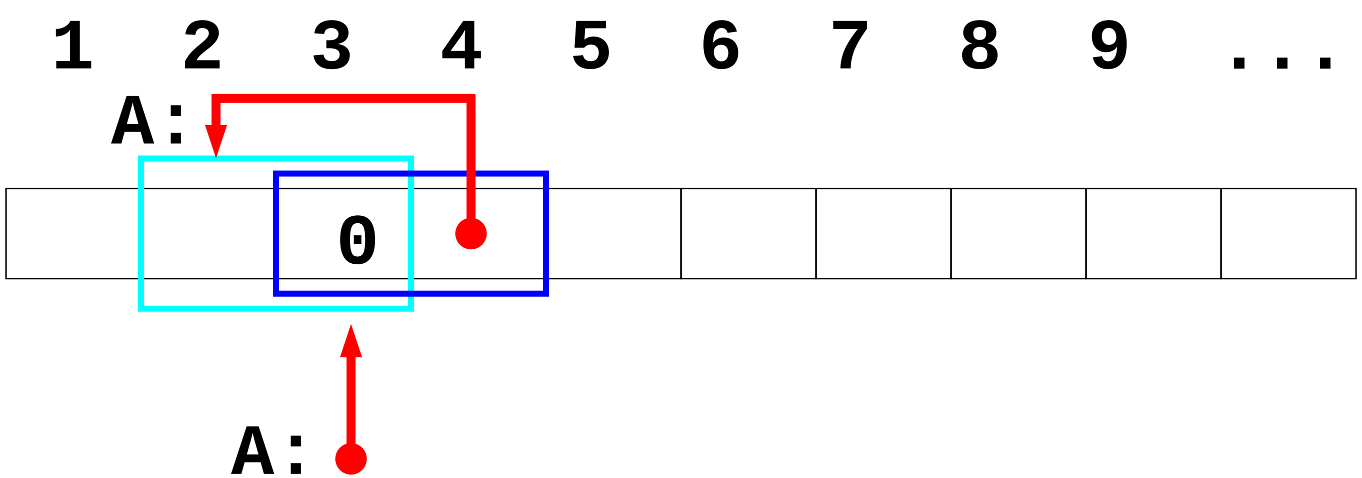
That could prevent:

```
(defthm challenge-theorem-1
 (implies
   (and
    (not (member addr (collect typ ptr n ram dcl)))
    (unique (collect typ ptr n ram dcl)))
   (equal (g addr (mark typ ptr n ram dcl))
          (g addr ram))))
```

How Might Mark Mutate a Pointer?

declaration:

`(A . ("data" A))`

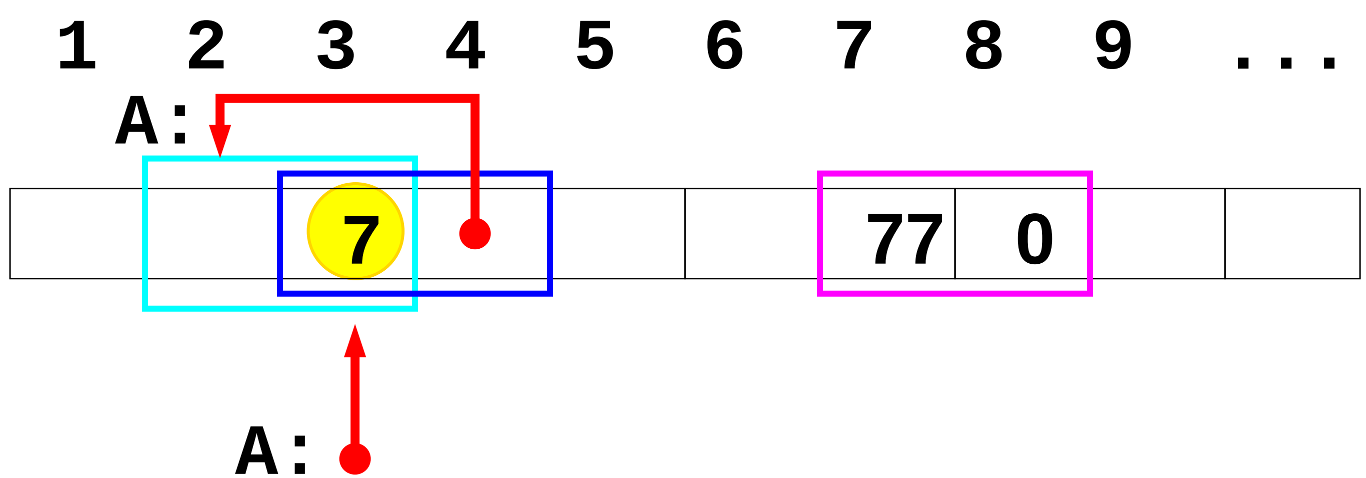


`collect = '(3 4 2 3)`

How Might Mark Mutate a Pointer?

declaration:

```
(A . ("data" A))
```

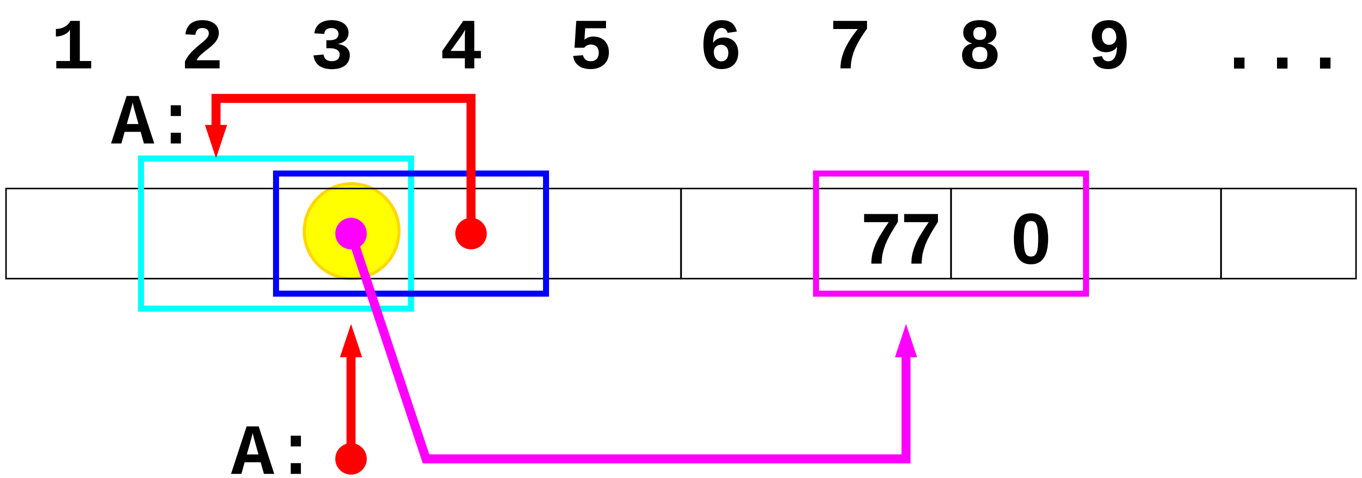



```
collect = '(3 4 2 3)
```

How Might Mark Mutate a Pointer?

declaration:

`(A . ("data" A))`

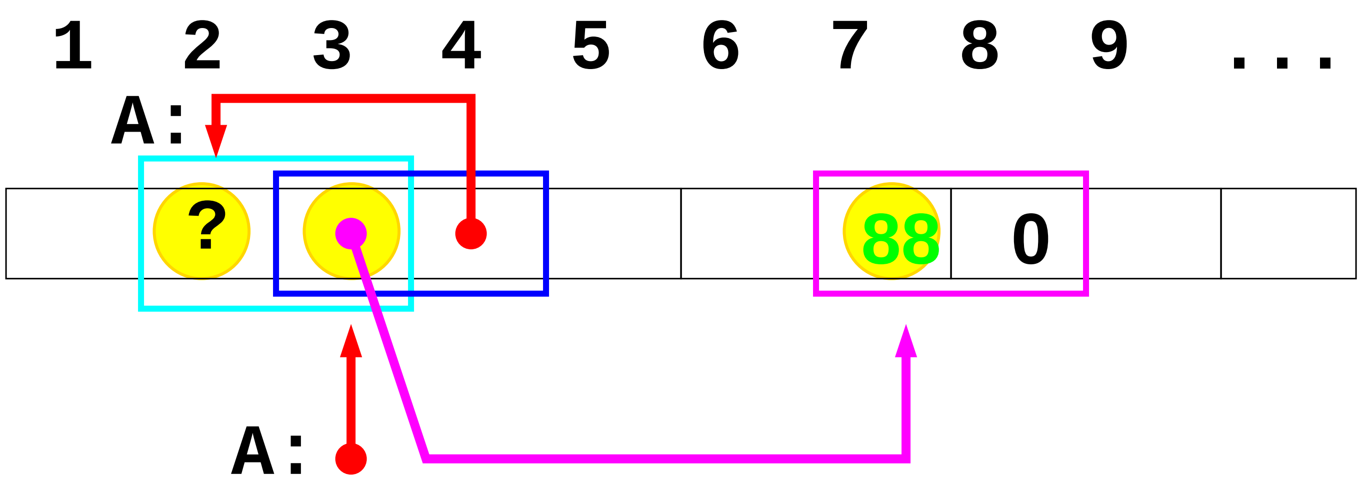


`collect = '(3 4 2 3)`

How Might Mark Mutate a Pointer?

declaration:

```
(A . ("data" A))
```




```
collect = '(3 4 2 3)
```

```
(defthm challenge-theorem-1
 (implies
   (and
    (not (member addr (collect typ ptr n ram dcl)))
    (unique (collect typ ptr n ram dcl)))
   (equal (g addr (mark typ ptr n ram dcl))
          (g addr ram))))
```

Obstacle 2 - Restated

Prove that if the reachable addresses are unique, the reachable addresses in `(mark typ ptr n ram dcl)` are the same as those in `ram`.

```
(implies
  (unique (collect typ ptr n ram dcl))
  (equal (collect typ ptr n (mark typ ptr n ram dcl) dcl)
         (collect typ ptr n ram dcl)))
```

Obstacle 3: Reflexivity

```
(defun mark-lst (typ ptr i n ram dcl)
 (let* ((descriptor (cdr (assoc typ dcl)))
        (slot-typ (nth i descriptor))
        (i (nfix i)))
  (cond ((zp ptr) ram)
        ((<= (len descriptor) i) ram)
        (t (let ((ram (if (symbolp slot-typ)
                          (mark slot-typ
                                (g (+ ptr i) ram)
                                n ram dcl)
                          ram)))
            (mark-lst typ ptr (+ 1 i) n ram dcl))))))
```

Obstacle 3: Reflexivity

```
(defun mark-lst (typ ptr i n ram dcl)
 (let* ((descriptor (cdr (assoc typ dcl)))
        (slot-typ (nth i descriptor))
        (i (nfix i)))
  (cond ((zp ptr) ram)
        ((<= (len descriptor) i) ram)
        (t (let ((ram (if (symbolp slot-typ)
                          (mark slot-typ
                                (g (+ ptr i) ram)
                                n ram dcl)
                          ram)))
             (mark-lst typ ptr (+ 1 i) n ram dcl))))))
```

Proving any theorem by induction about `mark` will be a problem because an induction hypothesis will be about `mark`!

In particular, consider the requirement of Obstacle 2:

```
(implies
  (unique (collect typ ptr n ram dcl))
  (equal (collect typ ptr n (mark typ ptr n ram dcl) dcl)
         (collect typ ptr n ram dcl)))
```

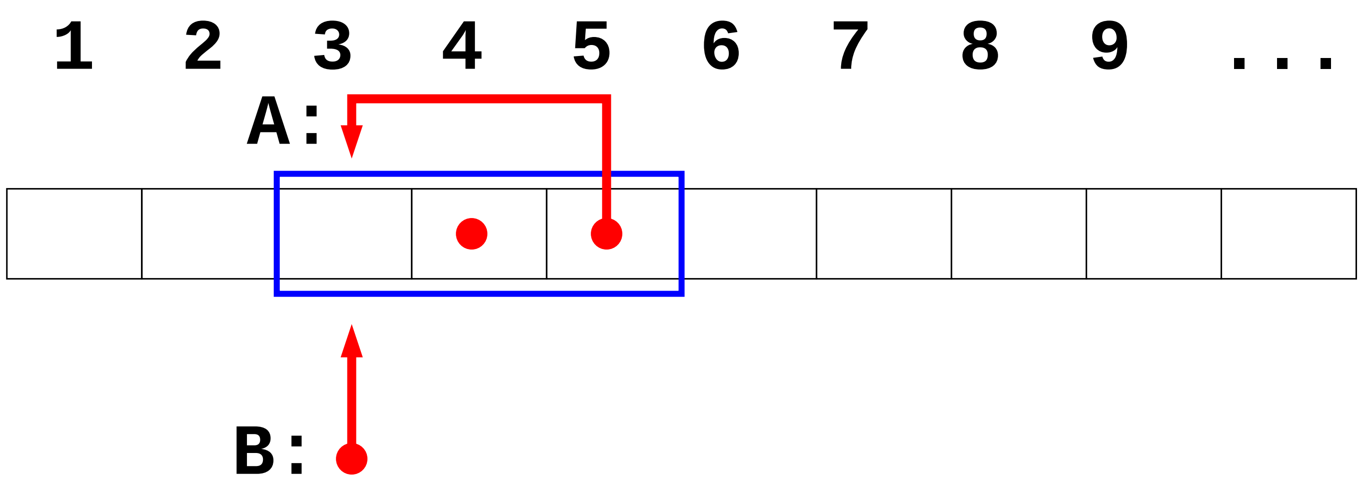
Induction Step for Obstacle 2:

```
(implies
 (and ...
      (implies
       (unique (collect typ ptr n (mark ...) dcl))
       (equal (collect typ ptr n (mark typ ptr n (mark ...) dcl) dcl)
              (collect typ ptr n (mark ...) dcl))))
 (implies
  (unique (collect typ ptr n ram dcl))
  (equal (collect typ ptr n (mark typ ptr n ram dcl) dcl)
         (collect typ ptr n ram dcl))))
```

Overlays Can Be Tolerated

declaration:

```
(A . ("data" A)) (B . ("data" B A))
```




```
collect = ’(3 4 5 3 4 ...)
```

Moral

We should talk about how addresses are *used* globally by a given traversal of the ram, not how addresses are *declared* or how often addresses are *visited*.

Solution to Obstacles 2 and 3

A *memory tagging* is a map from addresses to their *use* as either `:DATA` or `:PTR`.

Def. A tagging is *consistent* with a pointer to a data structure iff every address encountered while traversing the data structure is used as specified in the tagging.

```
(tags-ok-fn fn typ ptr i n ram dcl tags)
```

Key Lemmas

If tagging `tags` is consistent with `ptr2` in `ram` then its consistency with `ptr1` is not affected by `marking` `ptr2`.

```
(defthm tags-ok-fn-mark-fn
 (implies
  (tags-ok-fn fn2 typ2 ptr2 i2 n2 ram dcl tags)
  (equal (tags-ok-fn fn1 typ1 ptr1 i1 n1
                     (mark-fn fn2 typ2 ptr2 i2 n2 ram dcl)
                     dcl tags)
         (tags-ok-fn fn1 typ1 ptr1 i1 n1 ram dcl tags))))
```

If a tagging is consistent with two pointers, then the reachable addresses from one are not changed by marking the other.

```
(defthm collect-fn-mark-fn
 (implies
  (and (tags-ok-fn fn1 typ1 ptr1 i1 n1 ram dcl tags)
       (tags-ok-fn fn2 typ2 ptr2 i2 n2 ram dcl tags))
  (equal
   (collect-fn fn1 typ1 ptr1 i1 n1
               (mark-fn fn2 typ2 ptr2 i2 n2 ram dcl)
               dcl)
   (collect-fn fn1 typ1 ptr1 i1 n1 ram dcl))))
```

Generalized Challenge Theorem 1

```
(defthm g-mark-fn-2
 (implies
  (and
   (not (member addr (collect-fn fn typ ptr i n ram dcl)))
   (tags-ok-fn fn typ ptr i n ram dcl tags))
  (equal (g addr (mark-fn fn typ ptr i n ram dcl))
         (g addr ram))))
```

(Proof by induction, with lemmas above.)

Original Challenge Theorem 1

```
(defthm challenge-theorem-1
 (implies
   (and
    (not (member addr (collect typ ptr n ram dcl)))
    (unique (collect typ ptr n ram dcl)))
   (equal (g addr (mark typ ptr n ram dcl))
          (g addr ram))))
```

Key Idea: Tagging Existence Lemma 1

The uniqueness of reachable addresses imply that there exists a consistent tagging.

Generalized Challenge Theorem 3

```
(defthm mark-fn-mark-fn
  (implies
   (and (tags-ok-fn fn1 typ1 ptr1 i1 n1 ram dcl tags)
        (tags-ok-fn fn2 typ2 ptr2 i2 n2 ram dcl tags)
        (not (intersectp
              (collect-fn fn1 typ1 ptr1 i1 n1 ram dcl)
              (collect-fn fn2 typ2 ptr2 i2 n2 ram dcl))))
   (equal (mark-fn fn1 typ1 ptr1 i1 n1
                   (mark-fn fn2 typ2 ptr2 i2 n2 ram dcl)
                   dcl)
          (mark-fn fn2 typ2 ptr2 i2 n2
                   (mark-fn fn1 typ1 ptr1 i1 n1 ram dcl)
                   dcl))))
```

Original Challenge Theorem 3

```
(defthm challenge-theorem-3
 (implies
   (unique (append (collect typ1 ptr1 n1 ram dcl)
                   (collect typ2 ptr2 n2 ram dcl)))
   (equal
    (mark typ1 ptr1 n1 (mark typ2 ptr2 n2 ram dcl) dcl)
    (mark typ2 ptr2 n2 (mark typ1 ptr1 n1 ram dcl) dcl))))
```

Original Challenge Theorem 3

```
(defthm challenge-theorem-3
 (implies
  (and
   (unique (collect typ1 ptr1 n1 ram dcl))
   (unique (collect typ2 ptr2 n2 ram dcl))
   (not (intersectp
         (collect typ1 ptr1 n1 ram dcl)
         (collect typ2 ptr2 n2 ram dcl))))
   (equal
    (mark typ1 ptr1 n1 (mark typ2 ptr2 n2 ram dcl) dcl)
    (mark typ2 ptr2 n2 (mark typ1 ptr1 n1 ram dcl) dcl))))
```

Generalized Challenge Theorem 3

```
(defthm mark-fn-mark-fn
  (implies
   (and (tags-ok-fn fn1 typ1 ptr1 i1 n1 ram dcl tags)
        (tags-ok-fn fn2 typ2 ptr2 i2 n2 ram dcl tags)
        (not (intersectp
              (collect-fn fn1 typ1 ptr1 i1 n1 ram dcl)
              (collect-fn fn2 typ2 ptr2 i2 n2 ram dcl))))
   (equal (mark-fn fn1 typ1 ptr1 i1 n1
                   (mark-fn fn2 typ2 ptr2 i2 n2 ram dcl)
                   dcl)
          (mark-fn fn2 typ2 ptr2 i2 n2
                   (mark-fn fn1 typ1 ptr1 i1 n1 ram dcl)
                   dcl))))
```

Key Idea: Tagging Existence Lemma 2

If there are taggings for each of two pointers, and the reachable addresses are disjoint, there exists a single tagging for both.