EVENT: Start with the library "c3".

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                              ;;
;;                      %CORRESPONDENCE BETWEEN MG AND P                        ;;
;;                                                                              ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

DEFINITION:
mg-to-p-simple-literal (*lit*)
= **if** int-literalp (*lit*) **then** list ('int, cadr (*lit*))
elseif boolean-literalp (*lit*)
then if cadr (*lit*) = 'false-mg **then** list ('bool, 'f)
else list ('bool, 't) **endif**
elseif character-literalp (*lit*) **then** list ('int, cadr (*lit*))
else 0 **endif**

EVENT: Disable mg-to-p-simple-literal.

```
;; An array value is a list of simple mg literals.  Recall that in the most common
;; case, the values will be reversed because they will be in contiguous locations
;; on the stack, which is indexed from the bottom.
```

DEFINITION:
mg-to-p-simple-literal-list (*lst*)
= **if** *lst* ≃ **nil then nil**
else cons (mg-to-p-simple-literal (car (*lst*)),
mg-to-p-simple-literal-list (cdr (*lst*))) **endif**

THEOREM: mg-to-p-simple-literal-list-plistp
plistp (mg-to-p-simple-literal-list (*x*))

THEOREM: length-mg-to-p-simple-literal-list
length (mg-to-p-simple-literal-list (*x*)) = length (*x*)

DEFINITION:
p-to-mg-simple-literal (*lit*, *typespec*)
= **if** *typespec* = 'int-mg **then** list ('int-mg, cadr (*lit*))
elseif *typespec* = 'boolean-mg
then list ('boolean-mg,
if cadr (*lit*) = 'f **then** 'false-mg
else 'true-mg **endif**)
else list ('character-mg, cadr (*lit*)) **endif**

DEFINITION:
p-to-mg-simple-literal-list (*lst*, *typespec*)
= **if** *lst* ≃ **nil then nil**
else cons (p-to-mg-simple-literal (car (*lst*), *typespec*),
p-to-mg-simple-literal-list (cdr (*lst*), *typespec*)) **endif**

THEOREM: p-to-mg-to-p-simple-literal
simple-typed-literalp (*lit*, *typespec*)
→ (p-to-mg-simple-literal (mg-to-p-simple-literal (*lit*), *typespec*) = *lit*)

EVENT: Disable p-to-mg-to-p-simple-literal.

THEOREM: p-to-mg-to-p-simple-literal-list
simple-typed-literal-plistp (*lst*, *type*)
→ (p-to-mg-simple-literal-list (mg-to-p-simple-literal-list (*lst*), *type*)
= *lst*)

EVENT: Disable p-to-mg-to-p-simple-literal-list.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                            ;;
;;                          %Mapping Conditions                               ;;
;;                                                                            ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; Each MG condition in the state is translated to a natural in the
;; Piton state.  This is done by finding it's index in the cond-list,
;; except for 'normal, 'routineerror, 'leave, and 'timed-out which are computed
;; directly.

;; The assumption in the Piton world is that there is always an operative list of
;; conditions (which is really (append formal-conds local-conds) of the procedure
;; definition.  The Piton condition stored is the index of the MG condition into
;; this list.  However, there are potential conditions which are not in this list also.
;; These are 'timed-out, 'leave, 'routineerror, and 'normal and are computed specially.
;; These result is a Piton natural.
```

DEFINITION:
condition-index (*cond*, *cond-list*)
= **if** *cond* = 'leave **then** 0
elseif *cond* = 'routineerror **then** 1
elseif *cond* = 'normal **then** 2
else 1 + (1 + index (*cond*, *cond-list*)) **endif**

THEOREM: condition-index-append
$(cc \in lst1)$
$\rightarrow$ (condition-index $(cc,$ append $(lst1,\ lst2)) =$ condition-index $(cc,\ lst1)$)

THEOREM: condition-index-append2
$((cc \notin lst1) \wedge (cc \notin$ `'(normal routineerror leave)`$))$
$\rightarrow$ (condition-index $(cc,$ append $(lst1,\ lst2))$
$=$ (length $(lst1) +$ condition-index $(cc,\ lst2)))$

DEFINITION:
mg-cond-to-p-nat $(c,\ cond\text{-}list) =$ list (`'nat`, condition-index $(c,\ cond\text{-}list)$)

DEFINITION:
p-nat-to-mg-cond $(p\text{-}nat,\ cond\text{-}list)$
$=$ **if** $p\text{-}nat =$ `'(nat 0)` **then** `'leave`
elseif $p\text{-}nat =$ `'(nat 1)` **then** `'routineerror`
elseif $p\text{-}nat =$ `'(nat 2)` **then** `'normal`
else car (nth $(cond\text{-}list,$ ((cadr $(p\text{-}nat) - 1) - 1) - 1)$) **endif**

THEOREM: condition-mapping-inverts
ok-cc $(c,\ cond\text{-}list)$
$\rightarrow$ (p-nat-to-mg-cond (mg-cond-to-p-nat $(c,\ cond\text{-}list),\ cond\text{-}list) = c$)

EVENT: Disable condition-mapping-inverts.

EVENT: Disable mg-cond-to-p-nat.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                              ;;
;;                        %Depositing Array Values                              ;;
;;                                                                              ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; The hypothesis will be that each of the locals in bindings contains an address into the
;; temp-stk.  This takes the values of the mg variables and puts them into those locations.

;; Given the values-list for an array, this maps them into successive values in the
;; temp-stk starting at index nat.
```

DEFINITION:
deposit-temp $(val,\ nat,\ temp\text{-}stk) =$ rput $(val,$ untag $(nat),\ temp\text{-}stk)$

DEFINITION:
fetch-temp $(nat,\ temp\text{-}stk) =$ rget (untag $(nat),\ temp\text{-}stk)$

THEOREM: deposit-temp-preserves-length
(untag (*y*) < length (*z*)) → (length (deposit-temp (*x*, *y*, *z*)) = length (*z*))

EVENT: Disable deposit-temp.

EVENT: Disable fetch-temp.

DEFINITION:
fetch-n-temp-stk-elements (*temp-stk*, *nat*, *n*)
= **if** *n* ≃ 0 **then nil**
else cons (fetch-temp (*nat*, *temp-stk*),
fetch-n-temp-stk-elements (*temp-stk*,
add1-nat (*nat*),
n − 1)) **endif**

DEFINITION:
deposit-array-value (*lit-list*, *nat*, *temp-stk*)
= **if** *lit-list* ≃ **nil then** *temp-stk*
else deposit-array-value (cdr (*lit-list*),
add1-nat (*nat*),
deposit-temp (mg-to-p-simple-literal (car (*lit-list*)),
nat,
temp-stk)) **endif**

DEFINITION:
deposit-alist-value (*mg-alist-element*, *bindings*, *temp-stk*)
= **if** simple-mg-type-refp (m-type (*mg-alist-element*))
then deposit-temp (mg-to-p-simple-literal (m-value (*mg-alist-element*)),
cdr (assoc (car (*mg-alist-element*), *bindings*)),
temp-stk)
else deposit-array-value (m-value (*mg-alist-element*),
cdr (assoc (car (*mg-alist-element*),
bindings)),
temp-stk) **endif**

DEFINITION:
map-down-values (*mg-alist*, *bindings*, *temp-stk*)
= **if** *mg-alist* ≃ **nil then** *temp-stk*
else map-down-values (cdr (*mg-alist*),
bindings,
deposit-alist-value (car (*mg-alist*),
bindings,
temp-stk)) **endif**

THEOREM: map-down-values-distributes
map-down-values (append (*lst1*, *lst2*), *bindings*, *temp-stk*)
= map-down-values (*lst2*, *bindings*, map-down-values (*lst1*, *bindings*, *temp-stk*))

EVENT: Disable map-down-values-distributes.

THEOREM: deposit-array-value-preserves-plistp
plistp (*temp-stk*) → plistp (deposit-array-value (*lst*, *nat*, *temp-stk*))

EVENT: Disable deposit-array-value-preserves-plistp.

THEOREM: map-down-values-preserves-plistp
plistp (*temp-stk*) → plistp (map-down-values (*alist*, *bindings*, *temp-stk*))

THEOREM: extra-bindings-dont-affect-deposit-alist-value
(car (*x*) ∉ listcars (*bindings1*))
→ (deposit-alist-value (*x*, append (*bindings1*, *bindings2*), *temp-stk*)
= deposit-alist-value (*x*, *bindings2*, *temp-stk*))

THEOREM: deposit-alist-value-not-affected-by-extra-element
(car (*y*) ≠ car (*x*))
→ (deposit-alist-value (*y*, append (*lst1*, cons (*x*, *lst2*)), *temp-stk*)
= deposit-alist-value (*y*, append (*lst1*, *lst2*), *temp-stk*))

THEOREM: deposit-alist-value-not-affected-by-extra-element2
(car (*y*) ≠ car (*x*))
→ (deposit-alist-value (*y*, cons (*x*, *lst2*), *temp-stk*)
= deposit-alist-value (*y*, *lst2*, *temp-stk*))

THEOREM: map-down-values-not-affected-by-extra-binding
(car (*x*) ∉ listcars (*lst*))
→ (map-down-values (*lst*, append (*lst1*, cons (*x*, *lst2*)), *temp-stk*)
= map-down-values (*lst*, append (*lst1*, *lst2*), *temp-stk*))

THEOREM: new-binding-doesnt-disturb-map-down-values
(car (*x*) ∉ listcars (*lst1*))
→ (map-down-values (*lst1*, cons (*x*, *bindings*), *temp-stk*)
= map-down-values (*lst1*, *bindings*, *temp-stk*))

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                      ;;
;;                         %Ok-Temp-Stk-Index                           ;;
;;                                                                      ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

DEFINITION:
ok-temp-stk-index (nat, $temp\text{-}stk$)
= (length-plistp (nat, 2)
 ∧ (type (nat) = `'nat`)
 ∧ (untag (nat) $\in \mathbf{N}$)
 ∧ (untag (nat) < length ($temp\text{-}stk$)))

EVENT: Disable ok-temp-stk-index.

```
;; This in the hyps says that each of the piton local names in the piton bindings
;; points to a slot in temp-stk.  It DOESNT say that the value is the right one.
;; The function map-down-values puts the right value at that spot.

;; This needs to also talk about the array case; the length of the temp-stk should be
;; such that not only is the temp-stk-index in the right range, but so are all of the
;; indices of the array elements.
```

DEFINITION:
ok-temp-stk-array-index (nat, $temp\text{-}stk$, $lngth$)
= (length-plistp (nat, 2)
 ∧ (type (nat) = `'nat`)
 ∧ (untag (nat) $\in \mathbf{N}$)
 ∧ ((untag (nat) + ($lngth$ − 1)) < length ($temp\text{-}stk$)))

EVENT: Disable ok-temp-stk-array-index.

THEOREM: ok-temp-stk-index-sensitive-to-length
(ok-temp-stk-index (x, $temp\text{-}stk1$)
 ∧ (length ($temp\text{-}stk2$) $\not<$ length ($temp\text{-}stk1$)))
→ ok-temp-stk-index (x, $temp\text{-}stk2$)

EVENT: Disable ok-temp-stk-index-sensitive-to-length.

THEOREM: ok-temp-stk-array-index-sensitive-to-length
(ok-temp-stk-array-index (x, $temp\text{-}stk1$, n)
 ∧ (length ($temp\text{-}stk2$) $\not<$ length ($temp\text{-}stk1$)))
→ ok-temp-stk-array-index (x, $temp\text{-}stk2$, n)

EVENT: Disable ok-temp-stk-array-index-sensitive-to-length.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                            ;;
```

```
;;              %Operations Preserve Temp-Stk Length                    ;;
;;                                                                      ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

THEOREM: deposit-temp-never-shrinks
$(\text{length}\,(\text{deposit-temp}\,(x,\ y,\ z)) < \text{length}\,(z)) = \mathbf{f}$

EVENT: Disable deposit-temp-never-shrinks.

THEOREM: lessp-transitive2
$((x \not< y) \wedge (y \not< z)) \rightarrow ((x < z) = \mathbf{f})$

EVENT: Disable lessp-transitive2.

THEOREM: deposit-array-value-never-shrinks
$(\text{length}\,(\text{deposit-array-value}\,(x,\ y,\ z)) < \text{length}\,(z)) = \mathbf{f}$

EVENT: Disable deposit-array-value-never-shrinks.

THEOREM: deposit-alist-value-never-shrinks
$(\text{length}\,(\text{deposit-alist-value}\,(y,\ \textit{bindings},\ \textit{temp-stk})) < \text{length}\,(\textit{temp-stk}))$
$= \quad \mathbf{f}$

EVENT: Disable deposit-alist-value-never-shrinks.

EVENT: Disable deposit-alist-value.

THEOREM: deposit-alist-value-never-shrinks2
$(u < \text{length}\,(\textit{temp-stk}))$
$\rightarrow \quad ((u < \text{length}\,(\text{deposit-alist-value}\,(y,\ \textit{bindings},\ \textit{temp-stk}))) = \mathbf{t})$

EVENT: Disable deposit-alist-value-never-shrinks2.

THEOREM: map-down-values-never-shrinks
$(\text{length}\,(\text{map-down-values}\,(y,\ \textit{bindings},\ \textit{temp-stk})) < \text{length}\,(\textit{temp-stk})) = \mathbf{f}$

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                      ;;
;;                      %Mg-vars-ok-in-p-state                          ;;
;;                                                                      ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

DEFINITION:
mg-var-ok-in-p-state (*mg-var*, *bindings*, *temp-stk*)
= (definedp (car (*mg-var*), *bindings*)
∧ **if** simple-mg-type-refp (m-type (*mg-var*))
then ok-temp-stk-index (cdr (assoc (car (*mg-var*), *bindings*)),
temp-stk)
else ok-temp-stk-array-index (cdr (assoc (car (*mg-var*),
bindings)),
temp-stk,
array-length (m-type (*mg-var*))) **endif**)

DEFINITION:
mg-vars-list-ok-in-p-state (*mg-vars*, *bindings*, *temp-stk*)
= **if** *mg-vars* ≃ **nil then t**
else mg-var-ok-in-p-state (car (*mg-vars*), *bindings*, *temp-stk*)
∧ mg-vars-list-ok-in-p-state (cdr (*mg-vars*),
bindings,
temp-stk) **endif**

THEOREM: mg-vars-list-ok-in-p-state-cdr
(mg-vars-list-ok-in-p-state (*x*, *y*, *z*) ∧ listp (*x*))
→ mg-vars-list-ok-in-p-state (cdr (*x*), *y*, *z*)

THEOREM: mg-var-ok-temp-stk-index
(mg-vars-list-ok-in-p-state (*lst*, *bindings*, *temp-stk*) ∧ definedp (*b*, *lst*))
→ ((untag (value (*b*, *bindings*)) < length (*temp-stk*)) = **t**)

EVENT: Disable mg-var-ok-temp-stk-index.

THEOREM: mg-vars-list-ok-in-p-state-distributes
mg-vars-list-ok-in-p-state (append (*lst1*, *lst2*), *bindings*, *temp-stk*)
= (mg-vars-list-ok-in-p-state (*lst1*, *bindings*, *temp-stk*)
∧ mg-vars-list-ok-in-p-state (*lst2*, *bindings*, *temp-stk*))

THEOREM: mg-vars-list-ok-reorder-cars
mg-vars-list-ok-in-p-state (cons (*x*, cons (*y*, *lst*)), *bindings*, *temp-stk*)
→ mg-vars-list-ok-in-p-state (cons (*y*, cons (*x*, *lst*)), *bindings*, *temp-stk*)

THEOREM: mg-vars-list-ok-strip-off-car
(car (*x*) ∉ listcars (*lst1*))
→ (mg-vars-list-ok-in-p-state (*lst1*, cons (*x*, *lst2*), *temp-stk*)
= mg-vars-list-ok-in-p-state (*lst1*, *lst2*, *temp-stk*))

EVENT: Disable mg-vars-list-ok-strip-off-car.

THEOREM: mg-vars-list-ok-sensitive-to-temp-stk-length2
((length (*temp-stk2*) $\not<$ length (*temp-stk1*))
 $\wedge$ mg-vars-list-ok-in-p-state (*alist*, *bindings*, *temp-stk1*))
$\rightarrow$ mg-vars-list-ok-in-p-state (*alist*, *bindings*, *temp-stk2*)

EVENT: Disable mg-vars-list-ok-sensitive-to-temp-stk-length2.

THEOREM: append-bindings-doesnt-affect-mg-vars-list-ok
mg-vars-list-ok-in-p-state (*alist*, *bindings*, *temp-stk*)
$\rightarrow$ mg-vars-list-ok-in-p-state (*alist*, append (*bindings*, *lst*), *temp-stk*)

EVENT: Disable append-bindings-doesnt-affect-mg-vars-list-ok.

THEOREM: append-bindings-left-doesnt-affect-mg-vars-list-ok
(mg-vars-list-ok-in-p-state (*alist*, *bindings2*, *temp-stk*)
 $\wedge$ all-cars-unique (append (*bindings1*, *bindings2*)))
$\rightarrow$ mg-vars-list-ok-in-p-state (*alist*, append (*bindings1*, *bindings2*), *temp-stk*)

THEOREM: mg-vars-list-members-ok-vars
(mg-vars-list-ok-in-p-state (*alist*, *bindings*, *temp-stk*) $\wedge$ ($x \in$ *alist*))
$\rightarrow$ mg-var-ok-in-p-state (x, *bindings*, *temp-stk*)

THEOREM: mg-var-ok-value-lessp-length-temp-stk
(mg-vars-list-ok-in-p-state (*lst*, *bindings*, *temp-stk*) $\wedge$ definedp (x, *lst*))
$\rightarrow$ ((untag (cdr (assoc (x, *bindings*))) $<$ length (*temp-stk*)) = **t**)

EVENT: Disable mg-var-ok-value-lessp-length-temp-stk.

THEOREM: mg-var-ok-untag-value-numberp
(mg-vars-list-ok-in-p-state (*lst*, *bindings*, *temp-stk*) $\wedge$ definedp (x, *lst*))
$\rightarrow$ (untag (cdr (assoc (x, *bindings*))) $\in$ **N**)

EVENT: Disable mg-var-ok-untag-value-numberp.

THEOREM: mg-var-ok-array-index-ok
(mg-vars-list-ok-in-p-state (*lst*, *bindings*, *temp-stk*)
 $\wedge$ mg-alistp (*lst*)
 $\wedge$ ($y \in$ *lst*)
 $\wedge$ ($\neg$ simple-mg-type-refp (m-type (y))))
$\rightarrow$ (((untag (cdr (assoc (car (y), *bindings*))) + (length (caddr (y)) $-$ 1))
  $<$ length (*temp-stk*))
 = **t**)

EVENT: Disable mg-var-ok-array-index-ok.

THEOREM: mg-var-ok-array-index-ok2
(mg-vars-list-ok-in-p-state (*lst*, *bindings*, *temp-stk*)
 ∧ mg-alistp (*lst*)
 ∧ (*y* ∈ *lst*)
 ∧ (¬ simple-mg-type-refp (m-type (*y*))))
→ (((untag (cdr (assoc (car (*y*), *bindings*))) + (array-length (cadr (*y*)) − 1))
 < length (*temp-stk*))
 = **t**)

EVENT: Disable mg-var-ok-array-index-ok2.

THEOREM: deposit-alist-value-preserves-mg-vars-list-ok
mg-vars-list-ok-in-p-state (cons (*x*, *lst*), *bindings*, *temp-stk*)
→ mg-vars-list-ok-in-p-state (*lst*,
 bindings,
 deposit-alist-value (*x*, *bindings*, *temp-stk*))

THEOREM: deposit-array-value-preserves-length
((untag (*nat*) + (length (*value*) − 1)) < length (*temp-stk*))
→ (length (deposit-array-value (*value*, *nat*, *temp-stk*)) = length (*temp-stk*))

THEOREM: member-array-lengths-match
(mg-alistp (*lst*) ∧ (*x* ∈ *lst*) ∧ (¬ simple-mg-type-refp (m-type (*x*))))
→ (length (caddr (*x*)) = array-length (cadr (*x*)))

THEOREM: array-index-lessp
(mg-alistp (*lst*)
 ∧ (*x* ∈ *lst*)
 ∧ mg-vars-list-ok-in-p-state (*lst*, *bindings*, *temp-stk*)
 ∧ (¬ simple-mg-type-refp (m-type (*x*))))
→ (((untag (cdr (assoc (car (*x*), *bindings*))) + (length (caddr (*x*)) − 1))
 < length (*temp-stk*))
 = **t**)

THEOREM: deposit-alist-value-preserves-length
(mg-alistp (*lst*)
 ∧ (*x* ∈ *lst*)
 ∧ mg-vars-list-ok-in-p-state (*lst*, *bindings*, *temp-stk*))
→ (length (deposit-alist-value (*x*, *bindings*, *temp-stk*))
 = length (*temp-stk*))

THEOREM: map-down-values-preserves-length
(mg-vars-list-ok-in-p-state (*x*, *y*, *temp-stk*) ∧ mg-alistp (*x*))
→ (length (map-down-values (*x*, *y*, *temp-stk*)) = length (*temp-stk*))

THEOREM: deposit-temp-append1
((untag (*nat*) ∈ **N**) ∧ (untag (*nat*) < length (*temp-stk*)))
→ (deposit-temp (*x*, *nat*, append (*lst*, *temp-stk*))
= append (*lst*, deposit-temp (*x*, *nat*, *temp-stk*)))

EVENT: Disable deposit-temp-append1.

THEOREM: deposit-temp-append
((*x* ∈ *vars*)
∧ simple-mg-type-refp (cadr (*x*))
∧ mg-vars-list-ok-in-p-state (*vars*, *bindings*, *temp-stk*))
→ (deposit-temp (*value*, cdr (assoc (car (*x*), *bindings*)), append (*lst*, *temp-stk*))
= append (*lst*,
deposit-temp (*value*,
cdr (assoc (car (*x*), *bindings*)),
temp-stk)))

EVENT: Disable deposit-temp-append.

THEOREM: map-down-values-doesnt-affect-mg-vars-list-ok
(mg-alistp (*u*)
∧ mg-alistp (*x*)
∧ mg-vars-list-ok-in-p-state (*u*, *v*, *temp-stk*)
∧ mg-vars-list-ok-in-p-state (*x*, *y*, append (*lst1*, *temp-stk*)))
→ mg-vars-list-ok-in-p-state (*x*,
y,
append (*lst1*, map-down-values (*u*, *v*, *temp-stk*)))

THEOREM: signatures-match-preserves-mg-vars-list-ok
(signatures-match (*x*, *y*) ∧ mg-vars-list-ok-in-p-state (*x*, *z*, *w*))
→ mg-vars-list-ok-in-p-state (*y*, *z*, *w*)

EVENT: Disable signatures-match-preserves-mg-vars-list-ok.

THEOREM: ok-temp-stk-simple-member
(simple-mg-type-refp (cadr (assoc (*x*, *mg-vars*)))
∧ mg-vars-list-ok-in-p-state (*mg-vars*, *bindings*, *temp-stk*))
→ ok-temp-stk-index (cdr (assoc (*x*, *bindings*)), *temp-stk*)

THEOREM: ok-temp-stk-index-actual
(listp (*formals*)
∧ plistp (*mg-vars*)
∧ ok-actual-params-list (*actuals*, *mg-vars*)

$\wedge$ data-param-lists-match (*actuals*, *formals*, *mg-vars*)
$\wedge$ ok-mg-formal-data-params-plistp (*formals*)
$\wedge$ mg-vars-list-ok-in-p-state (*mg-vars*, *bindings*, *temp-stk*)
$\wedge$ simple-mg-type-refp (cadar (*formals*)))
$\rightarrow$ ok-temp-stk-index (cdr (assoc (car (*actuals*), *bindings*)), *temp-stk*)

THEOREM: ok-temp-stk-array-simple-member
(array-mg-type-refp (cadr (assoc (x, *mg-vars*)))
$\wedge$ mg-vars-list-ok-in-p-state (*mg-vars*, *bindings*, *temp-stk*))
$\rightarrow$ ok-temp-stk-array-index (cdr (assoc (x, *bindings*)),
temp-stk,
array-length (cadr (assoc (x, *mg-vars*))))

THEOREM: ok-temp-stk-index-array-actual
(listp (*formals*)
$\wedge$ plistp (*mg-vars*)
$\wedge$ ok-actual-params-list (*actuals*, *mg-vars*)
$\wedge$ data-param-lists-match (*actuals*, *formals*, *mg-vars*)
$\wedge$ ok-mg-formal-data-params-plistp (*formals*)
$\wedge$ mg-vars-list-ok-in-p-state (*mg-vars*, *bindings*, *temp-stk*)
$\wedge$ ($\neg$ simple-mg-type-refp (cadar (*formals*))))
$\rightarrow$ ok-temp-stk-array-index (cdr (assoc (car (*actuals*), *bindings*)),
temp-stk,
array-length (cadar (*formals*)))

```
;; %mg-vars-list-ok-in-p-state
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                            ;;
;;                        %N-Successive-Pointers                              ;;
;;                                                                            ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

DEFINITION:
n-successive-pointers (*nat*, n)
= **if** $n \simeq$ `0` **then nil**
else cons (untag (*nat*), n-successive-pointers (add1-nat (*nat*), $n - 1$)) **endif**

THEOREM: n-successive-pointers-plistp
plistp (n-successive-pointers (*nat*, n))

THEOREM: lesser-number-not-member-n-successive-pointers
($i <$ untag (*nat*)) $\rightarrow$ ($i \notin$ n-successive-pointers (*nat*, n))

THEOREM: no-duplicates-successive-pointers
(untag (*nat*) $\in \mathbf{N}$) $\rightarrow$ no-duplicates (n-successive-pointers (*nat*, n))

THEOREM: car-member-n-successive-pointers
$(n \not\simeq 0) \rightarrow (\text{untag}\,(nat) \in \text{n-successive-pointers}\,(nat,\ n))$

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                            ;;
;;                          %Collect-Pointers                                 ;;
;;                                                                            ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

DEFINITION:
collect-pointers (*bindings*, *alist*)
= **if** *alist* $\simeq$ **nil then nil**
elseif simple-mg-type-refp (cadr (car (*alist*)))
then cons (untag (cdr (assoc (car (car (*alist*)), *bindings*))),
collect-pointers (*bindings*, cdr (*alist*)))
else append (n-successive-pointers (cdr (assoc (car (car (*alist*)),
bindings)),
array-length (cadr (car (*alist*)))),
collect-pointers (*bindings*, cdr (*alist*))) **endif**

THEOREM: collect-pointers-plistp
plistp (collect-pointers (*bindings*, *alist*))

THEOREM: collect-pointers-distributes
collect-pointers (*bindings*, append (*lst1*, *lst2*))
= append (collect-pointers (*bindings*, *lst1*),
collect-pointers (*bindings*, *lst2*))

THEOREM: signatures-match-preserves-collect-pointers
signatures-match (*alist2*, *alist1*)
→ (collect-pointers (*bindings*, *alist1*)
= collect-pointers (*bindings*, *alist2*))

EVENT: Disable signatures-match-preserves-collect-pointers.

THEOREM: extra-bindings-dont-affect-collect-pointers
no-duplicates (listcars (append (*bindings1*, *lst*)))
→ (collect-pointers (append (*bindings1*, *bindings2*), *lst*)
= collect-pointers (*bindings2*, *lst*))

EVENT: Disable extra-bindings-dont-affect-collect-pointers.

THEOREM: extra-bindings-dont-affect-collect-pointers2

no-duplicates (listcars (append (*bindings2*, *lst*)))
→ (collect-pointers (append (*bindings1*, *bindings2*), *lst*)
= collect-pointers (*bindings1*, *lst*))

EVENT: Disable extra-bindings-dont-affect-collect-pointers2.

THEOREM: extra-binding-doesnt-affect-collect-pointers
($x \notin$ listcars (*lst*))
→ (collect-pointers (cons (cons (x, y), *bindings*), *lst*)
= collect-pointers (*bindings*, *lst*))

EVENT: Disable extra-binding-doesnt-affect-collect-pointers.

THEOREM: defined-array-pointers-subset-collect-pointers
(definedp (x, *lst*)
∧ (¬ simple-mg-type-refp (cadr (assoc (x, *lst*))))
∧ mg-alistp (*lst*))
→ subset (n-successive-pointers (cdr (assoc (x, *bindings*)),
length (caddr (assoc (x, *lst*)))),
collect-pointers (*bindings*, *lst*))

EVENT: Disable defined-array-pointers-subset-collect-pointers.

```
;; %collect-pointers
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                            ;;
;;                            %No-P-Aliasing                                  ;;
;;                                                                            ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; This says that none of the pointers in the variable list point to the same location.
;; This is much stricter than the Gypsy aliasing requirement since it doesn't make any
;; distinction between const and var parameters.  It is consistent with the restriction
;; made in mg-meaning, but both need to be relaxed.
```

DEFINITION:
no-p-aliasing (*bindings*, *alist*)
= no-duplicates (collect-pointers (*bindings*, *alist*))

THEOREM: no-p-aliasing-cdr
(listp (*lst*) ∧ no-p-aliasing (*bindings*, *lst*))
→ no-p-aliasing (*bindings*, cdr (*lst*))

THEOREM: no-p-aliasing-cdr2
no-p-aliasing (*bindings*, cons (*x*, *lst*)) → no-p-aliasing (*bindings*, *lst*)

THEOREM: no-p-aliasing-append-commutes
no-p-aliasing (*bindings*, append (*x*, *y*)) → no-p-aliasing (*bindings*, append (*y*, *x*))

EVENT: Disable no-p-aliasing-append-commutes.

THEOREM: cons-car-append-no-p-aliasing
(listp (*x*) ∧ no-p-aliasing (*bindings*, append (*x*, *y*)))
→ no-p-aliasing (*bindings*, cons (car (*x*), *y*))

EVENT: Disable cons-car-append-no-p-aliasing.

THEOREM: no-p-aliasing-cdr-append
(listp (*y*) ∧ no-p-aliasing (*bindings*, append (*x*, *y*)))
→ no-p-aliasing (*bindings*, append (*x*, cdr (*y*)))

EVENT: Disable no-p-aliasing-cdr-append.

THEOREM: no-p-aliasing-cons-cdr
no-p-aliasing (*bindings*, cons (*x*, cons (*y*, *lst*)))
→ no-p-aliasing (*bindings*, cons (*x*, *lst*))

EVENT: Disable no-p-aliasing-cons-cdr.

THEOREM: signatures-match-preserves-no-p-aliasing
(signatures-match (*alist1*, *alist2*) ∧ no-p-aliasing (*bindings*, *alist1*))
→ no-p-aliasing (*bindings*, *alist2*)

EVENT: Disable signatures-match-preserves-no-p-aliasing.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                              ;;
;;           %Distinct Variables Have Distinct Pointers                          ;;
;;                                                                              ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

THEOREM: member-pointer-collect-pointers
(definedp (*x*, *alist*) ∧ mg-name-alistp (*alist*))
→ (untag (cdr (assoc (*x*, *bindings*))) ∈ collect-pointers (*bindings*, *alist*))

THEOREM: no-p-aliasing-distinct-variables-base-case0
$$
\begin{array}{ll}
(\text{listp}\,(\mathit{alist}) & \\
\wedge & (x = \text{caar}\,(\mathit{alist})) \\
\wedge & (\text{untag}\,(\text{cdr}\,(\text{assoc}\,(x,\ \mathit{bindings}))) = \text{untag}\,(\text{cdr}\,(\text{assoc}\,(y,\ \mathit{bindings})))) \\
\wedge & \text{mg-name-alistp}\,(\mathit{alist}) \\
\wedge & \text{definedp}\,(y,\ \mathit{alist}) \\
\wedge & (x \neq y)) \\
\rightarrow & (\neg\ \text{no-duplicates}\,(\text{collect-pointers}\,(\mathit{bindings},\ \mathit{alist})))
\end{array}
$$

EVENT: Disable no-p-aliasing-distinct-variables-base-case0.

THEOREM: no-p-aliasing-distinct-variables-base-case
$$
\begin{array}{ll}
(\text{listp}\,(\mathit{alist}) & \\
\wedge & (x = \text{caar}\,(\mathit{alist})) \\
\wedge & \text{no-p-aliasing}\,(\mathit{bindings},\ \mathit{alist}) \\
\wedge & \text{mg-name-alistp}\,(\mathit{alist}) \\
\wedge & \text{definedp}\,(x,\ \mathit{alist}) \\
\wedge & \text{definedp}\,(y,\ \mathit{alist}) \\
\wedge & (x \neq y)) \\
\rightarrow & (\text{untag}\,(\text{cdr}\,(\text{assoc}\,(x,\ \mathit{bindings}))) \neq \text{untag}\,(\text{cdr}\,(\text{assoc}\,(y,\ \mathit{bindings}))))
\end{array}
$$

EVENT: Disable no-p-aliasing-distinct-variables-base-case.

THEOREM: no-p-aliasing-distinct-variables
$$
\begin{array}{ll}
(\text{no-p-aliasing}\,(\mathit{bindings},\ \mathit{alist}) & \\
\wedge & \text{mg-name-alistp}\,(\mathit{alist}) \\
\wedge & \text{definedp}\,(x,\ \mathit{alist}) \\
\wedge & \text{definedp}\,(y,\ \mathit{alist}) \\
\wedge & (x \neq y)) \\
\rightarrow & (\text{untag}\,(\text{cdr}\,(\text{assoc}\,(x,\ \mathit{bindings}))) \neq \text{untag}\,(\text{cdr}\,(\text{assoc}\,(y,\ \mathit{bindings}))))
\end{array}
$$

THEOREM: distinct-variables-lemma2-base-case
$$
\begin{array}{ll}
(\text{listp}\,(\mathit{lst}) & \\
\wedge & (x = \text{caar}\,(\mathit{lst})) \\
\wedge & (\text{untag}\,(\text{cdr}\,(\text{assoc}\,(x,\ \mathit{bindings}))) \\
 & \quad \in \quad \text{n-successive-pointers}\,(\text{cdr}\,(\text{assoc}\,(y,\ \mathit{bindings})), \\
 & \qquad\qquad\qquad\qquad \text{length}\,(\text{caddr}\,(\text{assoc}\,(y,\ \mathit{lst}))))) \\
\wedge & \text{mg-alistp}\,(\mathit{lst}) \\
\wedge & \text{definedp}\,(x,\ \mathit{lst}) \\
\wedge & \text{definedp}\,(y,\ \mathit{lst}) \\
\wedge & (x \neq y) \\
\wedge & (\neg\ \text{simple-mg-type-refp}\,(\text{cadr}\,(\text{assoc}\,(y,\ \mathit{lst})))) \\
\rightarrow & (\neg\ \text{no-p-aliasing}\,(\mathit{bindings},\ \mathit{lst}))
\end{array}
$$

EVENT: Disable distinct-variables-lemma2-base-case.

THEOREM: distinct-variables-lemma2

```
(no-p-aliasing (bindings, lst)
 ∧   mg-alistp (lst)
 ∧   definedp (x, lst)
 ∧   definedp (y, lst)
 ∧   (x ≠ y)
 ∧   (¬ simple-mg-type-refp (cadr (assoc (y, lst)))))
 →   (untag (cdr (assoc (x, bindings)))
      ∉   n-successive-pointers (cdr (assoc (y, bindings)),
                                 length (caddr (assoc (y, lst)))))
```

THEOREM: distinct-array-values-one-way-disjoint

```
(mg-vars-list-ok-in-p-state (lst, bindings, temp-stk)
 ∧   no-p-aliasing (bindings, lst)
 ∧   mg-alistp (lst)
 ∧   all-cars-unique (lst)
 ∧   definedp (x, lst)
 ∧   definedp (y, lst)
 ∧   (x ≠ y)
 ∧   (¬ simple-mg-type-refp (cadr (assoc (x, lst))))
 ∧   (¬ simple-mg-type-refp (cadr (assoc (y, lst)))))
 →   one-way-disjoint (n-successive-pointers (cdr (assoc (x, bindings)),
                                              length (caddr (assoc (x, lst)))),
                       n-successive-pointers (cdr (assoc (y, bindings)),
                                              length (caddr (assoc (y, lst))))))
```

EVENT: Disable distinct-array-values-one-way-disjoint.

THEOREM: distinct-array-values-disjoint

```
(mg-vars-list-ok-in-p-state (lst, bindings, temp-stk)
 ∧   no-p-aliasing (bindings, lst)
 ∧   mg-alistp (lst)
 ∧   all-cars-unique (lst)
 ∧   definedp (x, lst)
 ∧   definedp (y, lst)
 ∧   (x ≠ y)
 ∧   (¬ simple-mg-type-refp (cadr (assoc (x, lst))))
 ∧   (¬ simple-mg-type-refp (cadr (assoc (y, lst)))))
 →   disjoint (n-successive-pointers (cdr (assoc (x, bindings)),
                                      array-length (cadr (assoc (x, lst)))),
               n-successive-pointers (cdr (assoc (y, bindings)),
                                      array-length (cadr (assoc (y, lst))))))
```

```
;; %distinct variables
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                            ;;
;;                          %Deposits Commutes                                ;;
;;                                                                            ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

THEOREM: deposit-array-value-append1
((untag (*nat*) ∈ **N**)
 ∧ ((untag (*nat*) + (length (*value*) − 1)) < length (*temp-stk*)))
→ (deposit-array-value (*value*, *nat*, append (*lst*, *temp-stk*))
 = append (*lst*, deposit-array-value (*value*, *nat*, *temp-stk*)))

EVENT: Disable deposit-array-value-append1.

THEOREM: deposit-array-value-append
((*x* ∈ *vars*)
 ∧ mg-alistp (*vars*)
 ∧ mg-vars-list-ok-in-p-state (*vars*, *bindings*, *temp-stk*)
 ∧ (¬ simple-mg-type-refp (cadr (*x*))))
→ (deposit-array-value (caddr (*x*),
 cdr (assoc (car (*x*), *bindings*)),
 append (*lst*, *temp-stk*))
 = append (*lst*,
 deposit-array-value (caddr (*x*),
 cdr (assoc (car (*x*), *bindings*)),
 temp-stk)))

EVENT: Disable deposit-array-value-append.

THEOREM: deposit-alist-value-append
((*x* ∈ *vars*)
 ∧ mg-alistp (*vars*)
 ∧ mg-vars-list-ok-in-p-state (*vars*, *bindings*, *temp-stk*))
→ (deposit-alist-value (*x*, *bindings*, append (*lst*, *temp-stk*))
 = append (*lst*, deposit-alist-value (*x*, *bindings*, *temp-stk*)))

THEOREM: deposit-temp-commutes
((untag (*y*) ≠ untag (*u*))
 ∧ (untag (*y*) ∈ **N**)
 ∧ (untag (*u*) ∈ **N**)
 ∧ (untag (*u*) < length (*v*))
 ∧ (untag (*y*) < length (*v*)))

$\rightarrow$ (deposit-temp (x, y, deposit-temp (z, u, v))
= deposit-temp (z, u, deposit-temp (x, y, v)))

EVENT: Disable deposit-temp-commutes.

THEOREM: deposit-temp-commutes0
(mg-vars-list-ok-in-p-state (*lst*, *bindings*, *temp-stk*)
$\wedge$ no-p-aliasing (*bindings*, *lst*)
$\wedge$ mg-alistp (*lst*)
$\wedge$ all-cars-unique (*lst*)
$\wedge$ ($x \in$ *lst*)
$\wedge$ ($y \in$ *lst*)
$\wedge$ (car (x) $\neq$ car (y)))
$\rightarrow$ (deposit-temp (*x-value*,
cdr (assoc (car (x), *bindings*)),
deposit-temp (*y-value*,
cdr (assoc (car (y), *bindings*)),
temp-stk))
= deposit-temp (*y-value*,
cdr (assoc (car (y), *bindings*)),
deposit-temp (*x-value*,
cdr (assoc (car (x), *bindings*)),
temp-stk)))

EVENT: Disable deposit-temp-commutes0.

THEOREM: deposit-temp-deposit-array-value-commute
((untag (*nat1*) $\notin$ n-successive-pointers (*nat2*, length (*value*)))
$\wedge$ (untag (*nat1*) $\in \mathbf{N}$)
$\wedge$ (untag (*nat2*) $\in \mathbf{N}$)
$\wedge$ (untag (*nat1*) $<$ length (*temp-stk*))
$\wedge$ ((untag (*nat2*) + (length (*value*) $-$ 1)) $<$ length (*temp-stk*)))
$\rightarrow$ (deposit-temp (z, *nat1*, deposit-array-value (*value*, *nat2*, *temp-stk*))
= deposit-array-value (*value*, *nat2*, deposit-temp (z, *nat1*, *temp-stk*)))

EVENT: Disable deposit-temp-deposit-array-value-commute.

THEOREM: deposit-temp-deposit-array-value-commute2
(mg-vars-list-ok-in-p-state (*lst*, *bindings*, *temp-stk*)
$\wedge$ no-p-aliasing (*bindings*, *lst*)
$\wedge$ mg-alistp (*lst*)
$\wedge$ all-cars-unique (*lst*)
$\wedge$ ($x \in$ *lst*)

$$
\begin{array}{ll}
\wedge & (y \in \mathit{lst}) \\
\wedge & (\mathrm{car}\,(x) \neq \mathrm{car}\,(y)) \\
\wedge & (\neg\ \text{simple-mg-type-refp}\,(\mathrm{cadr}\,(y))) \\
\wedge & (\mathrm{length}\,(\mathit{y\text{-}value}) = \text{array-length}\,(\mathrm{cadr}\,(y)))) \\
\rightarrow & (\text{deposit-temp}\,(z, \\
 & \qquad \mathrm{cdr}\,(\mathrm{assoc}\,(\mathrm{car}\,(x),\ \mathit{bindings})), \\
 & \qquad \text{deposit-array-value}\,(\mathit{y\text{-}value}, \\
 & \qquad\qquad \mathrm{cdr}\,(\mathrm{assoc}\,(\mathrm{car}\,(y),\ \mathit{bindings})), \\
 & \qquad\qquad \mathit{temp\text{-}stk})) \\
 & = \quad \text{deposit-array-value}\,(\mathit{y\text{-}value}, \\
 & \qquad\qquad \mathrm{cdr}\,(\mathrm{assoc}\,(\mathrm{car}\,(y),\ \mathit{bindings})), \\
 & \qquad\qquad \text{deposit-temp}\,(z, \\
 & \qquad\qquad\qquad \mathrm{cdr}\,(\mathrm{assoc}\,(\mathrm{car}\,(x),\ \mathit{bindings})), \\
 & \qquad\qquad\qquad \mathit{temp\text{-}stk})))
\end{array}
$$

EVENT: Disable deposit-temp-deposit-array-value-commute2.

THEOREM: one-way-disjoint-non-member-n-successive-pointers

$$
\begin{array}{ll}
 & (\text{one-way-disjoint}\,(\text{n-successive-pointers}\,(\mathit{x\text{-}nat},\ n), \\
 & \qquad\qquad \text{n-successive-pointers}\,(\mathit{y\text{-}nat},\ m)) \\
\wedge & (n \not\simeq \mathtt{0})) \\
\rightarrow & (\mathrm{untag}\,(\mathit{x\text{-}nat}) \notin \text{n-successive-pointers}\,(\mathit{y\text{-}nat},\ m))
\end{array}
$$

THEOREM: deposit-array-value-commutes

$$
\begin{array}{ll}
 & ((\mathrm{untag}\,(\mathit{x\text{-}nat}) \in \mathbf{N}) \\
\wedge & (\mathrm{untag}\,(\mathit{y\text{-}nat}) \in \mathbf{N}) \\
\wedge & ((\mathrm{untag}\,(\mathit{x\text{-}nat}) + (\mathrm{length}\,(\mathit{x\text{-}value}) - 1)) < \mathrm{length}\,(\mathit{temp\text{-}stk})) \\
\wedge & ((\mathrm{untag}\,(\mathit{y\text{-}nat}) + (\mathrm{length}\,(\mathit{y\text{-}value}) - 1)) < \mathrm{length}\,(\mathit{temp\text{-}stk})) \\
\wedge & \text{disjoint}\,(\text{n-successive-pointers}\,(\mathit{x\text{-}nat},\ \mathrm{length}\,(\mathit{x\text{-}value})), \\
 & \qquad\qquad \text{n-successive-pointers}\,(\mathit{y\text{-}nat},\ \mathrm{length}\,(\mathit{y\text{-}value})))) \\
\rightarrow & (\text{deposit-array-value}\,(\mathit{x\text{-}value}, \\
 & \qquad\qquad \mathit{x\text{-}nat}, \\
 & \qquad\qquad \text{deposit-array-value}\,(\mathit{y\text{-}value},\ \mathit{y\text{-}nat},\ \mathit{temp\text{-}stk})) \\
 & = \quad \text{deposit-array-value}\,(\mathit{y\text{-}value}, \\
 & \qquad\qquad \mathit{y\text{-}nat}, \\
 & \qquad\qquad \text{deposit-array-value}\,(\mathit{x\text{-}value}, \\
 & \qquad\qquad\qquad\qquad \mathit{x\text{-}nat}, \\
 & \qquad\qquad\qquad\qquad \mathit{temp\text{-}stk})))
\end{array}
$$

EVENT: Disable deposit-array-value-commutes.

THEOREM: deposit-array-value-commutes2

$$
\begin{array}{ll}
 & (\text{mg-vars-list-ok-in-p-state}\,(\mathit{lst},\ \mathit{bindings},\ \mathit{temp\text{-}stk}) \\
\wedge & \text{no-p-aliasing}\,(\mathit{bindings},\ \mathit{lst})
\end{array}
$$

$$
\begin{array}{ll}
\wedge & \text{mg-alistp}\,(\mathit{lst}) \\
\wedge & \text{all-cars-unique}\,(\mathit{lst}) \\
\wedge & (x \in \mathit{lst}) \\
\wedge & (y \in \mathit{lst}) \\
\wedge & (\text{car}\,(x) \neq \text{car}\,(y)) \\
\wedge & (\neg\ \text{simple-mg-type-refp}\,(\text{cadr}\,(x))) \\
\wedge & (\neg\ \text{simple-mg-type-refp}\,(\text{cadr}\,(y))) \\
\wedge & (\text{length}\,(\mathit{x\text{-}value}) = \text{array-length}\,(\text{cadr}\,(x))) \\
\wedge & (\text{length}\,(\mathit{y\text{-}value}) = \text{array-length}\,(\text{cadr}\,(y)))) \\
\rightarrow & (\text{deposit-array-value}\,(\mathit{x\text{-}value}, \\
 & \qquad \text{cdr}\,(\text{assoc}\,(\text{car}\,(x),\ \mathit{bindings})), \\
 & \qquad \text{deposit-array-value}\,(\mathit{y\text{-}value}, \\
 & \qquad\qquad \text{cdr}\,(\text{assoc}\,(\text{car}\,(y),\ \mathit{bindings})), \\
 & \qquad\qquad \mathit{temp\text{-}stk})) \\
 & = \quad \text{deposit-array-value}\,(\mathit{y\text{-}value}, \\
 & \qquad\qquad \text{cdr}\,(\text{assoc}\,(\text{car}\,(y),\ \mathit{bindings})), \\
 & \qquad\qquad \text{deposit-array-value}\,(\mathit{x\text{-}value}, \\
 & \qquad\qquad\qquad \text{cdr}\,(\text{assoc}\,(\text{car}\,(x), \\
 & \qquad\qquad\qquad\qquad \mathit{bindings})), \\
 & \qquad\qquad\qquad \mathit{temp\text{-}stk})))
\end{array}
$$

EVENT: Disable deposit-array-value-commutes2.

THEOREM: deposit-alist-value-commutes

$$
\begin{array}{ll}
 & (\text{mg-vars-list-ok-in-p-state}\,(\mathit{lst},\ \mathit{bindings},\ \mathit{temp\text{-}stk}) \\
\wedge & \text{no-p-aliasing}\,(\mathit{bindings},\ \mathit{lst}) \\
\wedge & \text{mg-alistp}\,(\mathit{lst}) \\
\wedge & \text{all-cars-unique}\,(\mathit{lst}) \\
\wedge & (x \in \mathit{lst}) \\
\wedge & (y \in \mathit{lst}) \\
\wedge & (\text{car}\,(x) \neq \text{car}\,(y))) \\
\rightarrow & (\text{deposit-alist-value}\,(x, \\
 & \qquad \mathit{bindings}, \\
 & \qquad \text{deposit-alist-value}\,(y,\ \mathit{bindings},\ \mathit{temp\text{-}stk})) \\
 & = \quad \text{deposit-alist-value}\,(y, \\
 & \qquad\qquad \mathit{bindings}, \\
 & \qquad\qquad \text{deposit-alist-value}\,(x,\ \mathit{bindings},\ \mathit{temp\text{-}stk})))
\end{array}
$$

THEOREM: deposit-temp-deposit-array-value-commute3

$$
\begin{array}{ll}
 & ((\text{untag}\,(\mathit{nat}) < \text{untag}\,(\mathit{nat1})) \\
\wedge & (\text{untag}\,(\mathit{nat}) \in \mathbf{N}) \\
\wedge & ((\text{untag}\,(\mathit{nat1}) + (\text{length}\,(\mathit{lst}) - 1)) < \text{length}\,(\mathit{temp\text{-}stk}))) \\
\rightarrow & (\text{deposit-array-value}\,(\mathit{lst},\ \mathit{nat1},\ \text{deposit-temp}\,(x,\ \mathit{nat},\ \mathit{temp\text{-}stk})) \\
 & = \quad \text{deposit-temp}\,(x,\ \mathit{nat},\ \text{deposit-array-value}\,(\mathit{lst},\ \mathit{nat1},\ \mathit{temp\text{-}stk})))
\end{array}
$$

EVENT: Disable deposit-temp-deposit-array-value-commute3.

THEOREM: multiple-rputs-cancel
rput (*x*, *y*, rput (*u*, *y*, *z*)) = rput (*x*, *y*, *z*)

THEOREM: multiple-deposit-temps-cancel
deposit-temp (*x*, *nat*, deposit-temp (*y*, *nat*, *temp-stk*))
= deposit-temp (*x*, *nat*, *temp-stk*)

EVENT: Disable multiple-deposit-temps-cancel.

DEFINITION:
double-cdr-induction2 (*x*, *y*, *nat*)
= **if** *x* ≃ **nil then t**
else double-cdr-induction2 (cdr (*x*), cdr (*y*), add1-nat (*nat*)) **endif**

THEOREM: multiple-deposit-array-values-cancel
((length (*value1*) = length (*value2*))
∧ (untag (*nat*) ∈ **N**)
∧ ((untag (*nat*) + (length (*value1*) − 1)) < length (*temp-stk*)))
→ (deposit-array-value (*value1*,
nat,
deposit-array-value (*value2*, *nat*, *temp-stk*))
= deposit-array-value (*value1*, *nat*, *temp-stk*))

EVENT: Disable multiple-deposit-array-values-cancel.

THEOREM: mg-var-ok-car-definedp
((*y* ∈ *alist*) ∧ mg-vars-list-ok-in-p-state (*alist*, *bindings*, *temp-stk*))
→ definedp (car (*y*), *bindings*)

THEOREM: deposit-temp-deposit-temp-commute2
(no-p-aliasing (append (*bindings1*, *bindings2*), append (*alist1*, *alist2*))
∧ mg-vars-list-ok-in-p-state (*alist1*, *bindings1*, *temp-stk*)
∧ mg-vars-list-ok-in-p-state (*alist2*, *bindings2*, *temp-stk*)
∧ all-cars-unique (append (*alist1*, *alist2*))
∧ all-cars-unique (append (*bindings1*, *bindings2*))
∧ mg-alistp (append (*alist1*, *alist2*))
∧ (*x* ∈ *alist1*)
∧ simple-mg-type-refp (cadr (*x*))
∧ (*y* ∈ *alist2*)
∧ simple-mg-type-refp (cadr (*y*)))
→ (deposit-temp (*x-value*,

```
                cdr (assoc (car (x), bindings1)),
                deposit-temp (y-value,
                              cdr (assoc (car (y), bindings2)),
                              temp-stk))
    =   deposit-temp (y-value,
                      cdr (assoc (car (y), bindings2)),
                      deposit-temp (x-value,
                                    cdr (assoc (car (x), bindings1)),
                                    temp-stk)))
```

EVENT: Disable deposit-temp-deposit-temp-commute2.

THEOREM: deposit-temp-deposit-array-value-commute5

```
 (no-p-aliasing (append (bindings1, bindings2), append (alist1, alist2))
  ∧   mg-vars-list-ok-in-p-state (alist1, bindings1, temp-stk)
  ∧   mg-vars-list-ok-in-p-state (alist2, bindings2, temp-stk)
  ∧   all-cars-unique (append (alist1, alist2))
  ∧   all-cars-unique (append (bindings1, bindings2))
  ∧   mg-alistp (append (alist1, alist2))
  ∧   (x ∈ alist1)
  ∧   simple-mg-type-refp (cadr (x))
  ∧   (y ∈ alist2)
  ∧   (¬ simple-mg-type-refp (cadr (y))))
→   (deposit-temp (value,
                   cdr (assoc (car (x), bindings1)),
                   deposit-array-value (caddr (y),
                                        cdr (assoc (car (y), bindings2)),
                                        temp-stk))
    =   deposit-array-value (caddr (y),
                             cdr (assoc (car (y), bindings2)),
                             deposit-temp (value,
                                           cdr (assoc (car (x), bindings1)),
                                           temp-stk)))
```

EVENT: Disable deposit-temp-deposit-array-value-commute5.

THEOREM: deposit-temp-deposit-alist-value-commute2

```
 (no-p-aliasing (append (bindings1, bindings2), append (alist1, alist2))
  ∧   mg-vars-list-ok-in-p-state (alist1, bindings1, temp-stk)
  ∧   mg-vars-list-ok-in-p-state (alist2, bindings2, temp-stk)
  ∧   all-cars-unique (append (alist1, alist2))
  ∧   all-cars-unique (append (bindings1, bindings2))
  ∧   mg-alistp (append (alist1, alist2))
```

$$
\begin{array}{ll}
\wedge & (x \in \mathit{alist1}) \\
\wedge & \text{simple-mg-type-refp}\,(\text{cadr}\,(x)) \\
\wedge & (y \in \mathit{alist2})) \\
\rightarrow & (\text{deposit-temp}\,(\mathit{value}, \\
 & \qquad \text{cdr}\,(\text{assoc}\,(\text{car}\,(x),\ \mathit{bindings1})), \\
 & \qquad \text{deposit-alist-value}\,(y,\ \mathit{bindings2},\ \mathit{temp\text{-}stk})) \\
 & = \quad \text{deposit-alist-value}\,(y, \\
 & \qquad\qquad \mathit{bindings2}, \\
 & \qquad\qquad \text{deposit-temp}\,(\mathit{value}, \\
 & \qquad\qquad\qquad\qquad \text{cdr}\,(\text{assoc}\,(\text{car}\,(x),\ \mathit{bindings1})), \\
 & \qquad\qquad\qquad\qquad \mathit{temp\text{-}stk})))
\end{array}
$$

EVENT: Disable deposit-temp-deposit-alist-value-commute2.

THEOREM: deposit-array-value-deposit-array-value-commute2

$$
\begin{array}{ll}
 & (\text{no-p-aliasing}\,(\text{append}\,(\mathit{bindings1},\ \mathit{bindings2}),\ \text{append}\,(\mathit{alist1},\ \mathit{alist2})) \\
\wedge & \text{mg-vars-list-ok-in-p-state}\,(\mathit{alist1},\ \mathit{bindings1},\ \mathit{temp\text{-}stk}) \\
\wedge & \text{mg-vars-list-ok-in-p-state}\,(\mathit{alist2},\ \mathit{bindings2},\ \mathit{temp\text{-}stk}) \\
\wedge & \text{all-cars-unique}\,(\text{append}\,(\mathit{alist1},\ \mathit{alist2})) \\
\wedge & \text{all-cars-unique}\,(\text{append}\,(\mathit{bindings1},\ \mathit{bindings2})) \\
\wedge & \text{mg-alistp}\,(\text{append}\,(\mathit{alist1},\ \mathit{alist2})) \\
\wedge & (x \in \mathit{alist1}) \\
\wedge & (\neg\ \text{simple-mg-type-refp}\,(\text{cadr}\,(x))) \\
\wedge & (y \in \mathit{alist2}) \\
\wedge & (\neg\ \text{simple-mg-type-refp}\,(\text{cadr}\,(y)))) \\
\rightarrow & (\text{deposit-array-value}\,(\text{caddr}\,(x), \\
 & \qquad\qquad \text{cdr}\,(\text{assoc}\,(\text{car}\,(x),\ \mathit{bindings1})), \\
 & \qquad\qquad \text{deposit-array-value}\,(\text{caddr}\,(y), \\
 & \qquad\qquad\qquad\qquad\qquad \text{cdr}\,(\text{assoc}\,(\text{car}\,(y),\ \mathit{bindings2})), \\
 & \qquad\qquad\qquad\qquad\qquad \mathit{temp\text{-}stk})) \\
 & = \quad \text{deposit-array-value}\,(\text{caddr}\,(y), \\
 & \qquad\qquad\qquad\qquad \text{cdr}\,(\text{assoc}\,(\text{car}\,(y),\ \mathit{bindings2})), \\
 & \qquad\qquad\qquad\qquad \text{deposit-array-value}\,(\text{caddr}\,(x), \\
 & \qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{cdr}\,(\text{assoc}\,(\text{car}\,(x), \\
 & \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathit{bindings1})), \\
 & \qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathit{temp\text{-}stk})))
\end{array}
$$

EVENT: Disable deposit-array-value-deposit-array-value-commute2.

THEOREM: deposit-array-value-deposit-temp-commute

$$
\begin{array}{ll}
 & (\text{no-p-aliasing}\,(\text{append}\,(\mathit{bindings1},\ \mathit{bindings2}),\ \text{append}\,(\mathit{alist1},\ \mathit{alist2})) \\
\wedge & \text{mg-vars-list-ok-in-p-state}\,(\mathit{alist1},\ \mathit{bindings1},\ \mathit{temp\text{-}stk}) \\
\wedge & \text{mg-vars-list-ok-in-p-state}\,(\mathit{alist2},\ \mathit{bindings2},\ \mathit{temp\text{-}stk})
\end{array}
$$

$$
\begin{array}{ll}
\wedge & \text{all-cars-unique}\,(\text{append}\,(\textit{alist1},\ \textit{alist2})) \\
\wedge & \text{all-cars-unique}\,(\text{append}\,(\textit{bindings1},\ \textit{bindings2})) \\
\wedge & \text{mg-alistp}\,(\text{append}\,(\textit{alist1},\ \textit{alist2})) \\
\wedge & (x \in \textit{alist1}) \\
\wedge & \text{simple-mg-type-refp}\,(\text{cadr}\,(y)) \\
\wedge & (y \in \textit{alist2}) \\
\wedge & (\neg\ \text{simple-mg-type-refp}\,(\text{cadr}\,(x)))) \\
\rightarrow & (\text{deposit-array-value}\,(\text{caddr}\,(x), \\
 & \qquad \text{cdr}\,(\text{assoc}\,(\text{car}\,(x),\ \textit{bindings1})), \\
 & \qquad \text{deposit-temp}\,(\textit{value}, \\
 & \qquad\qquad \text{cdr}\,(\text{assoc}\,(\text{car}\,(y),\ \textit{bindings2})), \\
 & \qquad\qquad \textit{temp-stk})) \\
 & = \quad \text{deposit-temp}\,(\textit{value}, \\
 & \qquad \text{cdr}\,(\text{assoc}\,(\text{car}\,(y),\ \textit{bindings2})), \\
 & \qquad \text{deposit-array-value}\,(\text{caddr}\,(x), \\
 & \qquad\qquad \text{cdr}\,(\text{assoc}\,(\text{car}\,(x),\ \textit{bindings1})), \\
 & \qquad\qquad \textit{temp-stk})))
\end{array}
$$

EVENT: Disable deposit-array-value-deposit-temp-commute.

THEOREM: deposit-array-value-deposit-alist-value-commute

$$
\begin{array}{ll}
 & (\text{no-p-aliasing}\,(\text{append}\,(\textit{bindings1},\ \textit{bindings2}),\ \text{append}\,(\textit{alist1},\ \textit{alist2})) \\
\wedge & \text{mg-vars-list-ok-in-p-state}\,(\textit{alist1},\ \textit{bindings1},\ \textit{temp-stk}) \\
\wedge & \text{mg-vars-list-ok-in-p-state}\,(\textit{alist2},\ \textit{bindings2},\ \textit{temp-stk}) \\
\wedge & \text{all-cars-unique}\,(\text{append}\,(\textit{alist1},\ \textit{alist2})) \\
\wedge & \text{all-cars-unique}\,(\text{append}\,(\textit{bindings1},\ \textit{bindings2})) \\
\wedge & \text{mg-alistp}\,(\text{append}\,(\textit{alist1},\ \textit{alist2})) \\
\wedge & (x \in \textit{alist1}) \\
\wedge & (\neg\ \text{simple-mg-type-refp}\,(\text{cadr}\,(x))) \\
\wedge & (y \in \textit{alist2})) \\
\rightarrow & (\text{deposit-array-value}\,(\text{caddr}\,(x), \\
 & \qquad \text{cdr}\,(\text{assoc}\,(\text{car}\,(x),\ \textit{bindings1})), \\
 & \qquad \text{deposit-alist-value}\,(y,\ \textit{bindings2},\ \textit{temp-stk})) \\
 & = \quad \text{deposit-alist-value}\,(y, \\
 & \qquad\quad \textit{bindings2}, \\
 & \qquad\quad \text{deposit-array-value}\,(\text{caddr}\,(x), \\
 & \qquad\qquad\qquad \text{cdr}\,(\text{assoc}\,(\text{car}\,(x), \\
 & \qquad\qquad\qquad\qquad \textit{bindings1})), \\
 & \qquad\qquad\qquad \textit{temp-stk})))
\end{array}
$$

EVENT: Disable deposit-array-value-deposit-alist-value-commute.

THEOREM: deposit-alist-value-commute2

(no-p-aliasing (append (*bindings1*, *bindings2*), append (*alist1*, *alist2*))
 ∧ mg-vars-list-ok-in-p-state (*alist1*, *bindings1*, *temp-stk*)
 ∧ mg-vars-list-ok-in-p-state (*alist2*, *bindings2*, *temp-stk*)
 ∧ all-cars-unique (append (*alist1*, *alist2*))
 ∧ all-cars-unique (append (*bindings1*, *bindings2*))
 ∧ mg-alistp (append (*alist1*, *alist2*))
 ∧ (*x* ∈ *alist1*)
 ∧ (*y* ∈ *alist2*))
→ (deposit-alist-value (*x*,
bindings1,
deposit-alist-value (*y*, *bindings2*, *temp-stk*))
= deposit-alist-value (*y*,
bindings2,
deposit-alist-value (*x*, *bindings1*, *temp-stk*)))

EVENT: Disable deposit-alist-value-commute2.

```
;; %deposits commute
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                              ;;
;;                        %Map-Down-Values Commutes                             ;;
;;                                                                              ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

THEOREM: map-down-values-deposit-alist-value-commute
(no-p-aliasing (*bindings*, cons (*x*, *lst*))
 ∧ mg-vars-list-ok-in-p-state (cons (*x*, *lst*), *bindings*, *temp-stk*)
 ∧ mg-alistp (cons (*x*, *lst*))
 ∧ all-cars-unique (cons (*x*, *lst*)))
→ (map-down-values (*lst*,
bindings,
deposit-alist-value (*x*, *bindings*, *temp-stk*))
= deposit-alist-value (*x*,
bindings,
map-down-values (*lst*, *bindings*, *temp-stk*)))

THEOREM: deposit-temp-map-down-values-commute
(all-cars-unique (cons (*x*, *lst*))
 ∧ mg-alistp (cons (*x*, *lst*))
 ∧ no-p-aliasing (*bindings*, cons (*x*, *lst*))
 ∧ mg-vars-list-ok-in-p-state (cons (*x*, *lst*), *bindings*, *temp-stk*))
→ (deposit-temp (*y*,
cdr (assoc (car (*x*), *bindings*)),

map-down-values (*lst*, *bindings*, *temp-stk*))
= map-down-values (*lst*,
bindings,
deposit-temp (*y*,
cdr (assoc (car (*x*), *bindings*)),
temp-stk)))

EVENT: Disable deposit-temp-map-down-values-commute.

EVENT: Disable no-p-aliasing.

THEOREM: deposit-temp-map-down-values-commute-2
(all-cars-unique (cons (*x*, *lst*))
∧ mg-alistp (cons (*x*, *lst*))
∧ no-p-aliasing (*bindings*, cons (*x*, *lst*))
∧ mg-vars-list-ok-in-p-state (cons (*x*, *lst*), *bindings*, *temp-stk*))
→ (map-down-values (*lst*,
bindings,
deposit-temp (*y*, cdr (assoc (car (*x*), *bindings*)), *temp-stk*))
= deposit-temp (*y*,
cdr (assoc (car (*x*), *bindings*)),
map-down-values (*lst*, *bindings*, *temp-stk*)))

EVENT: Disable deposit-temp-map-down-values-commute-2.

THEOREM: deposit-same-value-doesnt-disturb-map-down-values
(mg-vars-list-ok-in-p-state (*mg-vars*, *bindings*, *temp-stk*)
∧ all-cars-unique (*mg-vars*)
∧ mg-alistp (*mg-vars*)
∧ no-p-aliasing (*bindings*, *mg-vars*)
∧ (*x* ∈ listcars (*mg-vars*))
∧ simple-mg-type-refp (cadr (assoc (*x*, *mg-vars*))))
→ (rput (mg-to-p-simple-literal (caddr (assoc (*x*, *mg-vars*))),
untag (cdr (assoc (*x*, *bindings*))),
map-down-values (*mg-vars*, *bindings*, *temp-stk*))
= map-down-values (*mg-vars*, *bindings*, *temp-stk*))

EVENT: Disable deposit-same-value-doesnt-disturb-map-down-values.

THEOREM: deposit-same-array-value-doesnt-disturb-map-down-values
(mg-vars-list-ok-in-p-state (*mg-vars*, *bindings*, *temp-stk*)
∧ all-cars-unique (*mg-vars*)

$\wedge$ mg-alistp (*mg-vars*)
$\wedge$ no-p-aliasing (*bindings*, *mg-vars*)
$\wedge$ (*x* $\in$ listcars (*mg-vars*))
$\wedge$ ($\neg$ simple-mg-type-refp (cadr (assoc (*x*, *mg-vars*)))))
$\rightarrow$ (deposit-array-value (caddr (assoc (*x*, *mg-vars*)),
cdr (assoc (*x*, *bindings*)),
map-down-values (*mg-vars*, *bindings*, *temp-stk*))
= map-down-values (*mg-vars*, *bindings*, *temp-stk*))

EVENT: Disable deposit-same-array-value-doesnt-disturb-map-down-values.

THEOREM: deposit-same-value-doesnt-disturb-map-down-values2
(mg-vars-list-ok-in-p-state (*mg-vars*, *bindings*, *temp-stk*)
$\wedge$ all-cars-unique (*mg-vars*)
$\wedge$ mg-alistp (*mg-vars*)
$\wedge$ no-p-aliasing (*bindings*, *mg-vars*)
$\wedge$ (*x* $\in$ listcars (*mg-vars*))
$\wedge$ ($\neg$ simple-mg-type-refp (cadr (assoc (*x*, *mg-vars*)))))
$\rightarrow$ (rput (mg-to-p-simple-literal (caaddr (assoc (*x*, *mg-vars*))),
untag (cdr (assoc (*x*, *bindings*))),
map-down-values (*mg-vars*, *bindings*, *temp-stk*))
= map-down-values (*mg-vars*, *bindings*, *temp-stk*))

THEOREM: deposit-same-array-value-doesnt-disturb-map-down-values2
(mg-vars-list-ok-in-p-state (*mg-vars*, *bindings*, *temp-stk*)
$\wedge$ all-cars-unique (*mg-vars*)
$\wedge$ mg-alistp (*mg-vars*)
$\wedge$ no-p-aliasing (*bindings*, *mg-vars*)
$\wedge$ (*x* $\in$ listcars (*mg-vars*))
$\wedge$ ($\neg$ simple-mg-type-refp (cadr (assoc (*x*, *mg-vars*)))))
$\rightarrow$ (deposit-array-value (cdr (caddr (assoc (*x*, *mg-vars*))),
add1-nat (cdr (assoc (*x*, *bindings*))),
map-down-values (*mg-vars*, *bindings*, *temp-stk*))
= map-down-values (*mg-vars*, *bindings*, *temp-stk*))

THEOREM: map-down-values-commutes1
(all-cars-unique (append (*alist1*, *alist2*))
$\wedge$ mg-alistp (append (*alist1*, *alist2*))
$\wedge$ no-p-aliasing (*bindings*, append (*alist1*, *alist2*))
$\wedge$ mg-vars-list-ok-in-p-state (append (*alist1*, *alist2*), *bindings*, *temp-stk*))
$\rightarrow$ (map-down-values (*alist1*,
bindings,
map-down-values (*alist2*, *bindings*, *temp-stk*))
= map-down-values (*alist2*,

bindings,
map-down-values (*alist1*, *bindings*, *temp-stk*)))

THEOREM: listcars-restriction1
((listcars (*alist*) = append (listcars (*alist1*), listcars (*alist2*)))
∧ plistp (*alist*)
∧ no-duplicates (listcars (*alist*)))
→ (append (restrict (*alist*, listcars (*alist1*)),
restrict (*alist*, listcars (*alist2*)))
= *alist*)

EVENT: Disable listcars-restriction1.

THEOREM: meaning-restriction-append1
((listcars (*alist*) = append (listcars (*alist1*), listcars (*alist2*)))
∧ no-duplicates (listcars (*alist*))
∧ mg-alistp (*alist*)
∧ no-p-aliasing (*bindings*, *alist*)
∧ mg-vars-list-ok-in-p-state (*alist*, *bindings*, *temp-stk*))
→ (map-down-values (*alist*, *bindings*, *temp-stk*)
= map-down-values (restrict (*alist*, listcars (*alist1*)),
bindings,
map-down-values (restrict (*alist*,
listcars (*alist2*)),
bindings,
temp-stk)))

EVENT: Disable meaning-restriction-append1.

THEOREM: deposit-alist-value-map-down-values-commute2
(no-p-aliasing (append (*bindings1*, *bindings2*), append (*alist1*, *alist2*))
∧ mg-vars-list-ok-in-p-state (*alist1*, *bindings1*, *temp-stk*)
∧ mg-vars-list-ok-in-p-state (*alist2*, *bindings2*, *temp-stk*)
∧ all-cars-unique (append (*alist1*, *alist2*))
∧ all-cars-unique (append (*bindings1*, *bindings2*))
∧ plistp (*alist1*)
∧ mg-alistp (append (*alist1*, *alist2*))
∧ (*x* ∈ *alist1*))
→ (deposit-alist-value (*x*,
bindings1,
map-down-values (*alist2*, *bindings2*, *temp-stk*))
= map-down-values (*alist2*,
bindings2,
deposit-alist-value (*x*, *bindings1*, *temp-stk*)))

EVENT: Disable deposit-alist-value-map-down-values-commute2.

THEOREM: map-down-values-commute
(all-cars-unique (append (*alist1*, *alist2*))
 ∧ all-cars-unique (append (*bindings1*, *bindings2*))
 ∧ plistp (*alist1*)
 ∧ mg-alistp (append (*alist1*, *alist2*))
 ∧ no-p-aliasing (append (*bindings1*, *bindings2*), append (*alist1*, *alist2*))
 ∧ mg-vars-list-ok-in-p-state (*alist1*, *bindings1*, *temp-stk*)
 ∧ mg-vars-list-ok-in-p-state (*alist2*, *bindings2*, *temp-stk*))
→ (map-down-values (*alist1*,
 bindings1,
 map-down-values (*alist2*, *bindings2*, *temp-stk*))
 = map-down-values (*alist2*,
 bindings2,
 map-down-values (*alist1*, *bindings1*, *temp-stk*)))

EVENT: Disable map-down-values-commute.

THEOREM: deposit-temp-doesnt-affect-map-down-values
(mg-alistp (cons (*x*, *mg-vars*))
 ∧ all-cars-unique (cons (*x*, *mg-vars*))
 ∧ no-p-aliasing (*bindings*, cons (*x*, *mg-vars*))
 ∧ mg-vars-list-ok-in-p-state (cons (*x*, *mg-vars*), *bindings*, *temp-stk*))
→ (map-down-values (*mg-vars*,
 bindings,
 deposit-temp (*value*,
 cdr (assoc (car (*x*), *bindings*)),
 temp-stk))
 = deposit-temp (*value*,
 cdr (assoc (car (*x*), *bindings*)),
 map-down-values (*mg-vars*, *bindings*, *temp-stk*)))

EVENT: Disable deposit-temp-doesnt-affect-map-down-values.

DEFINITION:
rget-array-mapping-induction-hint (*value*, *index*, *nat*, *temp-stk*)
= **if** *value* ≃ **nil** **then** **t**
 elseif *index* ≃ `0` **then** **t**
 else rget-array-mapping-induction-hint (cdr (*value*),
 index − 1,
 add1-nat (*nat*),
 deposit-temp (mg-to-p-simple-literal (car (*value*)),

nat,
temp-stk)) **endif**

THEOREM: rget-array-index-mapping0
(((untag (*nat*) + (length (*value*) − 1)) < length (*temp-stk*))
∧ (untag (*nat*) ∈ **N**)
∧ (*index* < length (*value*)))
→ (rget (untag (*nat*) + *index*, deposit-array-value (*value*, *nat*, *temp-stk*))
= mg-to-p-simple-literal (get (*index*, *value*)))

THEOREM: rget-array-index-mapping
(mg-alistp (*mg-vars*)
∧ all-cars-unique (*mg-vars*)
∧ no-p-aliasing (*bindings*, *mg-vars*)
∧ array-identifierp (*a*, *mg-vars*)
∧ mg-vars-list-ok-in-p-state (*mg-vars*, *bindings*, *temp-stk*)
∧ (*index* < array-length (cadr (assoc (*a*, *mg-vars*)))))
→ (rget (untag (value (*a*, *bindings*)) + *index*,
map-down-values (*mg-vars*, *bindings*, *temp-stk*))
= mg-to-p-simple-literal (get (*index*, caddr (assoc (*a*, *mg-vars*)))))

EVENT: Disable rget-array-index-mapping.

THEOREM: append-doesnt-affect-rput
(*i* < length (*lst2*))
→ (rput (*value*, *i*, append (*lst1*, *lst2*)) = append (*lst1*, rput (*value*, *i*, *lst2*)))

EVENT: Disable append-doesnt-affect-rput.

THEOREM: append-doesnt-affect-deposit-temp
(definedp (*b*, *mg-vars*)
∧ mg-alistp (*mg-vars*)
∧ mg-vars-list-ok-in-p-state (*mg-vars*, *bindings*, *temp-stk*))
→ (deposit-temp (*value*,
cdr (assoc (*b*, *bindings*)),
append (*lst*, map-down-values (*mg-vars*, *bindings*, *temp-stk*)))
= append (*lst*,
deposit-temp (*value*,
cdr (assoc (*b*, *bindings*)),
map-down-values (*mg-vars*, *bindings*, *temp-stk*))))

EVENT: Disable append-doesnt-affect-deposit-temp.

```
;; %map-down-values commutes
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                            ;;
;;                  %Ascending-Local-Address-Sequence                         ;;
;;                                                                            ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; This function is used to describe the addresses pushed for the locals upon
;; entry to a procedure call.
```

DEFINITION:
ascending-local-address-sequence (*locals*, *n*)
= **if** *locals* ≃ **nil then nil**
elseif simple-mg-type-refp (formal-type (car (*locals*)))
then cons (tag (`'nat`, *n*),
ascending-local-address-sequence (cdr (*locals*), 1 + *n*))
else cons (tag (`'nat`, *n*),
ascending-local-address-sequence (cdr (*locals*),
array-length (formal-type (car (*locals*)))
+ *n*)) **endif**

THEOREM: length-ascending-local-address-sequence
length (ascending-local-address-sequence (*locals*, *k*)) = length (*locals*)

THEOREM: ascending-local-address-sequence-plistp
plistp (ascending-local-address-sequence (*x*, *y*))

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                            ;;
;;                     %All-Pointer-Smaller/Bigger                            ;;
;;                                                                            ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; These functions are used strictly in the proof that there is no-p-aliasing
;; in the call environment.
;; The previous version assumed that the pointers were still in the bindings.
;; This version assumes that we've applied the function collect-pointers and
;; therefore have a list of numberp's.
```

DEFINITION:
all-pointers-smaller (*lst*, *n*)
= **if** *lst* ≃ **nil then t**
else (car (*lst*) < *n*) ∧ all-pointers-smaller (cdr (*lst*), *n*) **endif**

THEOREM: all-pointers-smaller-distributes
all-pointers-smaller (append (*lst1*, *lst2*), *n*)
= (all-pointers-smaller (*lst1*, *n*) ∧ all-pointers-smaller (*lst2*, *n*))

EVENT: Disable all-pointers-smaller-distributes.

DEFINITION:
all-pointers-bigger (*lst*, *n*)
= **if** *lst* ≃ **nil then t**
else (*n* ≤ car (*lst*)) ∧ all-pointers-bigger (cdr (*lst*), *n*) **endif**

THEOREM: all-pointers-bigger-distributes
all-pointers-bigger (append (*lst1*, *lst2*), *n*)
= (all-pointers-bigger (*lst1*, *n*) ∧ all-pointers-bigger (*lst2*, *n*))

THEOREM: pointers-bigger-monotonic
((*m* ≮ *n*) ∧ all-pointers-bigger (*x*, *m*)) → all-pointers-bigger (*x*, *n*)

EVENT: Disable pointers-bigger-monotonic.

THEOREM: all-pointers-smaller-member
((*i* ≮ *n*) ∧ all-pointers-smaller (*lst*, *n*)) → (*i* ∉ *lst*)

EVENT: Disable all-pointers-smaller-member.

THEOREM: all-pointers-bigger-member
((*i* < *n*) ∧ all-pointers-bigger (*lst*, *n*)) → (*i* ∉ *lst*)

EVENT: Disable all-pointers-bigger-member.

DEFINITION:
successive-pointers-bigger-induction-hint (*nat*, *n*, *m*)
= **if** *m* ≃ `0` **then t**
else successive-pointers-bigger-induction-hint (add1-nat (*nat*),
1 + *n*,
m − 1) **endif**

THEOREM: successive-pointers-bigger0
(*nat* = tag (`'nat`, *n*))
→ all-pointers-bigger (n-successive-pointers (*nat*, *m*), *n*)

EVENT: Disable successive-pointers-bigger0.

THEOREM: successive-pointers-bigger
all-pointers-bigger (n-successive-pointers (tag ('nat, n), m), n)

EVENT: Disable successive-pointers-bigger.

THEOREM: no-duplicates-all-pointers-disjoint
(no-duplicates (*lst1*)
 ∧ no-duplicates (*lst2*)
 ∧ all-pointers-smaller (*lst2*, n)
 ∧ all-pointers-bigger (*lst1*, n))
→ no-duplicates (append (*lst1*, *lst2*))

EVENT: Disable no-duplicates-all-pointers-disjoint.

THEOREM: no-duplicates-all-pointers-disjoint2
(no-duplicates (*lst1*)
 ∧ no-duplicates (*lst2*)
 ∧ all-pointers-smaller (*lst1*, n)
 ∧ all-pointers-bigger (*lst2*, n))
→ no-duplicates (append (*lst1*, *lst2*))

THEOREM: successive-pointers-smaller2
((untag (*nat*) ∈ **N**) ∧ ($k \not< (\text{untag}\,(nat) + m)$))
→ all-pointers-smaller (n-successive-pointers (*nat*, m), k)

THEOREM: no-p-aliasing-append1
(no-p-aliasing (*bindings*, *lst1*)
 ∧ no-p-aliasing (*bindings*, *lst2*)
 ∧ all-pointers-bigger (collect-pointers (*bindings*, *lst1*), n)
 ∧ all-pointers-smaller (collect-pointers (*bindings*, *lst2*), n))
→ no-p-aliasing (*bindings*, append (*lst1*, *lst2*))

EVENT: Disable no-p-aliasing-append1.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                              ;;
;;                          %Mapping Data Up                                    ;;
;;                                                                              ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; The list of mg-vars is of the form (... (cons name (cons value type)) ...)
;; and the p-vars of the form (... (cons name value) ...).  This takes the
;; p-vars and maps them up to the corresponding mg-vars list.  Using the types
```

```
;; to give the value-value mapping.

;; In mapping up from the Piton to the MG world, I need to know the signatures of
;; the variables at the lower level but not their values.  This returns a value
;; which is just like the variable alist but has all of the values set to (false).

;; The map up from Piton to MG world now has a layer of indirection.  That is,
;; the variable in the Piton world contains an address to the value in the
;; my-stack array.

;; Notice that the other stuff which may appear on an alist entry is
;; not preserved.
;; Notice that the value of the Piton variable is actually on the
;; my-stack data structure.  Consequently, we require a "fetch" into
;; that data structure to get it.  Thus, we must have the Piton state
;; p-data-segment as a parameter to the map-up-vars functions.

;; Every local in the bindings of the top frame contains a natural which is
;; an index into the temp-stk where the actual value is stored.

;; This should be removed when I have the opportunity to redo the events and
;; adjust for this difference.
```

DEFINITION:
map-up-vars-list-array (*p-var*, *temp-stk*, *var-signature*)
= list (car (*var-signature*),
cadr (*var-signature*),
p-to-mg-simple-literal-list (fetch-n-temp-stk-elements (*temp-stk*,
cdr (*p-var*),
array-length (cadr (*var-signature*))),
array-elemtype (cadr (*var-signature*))))

EVENT: Disable map-up-vars-list-array.

DEFINITION:
map-up-vars-list-simple-element (*p-var*, *temp-stk*, *var-signature*)
= list (car (*var-signature*),
cadr (*var-signature*),
p-to-mg-simple-literal (fetch-temp (cdr (*p-var*), *temp-stk*),
cadr (*var-signature*)))

EVENT: Disable map-up-vars-list-simple-element.

DEFINITION:
map-up-vars-list-element (*p-var*, *temp-stk*, *var-signature*)
= **if** simple-mg-type-refp (cadr (*var-signature*))
then map-up-vars-list-simple-element (*p-var*, *temp-stk*, *var-signature*)
else map-up-vars-list-array (*p-var*, *temp-stk*, *var-signature*) **endif**

```
;; p-vars here is the bindings component from the topmost frame of the
;; P-CTRL-STK.
```

DEFINITION:
map-up-vars-list (*p-vars*, *temp-stk*, *signature*)
= **if** *signature* ≃ **nil then nil**
else cons (map-up-vars-list-element (assoc (caar (*signature*), *p-vars*),
temp-stk,
car (*signature*)),
map-up-vars-list (*p-vars*, *temp-stk*, cdr (*signature*))) **endif**

```
;; The mg-state is computed from the p-state by tranforming the value of the
;; global int variable 'c-c to an mg condition.  The vars are mapped up and
;; given the corresponding values in the mg-state.
```

DEFINITION:
piton-cc (*p*) = fetch-adp (`'(c-c . 0)`, p-data-segment (*p*))

DEFINITION:
map-up (*p-state*, *mg-signature*, *cond-list*)
= mg-state (p-nat-to-mg-cond (piton-cc (*p-state*), *cond-list*),
map-up-vars-list (bindings (top (p-ctrl-stk (*p-state*))),
p-temp-stk (*p-state*),
mg-signature),
`'run`)

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                            ;;
;;                        %Map Up Inverts Map Down                            ;;
;;                                                                            ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

THEOREM: definedp-car-cons2
listp (*y*) → definedp (caar (*y*), cons (*x*, *y*))

THEOREM: put-doesnt-affect-get

$((n \in \mathbf{N})$
$\wedge \quad (m \in \mathbf{N})$
$\wedge \quad (n < \text{length}\,(z))$
$\wedge \quad (m < \text{length}\,(z))$
$\wedge \quad (n \neq m))$
$\rightarrow \quad (\text{get}\,(n,\, \text{put}\,(x,\, m,\, z)) = \text{get}\,(n,\, z))$

EVENT: Disable put-doesnt-affect-get.

THEOREM: rput-doesnt-affect-rget
$((n \in \mathbf{N})$
$\wedge \quad (m \in \mathbf{N})$
$\wedge \quad (n < \text{length}\,(z))$
$\wedge \quad (m < \text{length}\,(z))$
$\wedge \quad (n \neq m))$
$\rightarrow \quad (\text{rget}\,(n,\, \text{rput}\,(x,\, m,\, z)) = \text{rget}\,(n,\, z))$

EVENT: Disable rput-doesnt-affect-rget.

THEOREM: deposit-temp-doesnt-affect-fetch-temp
$((\text{untag}\,(\textit{nat1}) \neq \text{untag}\,(\textit{nat2}))$
$\wedge \quad (\text{untag}\,(\textit{nat1}) \in \mathbf{N})$
$\wedge \quad (\text{untag}\,(\textit{nat2}) \in \mathbf{N})$
$\wedge \quad (\text{untag}\,(\textit{nat1}) < \text{length}\,(\textit{temp-stk}))$
$\wedge \quad (\text{untag}\,(\textit{nat2}) < \text{length}\,(\textit{temp-stk})))$
$\rightarrow \quad (\text{fetch-temp}\,(\textit{nat1},\, \text{deposit-temp}\,(\textit{value},\, \textit{nat2},\, \textit{temp-stk}))$
$\quad = \quad \text{fetch-temp}\,(\textit{nat1},\, \textit{temp-stk}))$

EVENT: Disable deposit-temp-doesnt-affect-fetch-temp.

THEOREM: deposit-array-value-doesnt-affect-fetch-temp
$((\text{untag}\,(\textit{nat1}) \in \mathbf{N})$
$\wedge \quad (\text{untag}\,(\textit{nat2}) \in \mathbf{N})$
$\wedge \quad (\text{untag}\,(\textit{nat1}) \notin \text{n-successive-pointers}\,(\textit{nat2},\, \text{length}\,(\textit{value})))$
$\wedge \quad (\text{untag}\,(\textit{nat1}) < \text{length}\,(\textit{temp-stk}))$
$\wedge \quad ((\text{untag}\,(\textit{nat2}) + (\text{length}\,(\textit{value}) - 1)) < \text{length}\,(\textit{temp-stk})))$
$\rightarrow \quad (\text{fetch-temp}\,(\textit{nat1},\, \text{deposit-array-value}\,(\textit{value},\, \textit{nat2},\, \textit{temp-stk}))$
$\quad = \quad \text{fetch-temp}\,(\textit{nat1},\, \textit{temp-stk}))$

EVENT: Disable deposit-array-value-doesnt-affect-fetch-temp.

THEOREM: deposit-temp-doesnt-affect-fetch-n-temp-stk-elements
$((\text{untag}\,(\textit{nat1}) \in \mathbf{N})$

$\wedge$ (untag (*nat2*) $\in$ **N**)
$\wedge$ (untag (*nat2*) $\notin$ n-successive-pointers (*nat1*, *n*))
$\wedge$ (untag (*nat2*) $<$ length (*temp-stk*))
$\wedge$ ((untag (*nat1*) + (*n* $-$ 1)) $<$ length (*temp-stk*)))
$\rightarrow$ (fetch-n-temp-stk-elements (deposit-temp (*x*, *nat2*, *temp-stk*), *nat1*, *n*)
= fetch-n-temp-stk-elements (*temp-stk*, *nat1*, *n*))

EVENT: Disable deposit-temp-doesnt-affect-fetch-n-temp-stk-elements.

THEOREM: deposit-array-value-doesnt-affect-fetch-n-temp-stk-elements
((untag (*nat1*) $\in$ **N**)
$\wedge$ (untag (*nat2*) $\in$ **N**)
$\wedge$ ((untag (*nat2*) + (*n* $-$ 1)) $<$ length (*temp-stk*))
$\wedge$ ((untag (*nat1*) + (length (*value*) $-$ 1)) $<$ length (*temp-stk*))
$\wedge$ disjoint (n-successive-pointers (*nat1*, length (*value*)),
n-successive-pointers (*nat2*, *n*)))
$\rightarrow$ (fetch-n-temp-stk-elements (deposit-array-value (*value*, *nat1*, *temp-stk*),
nat2,
n)
= fetch-n-temp-stk-elements (*temp-stk*, *nat2*, *n*))

EVENT: Disable deposit-array-value-doesnt-affect-fetch-n-temp-stk-elements.

THEOREM: deposit-at-lesser-index-doesnt-affect-fetch
((untag (*nat1*) $\in$ **N**)
$\wedge$ (untag (*nat1*) $<$ untag (*nat2*))
$\wedge$ ((untag (*nat2*) + (*n* $-$ 1)) $<$ length (*temp-stk*)))
$\rightarrow$ (fetch-n-temp-stk-elements (deposit-temp (*x*, *nat1*, *temp-stk*), *nat2*, *n*)
= fetch-n-temp-stk-elements (*temp-stk*, *nat2*, *n*))

EVENT: Disable deposit-at-lesser-index-doesnt-affect-fetch.

THEOREM: append-doesnt-affect-rget
(*n* $<$ length (*temp-stk*))
$\rightarrow$ (rget (*n*, append (*lst*, *temp-stk*)) = rget (*n*, *temp-stk*))

THEOREM: append-doesnt-affect-rget-corollary
(*n* $<$ length (*temp-stk*))
$\rightarrow$ (rget (*n*, push (*x*, *temp-stk*)) = rget (*n*, *temp-stk*))

DEFINITION:
fetch-temp-stk-elements-induction-hint (*value*, *temp-stk*, *nat*, *n*)
= **if** *value* $\simeq$ **nil then t**

else fetch-temp-stk-elements-induction-hint (cdr (*value*),
temp-stk,
add1-nat (*nat*),
$n - 1$) **endif**

THEOREM: fetch-n-temp-stk-elements-inverts-deposit-array-value
((*n* = length (*value*))
∧ (untag (*nat*) ∈ **N**)
∧ ((untag (*nat*) + (length (*value*) − 1)) < length (*temp-stk*)))
→ (fetch-n-temp-stk-elements (deposit-array-value (*value*, *nat*, *temp-stk*),
nat,
n)
= mg-to-p-simple-literal-list (*value*))

EVENT: Disable fetch-n-temp-stk-elements-inverts-deposit-array-value.

THEOREM: simple-vars-have-simple-values
(mg-alist-elementp (*x*) ∧ simple-mg-type-refp (cadr (*x*)))
→ simple-typed-literalp (caddr (*x*), cadr (*x*))

EVENT: Disable simple-vars-have-simple-values.

THEOREM: array-vars-have-simple-plistp-values
(mg-alist-elementp (*x*) ∧ (¬ simple-mg-type-refp (cadr (*x*))))
→ simple-typed-literal-plistp (caddr (*x*), array-elemtype (cadr (*x*)))

EVENT: Disable array-vars-have-simple-plistp-values.

THEOREM: map-up-map-down-preserves-mg-alist-elements-equal
((*mg-vars* ≠ **nil**)
∧ all-cars-unique (*mg-vars*)
∧ mg-alistp (*mg-vars*)
∧ no-p-aliasing (*bindings*, *mg-vars*)
∧ mg-vars-list-ok-in-p-state (*mg-vars*, *bindings*, *temp-stk*))
→ (map-up-vars-list-element (assoc (caar (*mg-vars*), *bindings*),
map-down-values (*mg-vars*, *bindings*, *temp-stk*),
list (caar (*mg-vars*), cadar (*mg-vars*)))
= car (*mg-vars*))

EVENT: Disable map-up-map-down-preserves-mg-alist-elements-equal.

```
;; The proof of this should really be "packaged" up into the four
;; cases depending on which of x and (car mg-vars) are simple or not.
```

THEOREM: deposit-alist-value-doesnt-affect-map-up-vars-list0
(all-cars-unique (cons (*x*, *mg-vars*))
 ∧ mg-alistp (cons (*x*, *mg-vars*))
 ∧ mg-vars-list-ok-in-p-state (cons (*x*, *mg-vars*), *bindings*, *temp-stk*)
 ∧ no-p-aliasing (*bindings*, cons (*x*, *mg-vars*)))
→ (map-up-vars-list (*bindings*,
 deposit-alist-value (*x*, *bindings*, *temp-stk*),
 signature (*mg-vars*))
 = map-up-vars-list (*bindings*, *temp-stk*, signature (*mg-vars*)))

EVENT: Disable deposit-alist-value-doesnt-affect-map-up-vars-list0.

THEOREM: deposit-alist-value-doesnt-affect-map-up-vars-list
(all-cars-unique (cons (*x*, *mg-vars*))
 ∧ mg-alistp (cons (*x*, *mg-vars*))
 ∧ mg-vars-list-ok-in-p-state (cons (*x*, *mg-vars*), *bindings*, *temp-stk*)
 ∧ no-p-aliasing (*bindings*, cons (*x*, *mg-vars*)))
→ (map-up-vars-list (*bindings*,
 map-down-values (*mg-vars*,
 bindings,
 deposit-alist-value (*x*,
 bindings,
 temp-stk)),
 signature (*mg-vars*))
 = map-up-vars-list (*bindings*,
 map-down-values (*mg-vars*, *bindings*, *temp-stk*),
 signature (*mg-vars*)))

EVENT: Disable deposit-alist-value-doesnt-affect-map-up-vars-list.

THEOREM: rget-rewrite1
(mg-alistp (*mg-vars*)
 ∧ all-cars-unique (*mg-vars*)
 ∧ no-p-aliasing (*bindings*, *mg-vars*)
 ∧ simple-identifierp (*y*, *mg-vars*)
 ∧ mg-vars-list-ok-in-p-state (*mg-vars*, *bindings*, *temp-stk*))
→ (rget (untag (value (*y*, *bindings*)),
 map-down-values (*mg-vars*, *bindings*, *temp-stk*))
 = mg-to-p-simple-literal (caddr (assoc (*y*, *mg-vars*))))

THEOREM: set-alist-value-deposit-temp-relation
(mg-alistp (*mg-vars*)
 ∧ all-cars-unique (*mg-vars*)
 ∧ no-p-aliasing (*bindings*, *mg-vars*)

$\wedge$ simple-identifierp (*y*, *mg-vars*)
$\wedge$ mg-vars-list-ok-in-p-state (*mg-vars*, *bindings*, *temp-stk*)
$\wedge$ ok-mg-valuep (*value*, cadr (assoc (*y*, *mg-vars*))))
$\rightarrow$ (map-down-values (set-alist-value (*y*, *value*, *mg-vars*), *bindings*, *temp-stk*)
= rput (mg-to-p-simple-literal (*value*),
untag (value (*y*, *bindings*)),
map-down-values (*mg-vars*, *bindings*, *temp-stk*)))

EVENT: Disable set-alist-value-deposit-temp-relation.

THEOREM: set-alist-value-deposit-array-value-relation
(mg-alistp (*mg-vars*)
$\wedge$ all-cars-unique (*mg-vars*)
$\wedge$ no-p-aliasing (*bindings*, *mg-vars*)
$\wedge$ array-identifierp (*a*, *mg-vars*)
$\wedge$ ok-mg-array-value (*lst*, cadr (assoc (*a*, *mg-vars*)))
$\wedge$ mg-vars-list-ok-in-p-state (*mg-vars*, *bindings*, *temp-stk*))
$\rightarrow$ (map-down-values (set-alist-value (*a*, *lst*, *mg-vars*), *bindings*, *temp-stk*)
= deposit-array-value (*lst*,
value (*a*, *bindings*),
map-down-values (*mg-vars*, *bindings*, *temp-stk*)))

EVENT: Disable set-alist-value-deposit-array-value-relation.

THEOREM: append-doesnt-affect-deposit-alist-value
(car (*x*) $\in$ listcars (*bindings1*))
$\rightarrow$ (deposit-alist-value (*x*, append (*bindings1*, *bindings2*), *temp-stk*)
= deposit-alist-value (*x*, *bindings1*, *temp-stk*))

THEOREM: deposit-temp-length-cons
plistp (*temp-stk*)
$\rightarrow$ (deposit-temp (*x*, tag (`'nat`, length (*temp-stk*)), cons (*y*, *temp-stk*))
= cons (*x*, *temp-stk*))

EVENT: Disable deposit-temp-length-cons.

DEFINITION:
deposit-array-value-same-value-induction-hint (*array-value*, *temp-stk*)
= **if** *array-value* $\simeq$ **nil then t**
else deposit-array-value-same-value-induction-hint (cdr (*array-value*),
deposit-temp (mg-to-p-simple-literal (car (*array-value*)
tag (`'nat`,
length (*temp-stk*)),

cons (mg-to-p-simple-literal (car (*array-temp-stk*))) **endif**

THEOREM: deposit-temp-preserves-plistp
plistp (*z*) → plistp (deposit-temp (*x*, *y*, *z*))

THEOREM: deposit-array-value-same-value-doesnt-disturb-temp-stk
plistp (*temp-stk*)
→ (deposit-array-value (*array-value*,
tag (`'nat`, length (*temp-stk*)),
append (*lst*,
append (reverse (mg-to-p-simple-literal-list (*array-value*)),
temp-stk)))
= append (*lst*,
append (reverse (mg-to-p-simple-literal-list (*array-value*)),
temp-stk)))

EVENT: Disable deposit-array-value-same-value-doesnt-disturb-temp-stk.

EVENT: Make the library `"c4"`.

Index