```
;; Requires sets.
```

EVENT: Start with the library "sets".

```
;; Alists, March 1990.  Most of the definitions and some of the lemmas
;; were contributed by Bill Bevier; the rest are by Matt Kaufmann.

;; Functions defined here:

;; (deftheory alist-defns
;;   (alistp domain range value bind rembind invert mapping
;;           restrict co-restrict))
```

DEFINITION:
alistp (x)
= **if** listp (x) **then** listp (car (x)) $\land$ alistp (cdr (x))
else x = **nil** **endif**

THEOREM: alistp-implies-properp
alistp (x) $\rightarrow$ properp (x)

THEOREM: alistp-nlistp
($x \simeq$ **nil**) $\rightarrow$ (alistp (x) = (x = **nil**))

THEOREM: alistp-cons
alistp (cons (a, x)) = (listp (a) $\land$ alistp (x))

EVENT: Disable alistp.

THEOREM: alistp-append
alistp (append (x, y)) = (alistp (fix-properp (x)) $\land$ alistp (y))

DEFINITION:
domain (map)
= **if** listp (map)
then if listp (car (map)) **then** cons (car (car (map)), domain (cdr (map)))
else domain (cdr (map)) **endif**
else nil endif

THEOREM: properp-domain
properp (domain (map))

THEOREM: domain-append
domain (append (x, y)) = append (domain (x), domain (y))

THEOREM: domain-nlistp
($map \simeq$ **nil**) $\rightarrow$ (domain (map) = **nil**)

THEOREM: domain-cons
domain (cons (a, map))
= **if** listp (a) **then** cons (car (a), domain (map))
else domain (map) **endif**

THEOREM: member-domain-sufficiency
(cons (a, x) $\in y$) $\rightarrow$ ($a \in$ domain (y))

THEOREM: subsetp-domain
subsetp (x, y) $\rightarrow$ subsetp (domain (x), domain (y))

EVENT: Disable domain.

DEFINITION:
range (map)
= **if** listp (map)
then if listp (car (map)) **then** cons (cdr (car (map)), range (cdr (map)))
else range (cdr (map)) **endif**
else nil endif

THEOREM: properp-range
properp (range (map))

THEOREM: range-append
range (append ($s1$, $s2$)) = append (range ($s1$), range ($s2$))

THEOREM: range-nlistp
($map \simeq$ **nil**) $\rightarrow$ (range (map) = **nil**)

THEOREM: range-cons
range (cons (a, map))
= **if** listp (a) **then** cons (cdr (a), range (map))
else range (map) **endif**

EVENT: Disable range.

```
;; BOUNDP has been eliminated in favor of membership in domain.
;; Notice that I have to talk about things like disjointness of
;; domains anyhow.  New definition body would be (member x (domain map)).
```

```
;(defn boundp (x map)
;  (if (listp map)
;      (if (listp (car map))
;          (if (equal x (caar map))
;              t
;            (boundp x (cdr map)))
;        (boundp x (cdr map)))
;    f))
```

DEFINITION:
value $(x,\ map)$
$=$ **if** listp (map)
then if listp (car $(map)) \wedge (x = \text{caar}(map))$ **then** cdr (map)
else value $(x,\ \text{cdr}(map))$ **endif**
else 0 **endif**

THEOREM: value-nlistp
$(map \simeq \textbf{nil}) \rightarrow (\text{value}(x,\ map) = 0)$

THEOREM: value-cons
value $(x,\ \text{cons}(pair,\ map))$
$=$ **if** listp $(pair) \wedge (x = \text{car}(pair))$ **then** cdr $(pair)$
else value $(x,\ map)$ **endif**

EVENT: Disable value.

DEFINITION:
bind $(x,\ v,\ map)$
$=$ **if** listp (map)
then if listp (car (map))
then if $x = \text{caar}(map)$ **then** cons (cons $(x,\ v)$, cdr (map))
else cons (car (map), bind $(x,\ v,\ \text{cdr}(map))$) **endif**
else cons (car (map), bind $(x,\ v,\ \text{cdr}(map))$) **endif**
else cons (cons $(x,\ v)$, **nil**) **endif**

DEFINITION:
rembind $(x,\ map)$
$=$ **if** listp (map)
then if listp (car (map))
then if $x = \text{caar}(map)$ **then** cdr (map)
else cons (car (map), rembind $(x,\ \text{cdr}(map))$) **endif**
else cons (car (map), rembind $(x,\ \text{cdr}(map))$) **endif**
else nil endif

Definition:
invert (map)
= **if** listp (map)
then if listp (car (map))
then cons (cons (cdr (car (map)), car (car (map))), invert (cdr (map)))
else invert (cdr (map)) **endif**
else nil endif

Theorem: properp-invert
properp (invert (map))

Theorem: invert-nlistp
($map \simeq$ **nil**) $\rightarrow$ (invert (map) = **nil**)

Theorem: invert-cons
invert (cons ($pair$, map))
= **if** listp ($pair$) **then** cons (cons (cdr ($pair$), car ($pair$)), invert (map))
else invert (map) **endif**

Theorem: value-invert-not-member-of-domain
(($g \in$ range (sg)) $\wedge$ disjoint (domain (s), domain (sg)))
$\rightarrow$ (value (g, invert (sg)) $\notin$ domain (s))

Event: Disable invert.

Definition: mapping (map) = (alistp (map) $\wedge$ setp (domain (map)))

```
;; For when we disable mapping:
```

Theorem: mapping-implies-alistp
mapping (map) $\rightarrow$ alistp (map)

Theorem: mapping-implies-setp-domain
mapping (map) $\rightarrow$ setp (domain (map))

Definition:
restrict (s, $new\text{-}domain$)
= **if** listp (s)
then if listp (car (s)) $\wedge$ (caar (s) $\in$ $new\text{-}domain$)
then cons (car (s), restrict (cdr (s), $new\text{-}domain$))
else restrict (cdr (s), $new\text{-}domain$) **endif**
else nil endif

Definition:
co-restrict (s, $new\text{-}domain$)

$=$ **if** listp (s)
then if listp (car (s)) $\wedge$ (caar (s) $\notin$ *new-domain*)
then cons (car (s), co-restrict (cdr (s), *new-domain*))
else co-restrict (cdr (s), *new-domain*) **endif**
else nil endif

Event: Let us define the theory *alist-defns* to consist of the following events: alistp, domain, range, value, bind, rembind, invert, mapping, restrict, co-restrict.

```
;;;;; alist lemmas

; DOMAIN

;; The following was proved in the course of the final run through
;; the generalization proof.  The one after it isn't needed but
;; seems like it's worth proving too.  Actually now I see that
;; some other lemmas are now obsolete, so I'll put these both
;; early in the file and delete the others.
```

Theorem: domain-restrict
domain (restrict (s, *dom*)) $=$ intersection (domain (s), *dom*)

Theorem: domain-co-restrict
domain (co-restrict (s, *dom*)) $=$ set-diff (domain (s), *dom*)

Theorem: domain-bind
domain (bind (x, v, *map*))
$=$ **if** $x \in$ domain (*map*) **then** domain (*map*)
else append (domain (*map*), list (x)) **endif**

Theorem: domain-rembind
domain (rembind (x, *map*)) $=$ delete (x, domain (*map*))

Theorem: domain-invert
domain (invert (*map*)) $=$ range (*map*)

```
; RANGE
```

Theorem: range-invert
range (invert (*map*)) $=$ domain (*map*)

```
; BOUNDP
```

THEOREM: boundp-bind
$(x \in \text{domain}(\text{bind}(y, v, map))) = ((x = y) \lor (x \in \text{domain}(map)))$

THEOREM: boundp-rembind
$\text{mapping}(map)$
$\rightarrow \quad ((x \in \text{domain}(\text{rembind}(y, map)))$
$\quad = \quad \textbf{if } x = y \textbf{ then f}$
$\quad\quad \textbf{else } x \in \text{domain}(map) \textbf{ endif})$

THEOREM: boundp-subsetp
$(\text{subsetp}(map1, map2) \land (name \in \text{domain}(map1)))$
$\rightarrow \quad (name \in \text{domain}(map2))$

THEOREM: disjoint-domain-singleton
$(\text{disjoint}(\text{domain}(s), \text{list}(x)) = (x \notin \text{domain}(s)))$
$\land \quad (\text{disjoint}(\text{list}(x), \text{domain}(s)) = (x \notin \text{domain}(s)))$

THEOREM: boundp-value-invert
$(x \in \text{range}(map)) \rightarrow (\text{value}(x, \text{invert}(map)) \in \text{domain}(map))$

```
; VALUE
```

THEOREM: value-when-not-bound
$(name \notin \text{domain}(map)) \rightarrow (\text{value}(name, map) = 0)$

THEOREM: value-bind
$\text{value}(x, \text{bind}(y, v, map))$
$= \quad \textbf{if } x = y \textbf{ then } v$
$\quad \textbf{else } \text{value}(x, map) \textbf{ endif}$

THEOREM: value-rembind
$\text{mapping}(map)$
$\rightarrow \quad (\text{value}(x, \text{rembind}(y, map))$
$\quad = \quad \textbf{if } x = y \textbf{ then } 0$
$\quad\quad \textbf{else } \text{value}(x, map) \textbf{ endif})$

THEOREM: value-append
$\text{value}(x, \text{append}(s1, s2))$
$= \quad \textbf{if } x \in \text{domain}(s1) \textbf{ then } \text{value}(x, s1)$
$\quad \textbf{else } \text{value}(x, s2) \textbf{ endif}$

THEOREM: value-value-invert
$((x \in \text{range}(s)) \land \text{mapping}(s)) \rightarrow (\text{value}(\text{value}(x, \text{invert}(s)), s) = x)$

```
; MAPPING
```

THEOREM: mapping-append
mapping (append ($s1$, $s2$))
$=$ (disjoint (domain ($s1$), domain ($s2$))
$\land$ mapping (fix-properp ($s1$))
$\land$ mapping ($s2$))

EVENT: Disable mapping.

```
;; RESTRICT and CO-RESTRICT
```

THEOREM: alistp-restrict
alistp (restrict (s, r))

THEOREM: alistp-co-restrict
alistp (co-restrict (s, r))

THEOREM: value-restrict
(($a \in r$) $\land$ ($a \in$ domain (s)))
$\rightarrow$ (value (a, restrict (s, r)) $=$ value (a, s))

THEOREM: value-co-restrict
(($a \notin r$) $\land$ ($a \in$ domain (s)))
$\rightarrow$ (value (a, co-restrict (s, r)) $=$ value (a, s))

THEOREM: mapping-restrict
mapping (s) $\rightarrow$ mapping (restrict (s, x))

THEOREM: mapping-co-restrict
mapping (s) $\rightarrow$ mapping (co-restrict (s, x))

EVENT: Disable restrict.

EVENT: Disable co-restrict.

EVENT: Make the library `"alists"`.

Index