```
#|

 Copyright (C) 1994 by Computational Logic, Inc.  All Rights Reserved.

 This script is hereby placed in the public domain, and therefore unlimited
 editing and redistribution is permitted.

 NO WARRANTY

 Computational Logic, Inc. PROVIDES ABSOLUTELY NO WARRANTY.  THE EVENT SCRIPT
 IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED,
 INCLUDING, BUT NOT LIMITED TO, ANY IMPLIED WARRANTIES OF MERCHANTABILITY AND
 FITNESS FOR A PARTICULAR PURPOSE.  THE ENTIRE RISK AS TO THE QUALITY AND
 PERFORMANCE OF THE SCRIPT IS WITH YOU.  SHOULD THE SCRIPT PROVE DEFECTIVE, YOU
 ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

 IN NO EVENT WILL Computational Logic, Inc. BE LIABLE TO YOU FOR ANY DAMAGES,
 ANY LOST PROFITS, LOST MONIES, OR OTHER SPECIAL, INCIDENTAL OR CONSEQUENTIAL
 DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THIS SCRIPT (INCLUDING BUT
 NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES
 SUSTAINED BY THIRD PARTIES), EVEN IF YOU HAVE ADVISED US OF THE POSSIBILITY OF
 SUCH DAMAGES, OR FOR ANY CLAIM BY ANY OTHER PARTY.
|#

;; Matt Kaufmann
```

EVENT: Start with the initial **nqthm** theory.

```
;; These are events in support of the paper ``A Mechanically-Checked
;; Correctness Proof for Generalization in the Presence of Free
;; Variables,'' Journal of Automated Reasoning, Vol. 7, 1991.

;; This events file contains no DEFN-SK events, but instead contains two
;; DCLs and two ADD-AXIOMs from a DEFN-SK event that I commented out.

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; sets.events file
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; Requires deftheory enhancement.
;; Requires only ground-zero theory, nqthm mode.

;; Sets; Matt Kaufmann, Dec. 1989, revised March 1990.  The first few events
```

```
;; are some basic events about lists.  I'll take the approach that all these
;; basic functions will be disabled once enough algebraic properties have
;; been proved.

;; Theories:

;; (deftheory set-defns
;;  (length properp fix-properp member append subsetp delete
;;          disjoint intersection set-diff setp))
```

DEFINITION:
length (x)
= **if** listp (x) **then** $1 +$ length (cdr (x))
else 0 **endif**

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

THEOREM: length-cons
length (cons $(a,\ x)$) $= (1 +$ length $(x))$

THEOREM: length-append
length (append $(x,\ y)$) $= (\text{length}\,(x) + \text{length}\,(y))$

EVENT: Disable length.

THEOREM: append-assoc
append (append $(x,\ y),\ z$) $=$ append $(x,$ append $(y,\ z))$

THEOREM: member-cons
$(a \in \text{cons}\,(x,\ l)) = ((a = x) \vee (a \in l))$

THEOREM: member-nlistp
$(l \simeq \textbf{nil}) \rightarrow (a \notin l)$

EVENT: Disable member.

DEFINITION:
subsetp $(x,\ y)$
= **if** $x \simeq$ **nil then t**
else $(\text{car}\,(x) \in y) \wedge$ subsetp (cdr $(x),\ y$) **endif**

DEFINITION:
subsetp-wit $(x,\ y)$
$=$ **if** $x \simeq$ **nil then t**
elseif car $(x) \in y$ **then** subsetp-wit (cdr (x), y)
else car (x) **endif**

THEOREM: subsetp-wit-witnesses
subsetp $(x,\ y)$
$=$ $(\neg\ ((\text{subsetp-wit}\ (x,\ y) \in x) \wedge (\text{subsetp-wit}\ (x,\ y) \notin y)))$

THEOREM: subsetp-wit-witnesses-general-1
$((\text{subsetp-wit}\ (x,\ y) \notin x) \wedge (a \in x)) \rightarrow (a \in y)$

THEOREM: subsetp-wit-witnesses-general-2
$((\text{subsetp-wit}\ (x,\ y) \in y) \wedge (a \in x)) \rightarrow (a \in y)$

EVENT: Disable subsetp-wit-witnesses.

EVENT: Disable subsetp-wit-witnesses-general-1.

EVENT: Disable subsetp-wit-witnesses-general-2.

THEOREM: subsetp-cons-1
subsetp $(\text{cons}\ (a,\ x),\ y) = ((a \in y) \wedge \text{subsetp}\ (x,\ y))$

THEOREM: subsetp-cons-2
subsetp $(l,\ m) \rightarrow \text{subsetp}\ (l,\ \text{cons}\ (a,\ m))$

THEOREM: subsetp-reflexivity
subsetp $(x,\ x)$

THEOREM: cdr-subsetp
subsetp $(\text{cdr}\ (x),\ x)$

THEOREM: member-subsetp
$((x \in y) \wedge \text{subsetp}\ (y,\ z)) \rightarrow (x \in z)$

THEOREM: subsetp-is-transitive
$(\text{subsetp}\ (x,\ y) \wedge \text{subsetp}\ (y,\ z)) \rightarrow \text{subsetp}\ (x,\ z)$

THEOREM: member-append
$(a \in \text{append}\ (x,\ y)) = ((a \in x) \vee (a \in y))$

THEOREM: subsetp-append
subsetp $(\text{append}\ (x,\ y),\ z) = (\text{subsetp}\ (x,\ z) \wedge \text{subsetp}\ (y,\ z))$

THEOREM: subsetp-of-append-sufficiency
$(\text{subsetp}(a,\ b) \vee \text{subsetp}(a,\ c)) \rightarrow \text{subsetp}(a,\ \text{append}(b,\ c))$

THEOREM: subsetp-nlistp
$(x \simeq \textbf{nil}) \rightarrow (\text{subsetp}(x,\ y) \wedge (\text{subsetp}(y,\ x) = (y \simeq \textbf{nil})))$

THEOREM: subsetp-cons-not-member
$(z \notin x) \rightarrow (\text{subsetp}(x,\ \text{cons}(z,\ v)) = \text{subsetp}(x,\ v))$

EVENT: Disable subsetp.

```
;;;;; Other set-theoretic and list-theoretic definitions, and properp observations.
```

DEFINITION:
properp (x)
= **if** listp (x) **then** properp (cdr (x))
else x = **nil endif**

DEFINITION:
fix-properp (x)
= **if** listp (x) **then** cons (car (x), fix-properp (cdr (x)))
else nil endif

THEOREM: properp-fix-properp
properp (fix-properp (x))

THEOREM: fix-properp-properp
properp $(x) \rightarrow$ (fix-properp $(x) = x$)

THEOREM: properp-cons
properp (cons $(x,\ y)$) = properp (y)

THEOREM: properp-nlistp
$(x \simeq \textbf{nil}) \rightarrow$ (properp $(x) = (x = \textbf{nil})$)

THEOREM: fix-properp-cons
fix-properp (cons $(x,\ y)$) = cons $(x,$ fix-properp $(y))$

THEOREM: fix-properp-nlistp
$(x \simeq \textbf{nil}) \rightarrow$ (fix-properp $(x) = \textbf{nil}$)

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

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

THEOREM: append-nil
append $(x,$ **nil**$) =$ fix-properp (x)

DEFINITION:
delete $(x,\ l)$
$=$ **if** listp (l)
then if $x =$ car (l) **then** cdr (l)
else cons (car (l), delete $(x,$ cdr $(l)))$ **endif**
else l **endif**

THEOREM: properp-delete
properp (delete $(x,\ l)) =$ properp (l)

DEFINITION:
disjoint $(x,\ y)$
$=$ **if** listp (x) **then** $(\text{car}\,(x) \notin y) \wedge$ disjoint (cdr $(x),\ y)$
else t endif

DEFINITION:
disjoint-wit $(x,\ y)$
$=$ **if** listp (x)
then if car $(x) \in y$ **then** car (x)
else disjoint-wit (cdr $(x),\ y)$ **endif**
else t endif

THEOREM: disjoint-wit-witnesses
disjoint $(x,\ y)$
$=$ $(\neg\ ((\text{disjoint-wit}\,(x,\ y) \in x) \wedge (\text{disjoint-wit}\,(x,\ y) \in y)))$

EVENT: Disable disjoint-wit.

EVENT: Disable disjoint-wit-witnesses.

DEFINITION:
intersection $(x,\ y)$
$=$ **if** listp (x)
then if car $(x) \in y$ **then** cons (car (x), intersection (cdr $(x),\ y)$)
else intersection (cdr $(x),\ y)$ **endif**
else nil endif

THEOREM: properp-intersection
properp (intersection $(x,\ y)$)

DEFINITION:
set-diff $(x,\ y)$
$=$ **if** listp (x)
then if car $(x) \in y$ **then** set-diff (cdr (x), y)
else cons (car (x), set-diff (cdr (x), y)) **endif**
else nil endif

THEOREM: properp-set-diff
properp (set-diff $(x,\ y)$)

DEFINITION:
setp (x)
$=$ **if** $\neg$ listp (x) **then** $x =$ **nil**
else (car $(x) \notin$ cdr (x)) $\wedge$ setp (cdr (x)) **endif**

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

EVENT: Disable properp.

EVENT: Let us define the theory *set-defns* to consist of the following events: length, properp, fix-properp, member, append, subsetp, delete, disjoint, intersection, set-diff, setp, properp.

```
;; Set theory lemmas
```

THEOREM: delete-cons
delete $(a,$ cons $(b,\ x))$
$=$ **if** $a = b$ **then** x
else cons $(b,$ delete $(a,\ x))$ **endif**

THEOREM: delete-nlistp
$(x \simeq$ **nil**$) \rightarrow$ (delete $(a,\ x) = x$)

THEOREM: listp-delete
listp (delete $(x,\ l)$)
$=$ **if** listp (l) **then** $(x \neq$ car $(l)) \vee$ listp (cdr (l))
else f endif

THEOREM: delete-non-member
$(x \notin y) \rightarrow$ (delete $(x,\ y) = y$)

THEOREM: delete-delete
delete $(y,$ delete $(x,\ z)) =$ delete $(x,$ delete $(y,\ z))$

THEOREM: member-delete
$\text{setp}(x) \rightarrow ((a \in \text{delete}(b, x)) = ((a \neq b) \wedge (a \in x)))$

THEOREM: setp-delete
$\text{setp}(x) \rightarrow \text{setp}(\text{delete}(a, x))$

EVENT: Disable delete.

THEOREM: disjoint-cons-1
$\text{disjoint}(\text{cons}(a, x), y) = ((a \notin y) \wedge \text{disjoint}(x, y))$

THEOREM: disjoint-cons-2
$\text{disjoint}(x, \text{cons}(a, y)) = ((a \notin x) \wedge \text{disjoint}(x, y))$

THEOREM: disjoint-nlistp
$((x \simeq \textbf{nil}) \vee (y \simeq \textbf{nil})) \rightarrow \text{disjoint}(x, y)$

THEOREM: disjoint-symmetry
$\text{disjoint}(x, y) = \text{disjoint}(y, x)$

THEOREM: disjoint-append-right
$\text{disjoint}(u, \text{append}(y, z)) = (\text{disjoint}(u, y) \wedge \text{disjoint}(u, z))$

THEOREM: disjoint-append-left
$\text{disjoint}(\text{append}(y, z), u) = (\text{disjoint}(y, u) \wedge \text{disjoint}(z, u))$

THEOREM: disjoint-non-member
$((a \in x) \wedge (a \in y)) \rightarrow (\neg\, \text{disjoint}(x, y))$

THEOREM: disjoint-subsetp-monotone-second
$(\text{subsetp}(y, z) \wedge \text{disjoint}(x, z)) \rightarrow \text{disjoint}(x, y)$

THEOREM: subsetp-disjoint-2
$(\text{subsetp}(x, y) \wedge \text{disjoint}(y, z)) \rightarrow \text{disjoint}(z, x)$

THEOREM: subsetp-disjoint-1
$(\text{subsetp}(x, y) \wedge \text{disjoint}(y, z)) \rightarrow \text{disjoint}(x, z)$

THEOREM: subsetp-disjoint-3
$(\text{subsetp}(x, y) \wedge \text{disjoint}(z, y)) \rightarrow \text{disjoint}(x, z)$

EVENT: Disable disjoint.

THEOREM: intersection-disjoint
$(\text{intersection}(x, y) = \textbf{nil}) = \text{disjoint}(x, y)$

THEOREM: intersection-nlistp
$((x \simeq \mathbf{nil}) \vee (y \simeq \mathbf{nil})) \rightarrow (\text{intersection}\,(x,\ y) = \mathbf{nil})$

THEOREM: member-intersection
$(a \in \text{intersection}\,(x,\ y)) = ((a \in x) \wedge (a \in y))$

THEOREM: subsetp-intersection
$\text{subsetp}\,(x,\ \text{intersection}\,(y,\ z)) = (\text{subsetp}\,(x,\ y) \wedge \text{subsetp}\,(x,\ z))$

THEOREM: intersection-symmetry
$\text{subsetp}\,(\text{intersection}\,(x,\ y),\ \text{intersection}\,(y,\ x))$

THEOREM: intersection-cons-1
$\text{intersection}\,(\text{cons}\,(a,\ x),\ y)$
$=$ **if** $a \in y$ **then** $\text{cons}\,(a,\ \text{intersection}\,(x,\ y))$
else $\text{intersection}\,(x,\ y)$ **endif**

THEOREM: intersection-cons-2
$(a \notin y) \rightarrow (\text{intersection}\,(y,\ \text{cons}\,(a,\ x)) = \text{intersection}\,(y,\ x))$

```
;; The following is needed because DISJOINT-INTERSECTION-COMMUTER,
;; added during polishing, caused the proof of
;; DISJOINT-DOMAIN-CO-RESTRICT (in "alists.events") to fail.
```

THEOREM: intersection-cons-3
$(w \in x)$
$\rightarrow$ $(\text{subsetp}\,(\text{intersection}\,(y,\ \text{cons}\,(w,\ z)),\ x)$
$=$ $\text{subsetp}\,(\text{intersection}\,(y,\ z),\ x))$

THEOREM: intersection-cons-subsetp
$\text{subsetp}\,(\text{intersection}\,(x,\ y),\ \text{intersection}\,(x,\ \text{cons}\,(a,\ y)))$

THEOREM: subsetp-intersection-left-1
$\text{subsetp}\,(\text{intersection}\,(x,\ y),\ x)$

THEOREM: subsetp-intersection-left-2
$\text{subsetp}\,(\text{intersection}\,(x,\ y),\ y)$

THEOREM: subsetp-intersection-sufficiency-1
$\text{subsetp}\,(y,\ z) \rightarrow \text{subsetp}\,(\text{intersection}\,(x,\ y),\ z)$

THEOREM: subsetp-intersection-sufficiency-2
$\text{subsetp}\,(y,\ z) \rightarrow \text{subsetp}\,(\text{intersection}\,(y,\ x),\ z)$

THEOREM: intersection-associative
$\text{intersection}\,(\text{intersection}\,(x,\ y),\ z) = \text{intersection}\,(x,\ \text{intersection}\,(y,\ z))$

THEOREM: intersection-elimination
subsetp $(x,\ y) \rightarrow$ (intersection $(x,\ y)$ = fix-properp (x))

THEOREM: length-intersection
length $(x) \not< $ length (intersection $(x,\ y)$)

THEOREM: subsetp-intersection-member
(subsetp (intersection $(x,\ y)$, $z) \wedge (a \notin z)$)
$\rightarrow$ $(((a \in x) \rightarrow (a \notin y)) \wedge ((a \in y) \rightarrow (a \notin x)))$

```
;; The following wasn't needed in the proof about generalization, but it's a nice rule.
```

THEOREM: intersection-append
intersection (append $(x,\ y)$, z) = append (intersection $(x,\ z)$, intersection $(y,\ z)$)

```
;; I'd rather just prove that intersection distributes over append on
;; the right but that isn't true.  Congruence relations would probably
;; help a lot with that problem.  In the meantime, I content myself
;; with the following.
```

THEOREM: disjoint-intersection-append
disjoint (x, intersection (y, append ($z1$, $z2$)))
= (disjoint (x, intersection (y, $z1$)) $\wedge$ disjoint (x, intersection (y, $z2$)))

```
;; See comment just above DISJOINT-INTERSECTION-APPEND
```

THEOREM: subsetp-intersection-append
subsetp (intersection (u, append $(x,\ y)$), z)
= (subsetp (intersection $(u,\ x)$, $z) \wedge$ subsetp (intersection $(u,\ y)$, z))

THEOREM: subsetp-intersection-elimination-lemma
(subsetp $(y,\ x) \wedge (\neg$ subsetp $(y,\ z)))$
$\rightarrow$ $(\neg$ subsetp (intersection $(x,\ y)$, $z))$

THEOREM: subsetp-intersection-elimination
subsetp $(y,\ x) \rightarrow$ (subsetp (intersection $(x,\ y)$, $z) \leftrightarrow$ subsetp $(y,\ z)$)

THEOREM: disjoint-intersection
disjoint (intersection $(x,\ y)$, z) = disjoint (x, intersection $(y,\ z)$)

THEOREM: subsetp-intersection-monotone-1
(subsetp (intersection $(x,\ y)$, $z) \wedge$ subsetp $(x1,\ x)$)
$\rightarrow$ subsetp (intersection $(x1,\ y)$, z)

```
;; The lemma SUBSETP-INTERSECTION-MONOTONE-2 below was added during
;; polishing of the final proof in "generalize.events", since the
;; lemma immediately above wasn't enough at that point.  Actually
;; I realized at this point that intersection commutes from the point
;; of view of subsetp:
```

THEOREM: subsetp-intersection-commuter
subsetp (intersection (x, y), z) = subsetp (intersection (y, x), z)

THEOREM: subsetp-intersection-monotone-2
(subsetp (intersection (y, x), z) ∧ subsetp (*x1*, x))
→ subsetp (intersection (*x1*, y), z)

THEOREM: disjoint-intersection-commuter
disjoint (x, intersection (y, z)) = disjoint (x, intersection (z, y))

THEOREM: disjoint-intersection3
disjoint (*free*, intersection (*vars*, x))
→ (intersection (x, intersection (*vars*, *free*)) = **nil**)

EVENT: Disable intersection.

THEOREM: member-set-diff
(a ∈ set-diff (y, z)) = ((a ∈ y) ∧ (a ∉ z))

THEOREM: subsetp-set-diff-1
subsetp (set-diff (x, y), x)

THEOREM: disjointp-set-diff
disjoint (set-diff (x, y), y)

THEOREM: subsetp-set-diff-2
subsetp (x, set-diff (y, z)) = (subsetp (x, y) ∧ disjoint (x, z))

THEOREM: set-diff-cons
set-diff (cons (a, x), y)
= **if** a ∈ y **then** set-diff (x, y)
else cons (a, set-diff (x, y)) **endif**

THEOREM: set-diff-nlistp
(x ≃ **nil**) → (set-diff (x, y) = **nil**)

```
;; The following was discovered during final polishing, for the
;; proof of MAIN-HYPS-RELIEVED-6-FIRST.
```

THEOREM: disjoint-set-diff-general
disjoint (x, set-diff (y, z)) = subsetp (intersection (x, y), z)

```
;; No longer relevant:
;(prove-lemma disjoint-set-diff-subsetp (rewrite)
;  (implies (and (disjoint x (set-diff y z))
;                (subsetp z z1))
;           (disjoint x (set-diff y z1)))
;  ((use (disjoint-wit-witnesses (y (set-diff y z1))))
;   (disable member-set-diff set-diff)))

;; Instead of the following I'll prove the corresponding (in light of
;; DISJOINT-SET-DIFF-GENERAL) fact INTERSECTION-X-X about intersection.
;(prove-lemma disjoint-set-diff (rewrite)
;  (disjoint x (set-diff y x)))
```

THEOREM: intersection-subsetp-identity
(properp (x) ∧ subsetp (x, y)) → (intersection (x, y) = x)

THEOREM: intersection-x-x
properp (x) → (intersection (x, x) = x)

THEOREM: subsetp-set-diff-mononone-2
subsetp (set-diff (x, append (y, z)), set-diff (x, z))

THEOREM: subsetp-set-diff-monotone-second
subsetp (set-diff (x, y), set-diff (x, z)) = subsetp (intersection (x, z), y)

THEOREM: set-diff-nil
set-diff (x, **nil**) = fix-properp (x)

THEOREM: set-diff-cons-non-member-1
(a ∉ x) → (set-diff (x, cons (a, y)) = set-diff (x, y))

THEOREM: length-intersection-set-diff
length (x) = (length (set-diff (x, y)) + length (intersection (x, y)))

THEOREM: length-set-diff-opener
length (set-diff (x, y)) = (length (x) − length (intersection (x, y)))

THEOREM: listp-set-diff
listp (set-diff (x, y)) = (¬ subsetp (x, y))

```
;; Here is a messy lemma about disjoint and such
```

THEOREM: disjoint-intersection-set-diff-intersection
disjoint $(x,$ intersection $(y,$ set-diff $(z,$ intersection $(y,\ x))))$

EVENT: Disable set-diff.

THEOREM: member-fix-properp
$(a \in \text{fix-properp}\,(x)) = (a \in x)$

THEOREM: setp-append
setp (append $(x,\ y)$) = (disjoint $(x,\ y) \wedge$ setp (fix-properp $(x)) \wedge$ setp $(y))$

THEOREM: setp-cons
setp (cons $(a,\ x)$) = $((a \notin x) \wedge$ setp $(x))$

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

DEFINITION:
make-set (l)
= **if** $\neg$ listp (l) **then nil**
elseif car $(l) \in$ cdr (l) **then** make-set (cdr $(l))$
else cons (car (l), make-set (cdr $(l))$) **endif**

THEOREM: make-set-preserves-member
$(x \in \text{make-set}\,(l)) = (x \in l)$

THEOREM: make-set-preserves-subsetp-1
subsetp (make-set (x), make-set (y)) = subsetp $(x,\ y)$

THEOREM: make-set-preserves-subsetp-2
subsetp $(x,$ make-set $(y)) =$ subsetp $(x,\ y)$

THEOREM: make-set-preserves-subsetp-3
subsetp (make-set $(x),\ y) =$ subsetp $(x,\ y)$

THEOREM: make-set-gives-setp
setp (make-set $(x))$

THEOREM: make-set-set-diff
make-set (set-diff $(x,\ y)) =$ set-diff (make-set (x), make-set $(y))$

THEOREM: set-diff-make-set
set-diff $(x,$ make-set $(y)) =$ set-diff $(x,\ y)$

THEOREM: listp-make-set
listp (make-set $(x)) =$ listp (x)

EVENT: Disable setp.

```
;;;;;; The following were proved in the course of the final run
;;;;;; through the generalization proof.  There are a couple or
;;;;;; so noted above here, too.
```

THEOREM: set-diff-append
set-diff $(x,$ append $(y,\ z)) =$ set-diff (set-diff $(x,\ z),\ y)$

THEOREM: length-set-diff-leq
length $(x) \not<$ length (set-diff $(x,\ y)$)

THEOREM: lessp-length
listp $(x) \rightarrow (0 <$ length $(x))$

THEOREM: listp-intersection
listp (intersection $(x,\ y)) =$ ($\neg$ disjoint $(x,\ y)$)

THEOREM: length-set-diff-lessp
($\neg$ disjoint $(x,\ new)) \rightarrow$ (length (set-diff $(x,\ new)) <$ length $(x))$

THEOREM: disjoint-implies-empty-intersection
disjoint $(x,\ y) \rightarrow$ (intersection $(x,\ y) =$ **nil**)

```
;; The following lemma DISJOINT-INTERSECTION3-MIDDLE is needed for the
;; proof of ALL-VARS-DISJOINT-OR-SUBSETP-GEN-CLOSURE in
;; generalize.events.  I think I could avoid lemmas like this one
;; INTERSECTION were actually commutative-associative (in which case
;; I'd get rid of disjoint and rely on normalization).
```

THEOREM: disjoint-intersection3-middle
disjoint $(y,$ intersection $(x,\ z))$
$\rightarrow$ (intersection $(x,$ intersection $(y,\ z)) =$ **nil**)

```
;; Maybe I should redo the notion of disjoint sometime, perhaps using
;; the fact that intersection is commutative and associative when it's
;; equated with nil.
```

THEOREM: disjoint-subsetp-hack
(disjoint $(x,$ intersection $(u,\ v)) \wedge$ subsetp $(w,\ x))$
$\rightarrow$ disjoint $(u,$ intersection $(w,\ v))$

THEOREM: subsetp-set-diff-sufficiency
subsetp $(x,\ y) \rightarrow$ subsetp (set-diff $(x,\ z),\ y$)

```
;; The following lemma SETP-INTERSECTION-SUFFICIENCY is needed for
;; MAPPING-RESTRICT from "alists.events", because (I believe)
;; DOMAIN-RESTRICT, which was added during polishing, changed the
;; course of the previous proof.  Similarly for
;; SETP-SET-DIFF-SUFFICIENCY and the lemma MAPPING-CO-RESTRICT.
```

THEOREM: setp-intersection-sufficiency
setp $(x) \rightarrow$ setp (intersection $(x,\ y)$)

THEOREM: setp-set-diff-sufficiency
setp $(x) \rightarrow$ setp (set-diff $(x,\ y)$)

```
;; The definition of FIX-PROPERP was also added in polishing because
;; of a problem with the proof of GEN-CLOSURE-ACCEPT in
;; "generalize.events".  Here are a couple of lemmas about it that
;; might or might not be useful; all other lemmas about it above, and
;; the definition, were added during polishing.
```

EVENT: Disable fix-properp.

THEOREM: subsetp-fix-properp-1
subsetp (fix-properp $(x),\ y$) = subsetp $(x,\ y)$

THEOREM: subsetp-fix-properp-2
subsetp $(x,$ fix-properp $(y))$ = subsetp $(x,\ y)$

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; alists.events file
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; Requires deftheory enhancement.
;; Requires 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)) $\wedge$ 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 $(a) \wedge$ alistp $(x))$

EVENT: Disable alistp.

THEOREM: alistp-append
alistp (append $(x,\ y)) =$ (alistp (fix-properp $(x)) \wedge$ 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
$(\text{cons}\,(a,\ x) \in y) \rightarrow (a \in \text{domain}\,(y))$

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* ≃ **nil**) → (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*)) ∧ (*x* = caar (*map*)) **then** cdar (*map*)
    else value (*x*, cdr (*map*)) **endif**
  else 0 **endif**

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

THEOREM: value-cons
value $(x,$ cons $(pair,\ map))$
$=$ **if** listp $(pair) \wedge (x =$ 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 $x =$ caar (map) **then** cons (cons $(x,\ v)$, cdr (map))
else cons (car (map), bind $(x,\ v,$ cdr $(map)))$ **endif**
else cons (car (map), bind $(x,\ v,$ cdr $(map)))$ **endif**
else cons (cons $(x,\ v)$, **nil**) **endif**

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

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* ∈ range (*sg*)) ∧ disjoint (domain (*s*), domain (*sg*)))
→ (value (*g*, invert (*sg*)) ∉ domain (*s*))

EVENT: Disable invert.

DEFINITION: mapping (*map*) = (alistp (*map*) ∧ setp (domain (*map*)))

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

THEOREM: mapping-implies-alistp
mapping (*map*) → alistp (*map*)

THEOREM: mapping-implies-setp-domain
mapping (*map*) → setp (domain (*map*))

DEFINITION:
restrict (*s*, *new-domain*)
= **if** listp (*s*)
then if listp (car (*s*)) ∧ (caar (*s*) ∈ *new-domain*)
then cons (car (*s*), restrict (cdr (*s*), *new-domain*))
else restrict (cdr (*s*), *new-domain*) **endif**
else nil endif

DEFINITION:
co-restrict (*s*, *new-domain*)
= **if** listp (*s*)
then if listp (car (*s*)) ∧ (caar (*s*) ∉ *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* ∈ 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* ∈ domain (bind (*y*, *v*, *map*))) = ((*x* = *y*) ∨ (*x* ∈ domain (*map*)))

THEOREM: boundp-rembind
mapping (*map*)
→ ((*x* ∈ domain (rembind (*y*, *map*)))
= **if** *x* = *y* **then f**
else *x* ∈ domain (*map*) **endif**)

THEOREM: boundp-subsetp
(subsetp (*map1*, *map2*) ∧ (*name* ∈ domain (*map1*)))
→ (*name* ∈ domain (*map2*))

THEOREM: disjoint-domain-singleton
(disjoint (domain (*s*), list (*x*)) = (*x* ∉ domain (*s*)))
∧ (disjoint (list (*x*), domain (*s*)) = (*x* ∉ domain (*s*)))

THEOREM: boundp-value-invert
(*x* ∈ range (*map*)) → (value (*x*, invert (*map*)) ∈ domain (*map*))

; VALUE

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

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

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

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

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

; MAPPING

THEOREM: mapping-append
$\text{mapping}(\text{append}(s1, s2))$
$=$ $(\text{disjoint}(\text{domain}(s1), \text{domain}(s2))$
$\wedge$ $\text{mapping}(\text{fix-properp}(s1))$
$\wedge$ $\text{mapping}(s2))$

EVENT: Disable mapping.

;; RESTRICT and CO-RESTRICT

THEOREM: alistp-restrict
$\text{alistp}(\text{restrict}(s, r))$

THEOREM: alistp-co-restrict
$\text{alistp}(\text{co-restrict}(s, r))$

THEOREM: value-restrict
$((a \in r) \wedge (a \in \text{domain}(s)))$
$\rightarrow \quad (\text{value}(a, \text{restrict}(s, r)) = \text{value}(a, s))$

THEOREM: value-co-restrict
$((a \notin r) \wedge (a \in \text{domain}(s)))$
$\rightarrow \quad (\text{value}(a, \text{co-restrict}(s, r)) = \text{value}(a, s))$

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

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

EVENT: Disable restrict.

EVENT: Disable co-restrict.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; terms.events file
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; Requires deftheory and constrain enhancments.
;; Requires sets and alists libraries.

;; This is a library of events about terms, including substitutions.
;; A TERMP is either a variable or the application of a function
;; symbol to a "proper list" of terms.  Variables and function symbols
;; are introduced with CONSTRAIN.

;; NOTE:  In functions like TERMP that have a flag, it seems to be
;; important to use T and F rather than, say, T and 'LIST.  That's
;; because otherwise, the "worse-than" heuristic will otherwise
;; prevent some necessary backchaining in cases where the hypothesis
;; to be relieved is of the form (TERMP 'LIST ...) and an "ancestor"
;; is of the form (TERMP T ...).

;; Definitions:

;; (deftheory term-defns
;;  (variablep-intro variable-listp termp function-symbol-intro all-vars))

;; (deftheory substitution-defns
;;   (instance var-substp compose apply-to-subst subst
```

```
;;    nullify-subst ;; returns a substitution whose range has no variables
;;    ))
```

CONSERVATIVE AXIOM: variablep-intro
$(\text{listp}\,(x) \rightarrow (\neg\ \text{variablep}\,(x)))$
$\wedge\quad (\text{truep}\,(\text{variablep}\,(x)) \vee \text{falsep}\,(\text{variablep}\,(x)))$

Simultaneously, we introduce the new function symbol *variablep*.

DEFINITION:
variable-listp (x)
$=$ **if** listp (x) **then** variablep $(\text{car}\,(x)) \wedge$ variable-listp $(\text{cdr}\,(x))$
else $x =$ **nil endif**

THEOREM: variable-listp-implies-properp
variable-listp $(x) \rightarrow$ properp (x)

THEOREM: variable-listp-cons
variable-listp $(\text{cons}\,(a,\ x)) = (\text{variablep}\,(a) \wedge \text{variable-listp}\,(x))$

THEOREM: variable-nlistp
$(x \simeq \textbf{nil}) \rightarrow (\text{variable-listp}\,(x) = (x = \textbf{nil}))$

EVENT: Disable variable-listp.

CONSERVATIVE AXIOM: function-symbol-intro
function-symbol-p (FN)

Simultaneously, we introduce the new function symbols *function-symbol-p* and *fn*.

DEFINITION:
termp $(\mathit{flg},\ x)$
$=$ **if** *flg*
then if variablep (x) **then t**
elseif listp (x)
then function-symbol-p $(\text{car}\,(x)) \wedge$ termp $(\textbf{f},\ \text{cdr}\,(x))$
else f endif
elseif listp (x) **then** termp $(\textbf{t},\ \text{car}\,(x)) \wedge$ termp $(\textbf{f},\ \text{cdr}\,(x))$
else $x =$ **nil endif**

THEOREM: termp-list-cons
termp $(\textbf{f},\ \text{cons}\,(a,\ x)) = (\text{termp}\,(\textbf{t},\ a) \wedge \text{termp}\,(\textbf{f},\ x))$

THEOREM: termp-list-nlistp
$(x \simeq \mathbf{nil}) \rightarrow (\text{termp}\,(\mathbf{f},\, x) = (x = \mathbf{nil}))$

THEOREM: termp-t-cons
$\mathit{flg} \rightarrow (\text{termp}\,(\mathit{flg},\, \text{cons}\,(a,\, x)) = (\text{function-symbol-p}\,(a) \wedge \text{termp}\,(\mathbf{f},\, x)))$

THEOREM: termp-t-nlistp
$(\mathit{flg} \wedge (\neg\, \text{listp}\,(x))) \rightarrow (\text{termp}\,(\mathit{flg},\, x) = \text{variablep}\,(x))$

EVENT: Disable termp.

THEOREM: termp-list-implies-properp
$\text{termp}\,(\mathbf{f},\, x) \rightarrow \text{properp}\,(x)$

DEFINITION:
all-vars $(\mathit{flg},\, x)$
= **if** *flg*
then if variablep (x) **then** list (x)
elseif listp (x) **then** all-vars $(\mathbf{f},\, \text{cdr}\,(x))$
else nil endif
elseif listp (x) **then** append $(\text{all-vars}\,(\mathbf{t},\, \text{car}\,(x)),\, \text{all-vars}\,(\mathbf{f},\, \text{cdr}\,(x)))$
else nil endif

THEOREM: properp-all-vars
$\text{properp}\,(\text{all-vars}\,(\mathit{flg},\, x))$

THEOREM: all-vars-list-cons
$\text{all-vars}\,(\mathbf{f},\, \text{cons}\,(a,\, x)) = \text{append}\,(\text{all-vars}\,(\mathbf{t},\, a),\, \text{all-vars}\,(\mathbf{f},\, x))$

THEOREM: all-vars-t-cons
$\mathit{flg} \rightarrow (\text{all-vars}\,(\mathit{flg},\, \text{cons}\,(a,\, x)) = \text{all-vars}\,(\mathbf{f},\, x))$

```
;; Here is a hack to deal with the flags.
```

THEOREM: all-vars-subsetp-append-hack
$(\mathit{flg1} \wedge \mathit{flg2})$
$\rightarrow$ $(\text{subsetp}\,(\text{all-vars}\,(\mathit{flg1},\, x),\, \text{append}\,(\text{all-vars}\,(\mathit{flg2},\, x),\, y))$
$\wedge$ $\text{subsetp}\,(\text{all-vars}\,(\mathit{flg1},\, x),\, \text{append}\,(y,\, \text{all-vars}\,(\mathit{flg2},\, x))))$

```
;; The following is used later in the proof of MEMBER-PRESERVES-DISJOINT-ALL-VARS
;; and could conceivably be of use elsewhere.
```

THEOREM: all-vars-flg-boolean
$\mathit{flg} \rightarrow (\text{all-vars}\,(\mathit{flg},\, x) = \text{all-vars}\,(\mathbf{t},\, x))$

EVENT: Disable all-vars.

EVENT: Let us define the theory *term-defns* to consist of the following events: variablep-intro, variable-listp, termp, function-symbol-intro, all-vars.

```
;;;;; lemmas about termps
```

THEOREM: variable-listp-set-diff
variable-listp $(x) \rightarrow$ variable-listp (set-diff $(x,\ y)$)

THEOREM: all-vars-variablep
$(\mathit{flg} \wedge$ variablep $(x)) \rightarrow$ (all-vars $(\mathit{flg},\ x) =$ list (x))

THEOREM: member-variable-listp-implies-variablep
$((a \in x) \wedge$ variable-listp $(x)) \rightarrow$ variablep (a)

```
;; The following was proved in the course of the final run through the
;; generalization proof.  But in fact it's useful for the next result,
;; especially in the presence of domain-restrict -- so I don't need to
;; enable restrict there any longer.  Similarly for the lemma after
;; that.
```

THEOREM: variable-listp-intersection
(variable-listp $(x) \vee$ variable-listp (y))
$\rightarrow$ variable-listp (intersection $(x,\ y)$)

THEOREM: variable-listp-domain-restrict
variable-listp (domain $(s)) \rightarrow$ variable-listp (domain (restrict $(s,\ x)$))

THEOREM: variable-listp-domain-co-restrict
variable-listp (domain $(s)) \rightarrow$ variable-listp (domain (co-restrict $(s,\ x)$))

THEOREM: termp-range-restrict
termp (**f**, range $(s)) \rightarrow$ termp (**f**, range (restrict $(s,\ x)$))

THEOREM: termp-range-co-restrict
termp (**f**, range $(s)) \rightarrow$ termp (**f**, range (co-restrict $(s,\ x)$))

THEOREM: member-preserves-disjoint-all-vars-lemma
(disjoint $(y$, all-vars (**f**, $x)) \wedge (g \in x)) \rightarrow$ disjoint $(y$, all-vars (**t**, $g))$

THEOREM: member-preserves-disjoint-all-vars
$(\mathit{flg} \wedge$ disjoint $(y$, all-vars (**f**, $x)) \wedge (g \in x))$
$\rightarrow$ disjoint $(y$, all-vars $(\mathit{flg},\ g))$

THEOREM: member-all-vars-subsetp
$(\mathit{flg} \wedge (a \in x)) \rightarrow \text{subsetp}(\text{all-vars}(\mathit{flg}, a), \text{all-vars}(\mathbf{f}, x))$

THEOREM: all-vars-f-monotone
$\text{subsetp}(x, y) \rightarrow \text{subsetp}(\text{all-vars}(\mathbf{f}, x), \text{all-vars}(\mathbf{f}, y))$

```
;;;;; substitutions:  definitions
```

DEFINITION:
var-substp (s)
= (mapping (s) $\wedge$ variable-listp (domain (s)) $\wedge$ termp (**f**, range (s)))

DEFINITION:
subst (flg, s, x)
= **if** flg
then if $x \in$ domain (s) **then** value (x, s)
elseif variablep (x) **then** x
elseif listp (x) **then** cons (car (x), subst (**f**, s, cdr (x)))
else f endif
elseif listp (x) **then** cons (subst (**t**, s, car (x)), subst (**f**, s, cdr (x)))
else nil endif

DEFINITION:
apply-to-subst ($s1$, $s2$)
= **if** listp ($s2$)
then if listp (car ($s2$))
then cons (cons (caar ($s2$), subst (**t**, $s1$, cdar ($s2$))),
apply-to-subst ($s1$, cdr ($s2$)))
else apply-to-subst ($s1$, cdr ($s2$)) **endif**
else nil endif

DEFINITION: compose ($s1$, $s2$) = append (apply-to-subst ($s2$, $s1$), $s2$)

```
;; The following was in the original version, but isn't needed.
;;;  (defn-sk instance (flg term1 term2)
;;;    ;; term1 is an instance of term2
;;;    (exists one-way-unifier
;;;            (and (var-substp one-way-unifier)
;;;                 (equal term1 (subst flg one-way-unifier term2)))))

;;;;; substitution lemmas
```

THEOREM: subst-list-cons
subst (**f**, s, cons (a, x)) = cons (subst (**t**, s, a), subst (**f**, s, x))

THEOREM: subst-list-nlistp
$(x \simeq \mathbf{nil}) \rightarrow (\text{subst}\,(\mathbf{f},\ s,\ x) = \mathbf{nil})$

THEOREM: subst-t-variablep
$(\mathit{flg} \wedge \text{variablep}\,(x))$
$\rightarrow \quad (\text{subst}\,(\mathit{flg},\ s,\ x)$
$\quad = \quad \mathbf{if}\ x \in \text{domain}\,(s)\ \ \mathbf{then}\ \text{value}\,(x,\ s)$
$\quad\quad \mathbf{else}\ x\ \mathbf{endif})$

THEOREM: subst-t-non-variablep
$\mathit{flg} \rightarrow (\text{subst}\,(\mathit{flg},\ s,\ \text{cons}\,(\mathit{fn},\ x))$
$\quad = \quad \mathbf{if}\ \text{cons}\,(\mathit{fn},\ x) \in \text{domain}\,(s)\ \ \mathbf{then}\ \text{value}\,(\text{cons}\,(\mathit{fn},\ x),\ s)$
$\quad\quad \mathbf{else}\ \text{cons}\,(\mathit{fn},\ \text{subst}\,(\mathbf{f},\ s,\ x))\ \mathbf{endif})$

THEOREM: all-vars-subst-lemma
$(\mathit{flg} \wedge (x \in \text{domain}\,(s)))$
$\rightarrow \quad \text{subsetp}\,(\text{all-vars}\,(\mathit{flg},\ \text{value}\,(x,\ s)),\ \text{all-vars}\,(\mathbf{f},\ \text{range}\,(s)))$

THEOREM: all-vars-subst
$\text{termp}\,(\mathit{flg},\ x)$
$\rightarrow \quad \text{subsetp}\,(\text{all-vars}\,(\mathit{flg},\ \text{subst}\,(\mathit{flg},\ s,\ x)),$
$\quad\quad\quad\quad \text{append}\,(\text{all-vars}\,(\mathit{flg},\ x),\ \text{all-vars}\,(\mathbf{f},\ \text{range}\,(s))))$

THEOREM: subst-occur
$(\mathit{flg} \wedge (x \in \text{domain}\,(s))) \rightarrow (\text{subst}\,(\mathit{flg},\ s,\ x) = \text{value}\,(x,\ s))$

THEOREM: boundp-in-var-substp-implies-variablep
$(\text{variable-listp}\,(\text{domain}\,(s)) \wedge (\neg\ \text{variablep}\,(a))) \rightarrow (a \notin \text{domain}\,(s))$

THEOREM: variablep-value-invert
$(\text{variable-listp}\,(\text{domain}\,(s)) \wedge (x \in \text{range}\,(s)))$
$\rightarrow \quad \text{variablep}\,(\text{value}\,(x,\ \text{invert}\,(s)))$

THEOREM: subst-invert
$(\text{termp}\,(\mathit{flg},\ x) \wedge \text{disjoint}\,(\text{domain}\,(s),\ \text{all-vars}\,(\mathit{flg},\ x)) \wedge \text{var-substp}\,(s))$
$\rightarrow \quad (\text{subst}\,(\mathit{flg},\ s,\ \text{subst}\,(\mathit{flg},\ \text{invert}\,(s),\ x)) = x)$

THEOREM: boundp-apply-to-subst
$(x \in \text{domain}\,(\text{apply-to-subst}\,(\mathit{s1},\ \mathit{s2}))) = (x \in \text{domain}\,(\mathit{s2}))$

THEOREM: mapping-apply-to-subst
$\text{mapping}\,(s) \rightarrow \text{mapping}\,(\text{apply-to-subst}\,(\mathit{s1},\ s))$

THEOREM: alistp-apply-to-subst
$\text{alistp}\,(\text{apply-to-subst}\,(\mathit{s1},\ \mathit{s2}))$

THEOREM: subst-flg-not-list
$flg \rightarrow (((\text{subst}\,(flg,\ s,\ x) = \text{subst}\,(\mathbf{t},\ s,\ x)) = \mathbf{t})$
$\quad \wedge \quad ((\text{subst}\,(\mathbf{t},\ s,\ x) = \text{subst}\,(flg,\ s,\ x)) = \mathbf{t}))$

THEOREM: subst-co-restrict
$(\text{disjoint}\,(x,\ \text{intersection}\,(\text{domain}\,(s),\ \text{all-vars}\,(flg,\ term)))$
$\wedge \quad \text{variable-listp}\,(\text{domain}\,(s))$
$\wedge \quad \text{termp}\,(flg,\ term))$
$\rightarrow \quad (\text{subst}\,(flg,\ \text{co-restrict}\,(s,\ x),\ term) = \text{subst}\,(flg,\ s,\ term))$

THEOREM: subst-restrict
$(\text{subsetp}\,(\text{intersection}\,(\text{domain}\,(s),\ \text{all-vars}\,(flg,\ term)),\ x)$
$\wedge \quad \text{variable-listp}\,(\text{domain}\,(s))$
$\wedge \quad \text{termp}\,(flg,\ term))$
$\rightarrow \quad (\text{subst}\,(flg,\ \text{restrict}\,(s,\ x),\ term) = \text{subst}\,(flg,\ s,\ term))$

THEOREM: termp-value
$(flg \wedge (x \in \text{domain}\,(s)) \wedge \text{termp}\,(\mathbf{f},\ \text{range}\,(s))) \rightarrow \text{termp}\,(flg,\ \text{value}\,(x,\ s))$

THEOREM: termp-subst
$(\text{termp}\,(flg,\ x) \wedge \text{termp}\,(\mathbf{f},\ \text{range}\,(s))) \rightarrow \text{termp}\,(flg,\ \text{subst}\,(flg,\ s,\ x))$

THEOREM: termp-domain
$\text{variable-listp}\,(\text{domain}\,(s)) \rightarrow \text{termp}\,(\mathbf{f},\ \text{domain}\,(s))$

THEOREM: domain-apply-to-subst
$\text{domain}\,(\text{apply-to-subst}\,(s1,\ s2)) = \text{domain}\,(s2)$

THEOREM: var-substp-apply-to-subst
$(\text{termp}\,(\mathbf{f},\ \text{range}\,(s)) \wedge \text{termp}\,(\mathbf{f},\ \text{range}\,(sg)))$
$\rightarrow \quad \text{termp}\,(\mathbf{f},\ \text{range}\,(\text{apply-to-subst}\,(sg,\ s)))$

THEOREM: value-apply-to-subst
$(g \in \text{domain}\,(s))$
$\rightarrow \quad (\text{value}\,(g,\ \text{apply-to-subst}\,(sg,\ s)) = \text{subst}\,(\mathbf{t},\ sg,\ \text{value}\,(g,\ s)))$

THEOREM: non-variablep-not-member-of-variable-listp
$(\text{variable-listp}\,(d) \wedge (\neg\ \text{variablep}\,(term))) \rightarrow (term \notin d)$

THEOREM: compose-property
$(\text{variable-listp}\,(\text{domain}\,(s2)) \wedge \text{termp}\,(flg,\ x))$
$\rightarrow \quad (\text{subst}\,(flg,\ \text{compose}\,(s1,\ s2),\ x) = \text{subst}\,(flg,\ s2,\ \text{subst}\,(flg,\ s1,\ x)))$

THEOREM: compose-property-reversed
$(\text{variable-listp}\,(\text{domain}\,(s2)) \wedge \text{termp}\,(flg,\ x))$
$\rightarrow \quad (\text{subst}\,(flg,\ s2,\ \text{subst}\,(flg,\ s1,\ x)) = \text{subst}\,(flg,\ \text{compose}\,(s1,\ s2),\ x))$

EVENT: Disable compose-property.

THEOREM: subst-not-occur
(termp (*flg*, *x*)
 ∧ variable-listp (domain (*s*))
 ∧ disjoint (domain (*s*), all-vars (*flg*, *x*)))
→ (subst (*flg*, *s*, *x*) = *x*)

THEOREM: disjoint-range-implies-disjoint-value
((*x* ∈ domain (*s*)) ∧ *flg* ∧ disjoint (*z*, all-vars (**f**, range (*s*))))
→ disjoint (*z*, all-vars (*flg*, value (*x*, *s*)))

THEOREM: disjoint-all-vars-subst
(termp (*flg*, *x*)
 ∧ disjoint (*z*, all-vars (*flg*, *x*))
 ∧ disjoint (*z*, all-vars (**f**, range (*s*))))
→ disjoint (*z*, all-vars (*flg*, subst (*flg*, *s*, *x*)))

THEOREM: all-vars-variable-listp
variable-listp (*x*) → (all-vars (**f**, *x*) = *x*)

THEOREM: variable-listp-append
variable-listp (append (*x*, *y*))
= (variable-listp (fix-properp (*x*)) ∧ variable-listp (*y*))

THEOREM: termp-list-append
termp (**f**, append (*x*, *y*)) = (termp (**f**, fix-properp (*x*)) ∧ termp (**f**, *y*))

THEOREM: apply-to-subst-append
apply-to-subst (*sg*, append (*s1*, *s2*))
= append (apply-to-subst (*sg*, *s1*), apply-to-subst (*sg*, *s2*))

THEOREM: subst-apply-to-subst
(*flg* ∧ (*g* ∈ domain (*s*)))
→ (subst (*flg*, apply-to-subst (*sg*, *s*), *g*) = subst (*flg*, *sg*, value (*g*, *s*)))

THEOREM: subst-append-not-occur-1
(termp (*flg*, *x*)
 ∧ variable-listp (domain (*s1*))
 ∧ disjoint (all-vars (**f**, domain (*s1*)), all-vars (*flg*, *x*)))
→ (subst (*flg*, append (*s1*, *s2*), *x*) = subst (*flg*, *s2*, *x*))

THEOREM: subst-append-not-occur-2
(termp (*flg*, *x*)
 ∧ variable-listp (domain (*s2*))
 ∧ disjoint (all-vars (**f**, domain (*s2*)), all-vars (*flg*, *x*)))
→ (subst (*flg*, append (*s1*, *s2*), *x*) = subst (*flg*, *s1*, *x*))

THEOREM: apply-to-subst-is-no-op-for-disjoint-domain
(variable-listp (domain (*s1*))
 ∧ alistp (*s2*)
 ∧ termp (**f**, range (*s2*))
 ∧ disjoint (domain (*s1*), all-vars (**f**, range (*s2*))))
→ (apply-to-subst (*s1*, *s2*) = *s2*)

THEOREM: member-subst
(*flg* ∧ (*a* ∈ *x*)) → (subst (*flg*, *s*, *a*) ∈ subst (**f**, *s*, *x*))

THEOREM: subsetp-subst
subsetp (*x*, *y*) → subsetp (subst (**f**, *s*, *x*), subst (**f**, *s*, *y*))

```
;;; (disable instance) -- not included in this version
;;;(disable compose) -- COMPOSE is left enabled for use with COMPOSE-PROPERTY-REVERSED
```

EVENT: Disable apply-to-subst.

EVENT: Disable subst.

EVENT: Disable rembind.

EVENT: Disable bind.

```
;;;;; nullify-subst:  a substitution that has a range containing
;;                    no variables
```

DEFINITION:
nullify-subst (*s*)
= **if** listp (*s*)
  then if listp (car (*s*))
    then cons (cons (caar (*s*), list (FN)), nullify-subst (cdr (*s*)))
    else nullify-subst (cdr (*s*)) **endif**
  else nil endif

THEOREM: properp-nullify-subst
properp (nullify-subst (*s*))

THEOREM: all-vars-f-range-nullify-subst
all-vars (**f**, range (nullify-subst (*s*))) = **nil**

THEOREM: termp-range-nullify-subst
termp (**f**, range (nullify-subst (*s*)))

THEOREM: domain-nullify-subst
domain (nullify-subst (s)) = domain (s)

THEOREM: mapping-nullify-subst
alistp (s) → (mapping (nullify-subst (s)) = mapping (s))

THEOREM: disjoint-all-vars-subst-nullify-subst
termp (*flg*, *term*)
→ disjoint (domain (sg), all-vars (*flg*, subst (*flg*, nullify-subst (sg), *term*)))

THEOREM: disjoint-all-vars-range-apply-subst-nullify-subst
termp (**f**, range (s))
→ disjoint (domain (sg),
all-vars (**f**, range (apply-to-subst (nullify-subst (sg), s))))

EVENT: Disable nullify-subst.

EVENT: Let us define the theory *substitution-defns* to consist of the following events: var-substp, compose, apply-to-subst, subst, nullify-subst.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; generalize.events file
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; Requires sets, alists, and terms, which currently contain a number
;; of rules that aren't really needed here, even indirectly.

;; This is a proof soundness of a slight abstraction of the GENERALIZE
;; command of PC-NQTHM.

#| Here's what I want to prove.

(implies (and (generalize-okp sg state)
              (valid-state (generalize sg state)))
         (valid-state state))

I also prove the much simpler fact, GENERALIZE-STATEP:

(implies (generalize-okp sg state)
         (statep (generalize sg state)))

|#

;; << 1 >>
```

CONSERVATIVE AXIOM: theorem-intro
$((\text{theorem}\,(x) \wedge \mathit{flg}) \rightarrow \text{termp}\,(\mathit{flg},\ x))$
$\wedge \quad ((\text{theorem}\,(x) \wedge \mathit{flg} \wedge \text{var-substp}\,(s)) \rightarrow \text{theorem}\,(\text{subst}\,(\mathit{flg},\ s,\ x)))$

Simultaneously, we introduce the new function symbol *theorem*.

```
;; << 2 >>
```

DEFINITION:
theorem-list (x)
$=$ **if** listp (x) **then** theorem (car (x)) $\wedge$ theorem-list (cdr (x))
else x = **nil endif**

```
;; << 3 >>
```

THEOREM: theorem-list-properties
$(\text{theorem-list}\,(x) \rightarrow \text{termp}\,(\mathbf{f},\ x))$
$\wedge \quad ((\text{theorem-list}\,(x) \wedge \text{var-substp}\,(s)) \rightarrow \text{theorem-list}\,(\text{subst}\,(\mathbf{f},\ s,\ x)))$

```
;; << 4 >>
```

DEFINITION:
statep (state)
$=$ $(\text{listp}\,(\mathit{state}) \wedge \text{termp}\,(\mathbf{f},\ \text{car}\,(\mathit{state})) \wedge \text{variable-listp}\,(\text{cdr}\,(\mathit{state})))$

```
;; << 5 >>
#|
(DEFN-SK VALID-STATE
         (STATE)
         (AND (STATEP STATE)
              (EXISTS WITNESSING-INSTANTIATION
                      (AND (VAR-SUBSTP WITNESSING-INSTANTIATION)
                           (SUBSETP (DOMAIN WITNESSING-INSTANTIATION)
                                    (CDR STATE))
                           (THEOREM-LIST (SUBST F WITNESSING-INSTANTIATION
                                                (CAR STATE)))))))


Adding the Skolem axiom:
      (AND
       (IMPLIES (AND (STATEP STATE)
                     (VAR-SUBSTP WITNESSING-INSTANTIATION)
                     (SUBSETP (DOMAIN WITNESSING-INSTANTIATION)
                              (CDR STATE))
                     (THEOREM-LIST (SUBST F WITNESSING-INSTANTIATION
```

```
                                        (CAR STATE))))
               (VALID-STATE STATE))
       (IMPLIES (NOT (AND (STATEP STATE)
                          (VAR-SUBSTP (WITNESSING-INSTANTIATION STATE))
                          (SUBSETP (DOMAIN (WITNESSING-INSTANTIATION STATE))
                                   (CDR STATE))
                          (THEOREM-LIST (SUBST F
                                               (WITNESSING-INSTANTIATION STATE)
                                               (CAR STATE)))))
               (NOT (VALID-STATE STATE)))).


     As this is a DEFN-SK we can conclude that:
      (OR (TRUEP (VALID-STATE STATE))
          (FALSEP (VALID-STATE STATE)))
is a theorem.


|#
```

EVENT: Introduce the function symbol *witnessing-instantiation* of one argument.

EVENT: Introduce the function symbol *valid-state* of one argument.

AXIOM: valid-state-intro

$$
\begin{array}{l}
((\text{statep}\,(\textit{state}) \\
\quad \wedge \quad \text{var-substp}\,(\textit{witnessing-instantiation}) \\
\quad \wedge \quad \text{subsetp}\,(\text{domain}\,(\textit{witnessing-instantiation}),\ \text{cdr}\,(\textit{state})) \\
\quad \wedge \quad \text{theorem-list}\,(\text{subst}\,(\mathbf{f},\ \textit{witnessing-instantiation},\ \text{car}\,(\textit{state})))) \\
\rightarrow \quad \text{valid-state}\,(\textit{state})) \\
\wedge \quad ((\neg\ (\text{statep}\,(\textit{state}) \\
\qquad \wedge \quad \text{var-substp}\,(\text{witnessing-instantiation}\,(\textit{state})) \\
\qquad \wedge \quad \text{subsetp}\,(\text{domain}\,(\text{witnessing-instantiation}\,(\textit{state})), \\
\qquad\qquad\qquad \text{cdr}\,(\textit{state})) \\
\qquad \wedge \quad \text{theorem-list}\,(\text{subst}\,(\mathbf{f}, \\
\qquad\qquad\qquad\qquad\qquad \text{witnessing-instantiation}\,(\textit{state}), \\
\qquad\qquad\qquad\qquad\qquad \text{car}\,(\textit{state}))))) \\
\quad \rightarrow \quad (\neg\ \text{valid-state}\,(\textit{state})))
\end{array}
$$

AXIOM: valid-state-type

$$\text{truep}\,(\text{valid-state}\,(\textit{state})) \vee \text{falsep}\,(\text{valid-state}\,(\textit{state}))$$

```
;; << 6 >>
```

DEFINITION:
new-gen-vars (*goals*, *free*, *vars*)
= **if** listp (*goals*)
then let *current-free-vars* **be** intersection (*free*,
all-vars (**t**, car (*goals*)))
in
if disjoint (*current-free-vars*, *vars*)
then new-gen-vars (cdr (*goals*), *free*, *vars*)
else append (*current-free-vars*,
new-gen-vars (cdr (*goals*), *free*, *vars*)) **endif endlet**
else nil endif

```
;; << 7 >>
```

DEFINITION: cardinality (*x*) = length (make-set (*x*))

```
;; Next goal:  get the definition of GEN-CLOSURE accepted.  In fact,
;; the lemma GEN-CLOSURE-ACCEPT below suffices, taking NEW to be
;; (NEW-GEN-VARS GOALS FREE FREE-VARS-SO-FAR), as long as we prove the
;; following lemma, NEW-GEN-VARS-SUBSET.
```

```
;; << 8 >>
```

THEOREM: new-gen-vars-subset
subsetp (new-gen-vars (*goals*, *free*, *vars*), *free*)

```
;; It is interesting to note that the exact form of the following
;; lemma changed while polishing the proof, since rewrite rules
;; applied to the old version so as to make it irrelevant.
```

```
;; << 9 >>
```

THEOREM: gen-closure-accept
((¬ subsetp (*new*, *free-vars-so-far*)) ∧ subsetp (*new*, *free*))
→ (((length (make-set (*free*))
− length (intersection (make-set (*free*), *free-vars-so-far*)))
− length (intersection (set-diff (make-set (*free*), *free-vars-so-far*),
new)))
< (length (make-set (*free*))
− length (intersection (make-set (*free*), *free-vars-so-far*))))

```
;; Here I have a choice:  I could intersect the accumulator with free
;; at the end, or I could assume that it's intersected with free
```

```
;; before it's input.  I'll choose the former approach, so that I'll
;; have a simpler rewrite rule and so that I can call gen-closure more
;; simply.  I may wish to commute the arguments to intersection in the
;; exit below, but probably that won't matter because I'll only be
;; talking about membership.

;; << 10 >>
```

DEFINITION:
gen-closure (*goals*, *free*, *free-vars-so-far*)
= **let** *new-free-vars* **be** new-gen-vars (*goals*, *free*, *free-vars-so-far*)
in
if subsetp (*new-free-vars*, *free-vars-so-far*)
then intersection (*free-vars-so-far*, *free*)
else gen-closure (*goals*,
free,
append (*new-free-vars*, *free-vars-so-far*)) **endif endlet**

```
;; << 11 >>
```

DEFINITION:
generalize-okp (*sg*, *state*)
= (var-substp (*sg*)
∧ statep (*state*)
∧ disjoint (domain (*sg*), all-vars (**f**, car (*state*)))
∧ listp (car (*state*))
∧ disjoint (domain (*sg*), cdr (*state*)))

```
;; << 12 >>
```

DEFINITION:
generalize (*sg*, *state*)
= **let** *g* **be** caar (*state*),
p **be** cdar (*state*),
free **be** cdr (*state*),
sg-vars **be** all-vars (**f**, range (*sg*))
in
let *new-g* **be** subst (**t**, invert (*sg*), *g*)
in
let *new-free* **be** set-diff (*free*,
intersection (gen-closure (cons (*new-g*,
p),
free,
all-vars (**t**,
new-g)),

all-vars (**f**,
range (*sg*))))
in
cons (cons (*new-g*, *p*), *new-free*) **endlet endlet endlet**

```
;; Here is a fact, not needed elsewhere, that is worth noticing, in
;; case we wish to extend the current theorem to a sequence of commands.

;; << 13 >>
```

THEOREM: generalize-statep
generalize-okp (*sg*, *state*) → statep (generalize (*sg*, *state*))

```
;; << 14 >>
```

DEFINITION:
gen-inst (*sg*, *state*)
= **let** *s* **be** witnessing-instantiation (generalize (*sg*, *state*)),
domain-1 **be** gen-closure (cons (subst (**t**, invert (*sg*), caar (*state*)),
cdar (*state*)),
cdr (*state*),
all-vars (**t**,
subst (**t**,
invert (*sg*),
caar (*state*))))
in
let *s1* **be** restrict (*s*, *domain-1*),
s2 **be** apply-to-subst (nullify-subst (*sg*),
co-restrict (*s*, *domain-1*))
in
apply-to-subst (apply-to-subst (*s2*, *sg*), append (*s1*, *s2*)) **endlet endlet**

```
;; Let's see that it suffices to prove the result of opening up the
;; conclusion of the main theorem with a particular witness.

#|

(add-axiom main-theorem-1 (rewrite)
  (let ((wit (gen-inst sg state)))
    (implies (and (generalize-okp sg state)
                  (valid-state (generalize sg state)))
             (and (statep state)
                  (var-substp wit)
                  (subsetp (domain wit) (cdr state))
                  (theorem-list (subst f wit (car state)))))))
```

```
(prove-lemma generalize-is-correct (rewrite)
  (implies (and (generalize-okp sg state)
                (valid-state (generalize sg state)))
           (valid-state state))
  ((disable-theory t)
   (enable-theory ground-zero)
   (enable main-theorem-1)
   (use (valid-state-intro
         (witnessing-instantiation (gen-inst sg state))))))

|#

;; So, it suffices to prove main-theorem-1.  The first three conjuncts
;; of the conclusion are quite trivial.

;; << 15 >>
```

THEOREM: main-theorem-1-case-1
generalize-okp (*sg*, *state*) → statep (*state*)

```
;; We put one direction of the definition of valid-state here, for
;; efficiency in proofs.

;; << 16 >>
```

THEOREM: valid-state-opener
valid-state (*state*)
= (statep (*state*)
∧ **let** *witnessing-instantiation* **be** witnessing-instantiation (*state*)
in
var-substp (*witnessing-instantiation*)
∧ subsetp (domain (*witnessing-instantiation*),
cdr (*state*))
∧ theorem-list (subst (**f**,
witnessing-instantiation,
car (*state*))) **endlet**)

```
;; << 17 >>
```

THEOREM: main-theorem-1-case-2
let *wit* **be** gen-inst (*sg*, *state*)
in
(generalize-okp (*sg*, *state*) ∧ valid-state (generalize (*sg*, *state*)))
→ var-substp (*wit*) **endlet**

```
;; << 18 >>
```

THEOREM: subsetp-cdr-generalize
subsetp (cdr (generalize (*sg*, *state*)), cdr (*state*))

```
;; At this point I had to prove SUBSETP-SET-DIFF-SUFFICIENCY because
;; of some lemma that was created during the polishing process
;; (perhaps DOMAIN-RESTRICT).

;; << 19 >>
```

THEOREM: main-theorem-1-case-3
let *wit* **be** gen-inst (*sg*, *state*)
in
valid-state (generalize (*sg*, *state*))
→ subsetp (domain (*wit*), cdr (*state*)) **endlet**

```
;; So now we only have to prove MAIN-THEOREM-1-CASE-4 (written here
;; without use of LET):

#|

(add-axiom main-theorem-1-case-4 (rewrite)
  (implies (and (generalize-okp sg state)
                (valid-state (generalize sg state)))
           (theorem-list (subst f (gen-inst sg state) (car state)))))

(prove-lemma main-theorem-1 (rewrite)
  (let ((wit (gen-inst sg state)))
    (implies (and (generalize-okp sg state)
                  (valid-state (generalize sg state)))
             (and (statep state)
                  (var-substp wit)
                  (subsetp (domain wit) (cdr state))
                  (theorem-list (subst f wit (car state)))))))
  ((disable-theory t)
   (enable-theory ground-zero)
   (enable main-theorem-1-case-1 main-theorem-1-case-2
           main-theorem-1-case-3 main-theorem-1-case-4)))

|#

;; << 20 >>
```

DEFINITION:

gen-setting-substitutions (*s1*, *s2*, *sg*)
= (var-substp (*s1*)
∧ var-substp (*s2*)
∧ var-substp (*sg*)
∧ disjoint (domain (*s1*), domain (*s2*))
∧ disjoint (domain (*s1*), domain (*sg*))
∧ disjoint (domain (*s2*), domain (*sg*))
∧ disjoint (all-vars (**f**, range (*sg*)), domain (*s1*))
∧ disjoint (all-vars (**f**, range (*s2*)), domain (*sg*)))

```
;; << 21 >>
```

DEFINITION:
main-hyps (*s1*, *s2*, *sg*, *g*, *p*)
= (termp (**t**, *g*)
∧ disjoint (all-vars (**t**, *g*), domain (*sg*))
∧ termp (**f**, *p*)
∧ disjoint (all-vars (**f**, *p*), domain (*sg*))
∧ gen-setting-substitutions (*s1*, *s2*, *sg*)
∧ theorem-list (subst (**f**,
append (*s1*, *s2*),
cons (subst (**t**, invert (*sg*), *g*), *p*))))

```
;; The goal above, MAIN-THEOREM-1-CASE-4, should follow from the
;; following two lemmas.

#|

(add-axiom main-hyps-suffice (rewrite)
  (implies (and (listp goals)
                (main-hyps s1 s2 sg (car goals) (cdr goals)))
           (theorem-list (subst f
                                (apply-to-subst (apply-to-subst s2 sg)
                                                (append s1 s2))
                                goals))))

(add-axiom main-hyps-relieved (rewrite)
  (let ((g (caar state))
        (p (cdar state))
        (free (cdr state))
        (s (witnessing-instantiation (generalize sg state))))
    (let ((new-g (subst t (invert sg) g)))
      (let ((domain-1
             (gen-closure (cons new-g p) free (all-vars t new-g))))
        (let ((s1 (restrict s domain-1))
```

```
             (s2 (apply-to-subst (nullify-subst sg)
                                 (co-restrict s domain-1))))
          (implies (and (generalize-okp sg state)
                        (valid-state (generalize sg state)))
                   (main-hyps s1 s2 sg g p)))))))

(prove-lemma main-theorem-1-case-4 (rewrite)
  (implies (and (generalize-okp sg state)
                (valid-state (generalize sg state)))
           (theorem-list (subst f (gen-inst sg state) (car state))))
  ((disable-theory t)
   (enable-theory ground-zero)
   (enable gen-inst main-hyps-suffice generalize-okp main-hyps-relieved)))

|#

;; So, now let us start with MAIN-HYPS-SUFFICE.  It should follow from
;; two subgoals, as shown:

#|

(add-axiom main-hyps-suffice-first (rewrite)
  (implies (main-hyps s1 s2 sg g p)
           (theorem (subst t
                           (apply-to-subst (apply-to-subst s2 sg)
                                           (append s1 s2))
                           g))))

(add-axiom main-hyps-suffice-rest (rewrite)
  (implies (main-hyps s1 s2 sg g p)
           (theorem-list (subst f
                                (apply-to-subst (apply-to-subst s2 sg)
                                                (append s1 s2))
                                p))))

(prove-lemma main-hyps-suffice (rewrite)
  (implies (and (listp goals)
                (main-hyps s1 s2 sg (car goals) (cdr goals)))
           (theorem-list (subst f
                                (apply-to-subst (apply-to-subst s2 sg)
                                                (append s1 s2))
                                goals)))
  ((disable-theory t)
   (enable-theory ground-zero)
```

```
   (enable theorem-list subst main-hyps-suffice-first main-hyps-suffice-rest)))

|#

;; Consider the first of these.  Although COMPOSE-PROPERTY-REVERSED is
;; used in the proof (because it's enabled), it's actually not
;; necessary.  A proof took slightly over 10 minutes with the rule
;; enabled, and roughly 9 minutes without.

;; << 22 >>
```

THEOREM: main-hyps-suffice-first-lemma-general
(termp (*flg*, *g*)
 ∧ disjoint (all-vars (*flg*, *g*), domain (*sg*))
 ∧ gen-setting-substitutions (*s1*, *s2*, *sg*)
 ∧ (*sg-1* = invert (*sg*)))
→ (subst (*flg*, apply-to-subst (apply-to-subst (*s2*, *sg*), append (*s1*, *s2*)), *g*)
 = subst (*flg*,
 apply-to-subst (*s2*, *sg*),
 subst (*flg*, append (*s1*, *s2*), subst (*flg*, *sg-1*, *g*))))

```
;; << 23 >>
```

THEOREM: main-hyps-suffice-first-lemma
(termp (**t**, *g*)
 ∧ disjoint (all-vars (**t**, *g*), domain (*sg*))
 ∧ gen-setting-substitutions (*s1*, *s2*, *sg*))
→ (subst (**t**, apply-to-subst (apply-to-subst (*s2*, *sg*), append (*s1*, *s2*)), *g*)
 = subst (**t**,
 apply-to-subst (*s2*, *sg*),
 subst (**t**, append (*s1*, *s2*), subst (**t**, invert (*sg*), *g*))))

```
;; << 24 >>
```

THEOREM: main-hyps-suffice-first
main-hyps (*s1*, *s2*, *sg*, *g*, *p*)
→ theorem (subst (**t**, apply-to-subst (apply-to-subst (*s2*, *sg*), append (*s1*, *s2*)), *g*))

```
;; The following is useful with s = (append s1 s2).

;; << 25 >>
```

THEOREM: main-hyps-suffice-rest-lemma
(termp (*flg*, *p*)
 ∧ variable-listp (domain (*sg*))

$\wedge$ disjoint (all-vars (*flg*, *p*), domain (*sg*)))
$\rightarrow$ (subst (*flg*, apply-to-subst (apply-to-subst (*s2*, *sg*), *s*), *p*)
= subst (*flg*, apply-to-subst (*s2*, *sg*), subst (*flg*, *s*, *p*)))

```
;; << 26 >>
```

THEOREM: main-hyps-suffice-rest
main-hyps (*s1*, *s2*, *sg*, *g*, *p*)
$\rightarrow$ theorem-list (subst (**f**,
apply-to-subst (apply-to-subst (*s2*, *sg*), append (*s1*, *s2*)),
p))

```
;; << 27 >>
```

THEOREM: main-hyps-suffice
(listp (*goals*) $\wedge$ main-hyps (*s1*, *s2*, *sg*, car (*goals*), cdr (*goals*)))
$\rightarrow$ theorem-list (subst (**f**,
apply-to-subst (apply-to-subst (*s2*, *sg*), append (*s1*, *s2*)),
goals))

```
;; I'll disable the two lemmas used above so that I avoid the possibility
;; of looping with COMPOSE-PROPERTY-REVERSED.

;; << 28 >>
```

EVENT: Disable main-hyps-suffice-first-lemma.

```
;; << 29 >>
```

EVENT: Disable main-hyps-suffice-rest-lemma.

```
;; All that remains now is to prove MAIN-HYPS-RELIEVED.  If we open up
;; MAIN-HYPS we see what the necessary subgoals are.  Recall the
;; definition of MAIN-HYPS:

#|
(defn main-hyps (s1 s2 sg g p)
  (and (termp t g)
       (disjoint (all-vars t g) (domain sg))
       (termp f p)
       (disjoint (all-vars f p) (domain sg))
       (gen-setting-substitutions s1 s2 sg)
       (theorem-list (subst f (append s1 s2)
                            (cons (subst t (invert sg) g) p)))))
```

```
|#

;; << 30 >>
```

THEOREM: main-hyps-relieved-1
let g **be** caar ($state$)
in
generalize-okp (sg, $state$) → termp (**t**, g) **endlet**

```
;; << 31 >>
```

THEOREM: main-hyps-relieved-2
let g **be** caar ($state$)
in
generalize-okp (sg, $state$) → disjoint (all-vars (**t**, g), domain (sg)) **endlet**

```
;; << 32 >>
```

THEOREM: main-hyps-relieved-3
let p **be** cdar ($state$)
in
generalize-okp (sg, $state$) → termp (**f**, p) **endlet**

```
;; << 33 >>
```

THEOREM: main-hyps-relieved-4
let p **be** cdar ($state$)
in
generalize-okp (sg, $state$) → disjoint (all-vars (**f**, p), domain (sg)) **endlet**

```
#|

(add-axiom main-hyps-relieved-5 (rewrite)
  (let ((g (caar state))
        (p (cdar state))
        (free (cdr state))
        (s (witnessing-instantiation (generalize sg state))))
    (let ((new-g (subst t (invert sg) g)))
      (let ((domain-1
             (gen-closure (cons new-g p) free (all-vars t new-g))))
        (let ((s1 (restrict s domain-1))
              (s2 (apply-to-subst (nullify-subst sg)
                                  (co-restrict s domain-1))))
          (implies (and (generalize-okp sg state)
                        (valid-state (generalize sg state)))
```

```
                  (gen-setting-substitutions s1 s2 sg)))))))

(add-axiom main-hyps-relieved-6 (rewrite)
  (let ((g (caar state))
        (p (cdar state))
        (free (cdr state))
        (s (witnessing-instantiation (generalize sg state))))
    (let ((new-g (subst t (invert sg) g)))
      (let ((domain-1
             (gen-closure (cons new-g p) free (all-vars t new-g))))
        (let ((s1 (restrict s domain-1))
              (s2 (apply-to-subst (nullify-subst sg)
                                  (co-restrict s domain-1))))
          (implies (and (generalize-okp sg state)
                        (valid-state (generalize sg state)))
                   (theorem-list (subst f (append s1 s2)
                                        (cons (subst t (invert sg) g) p)))))))))

(prove-lemma main-hyps-relieved (rewrite)
  (let ((g (caar state))
        (p (cdar state))
        (free (cdr state))
        (s (witnessing-instantiation (generalize sg state))))
    (let ((new-g (subst t (invert sg) g)))
      (let ((domain-1
             (gen-closure (cons new-g p) free (all-vars t new-g))))
        (let ((s1 (restrict s domain-1))
              (s2 (apply-to-subst (nullify-subst sg)
                                  (co-restrict s domain-1))))
          (implies (and (generalize-okp sg state)
                        (valid-state (generalize sg state)))
                   (main-hyps s1 s2 sg g p))))))
  ((disable-theory t)
   (enable-theory ground-zero)
   (enable main-hyps main-hyps-relieved-1 main-hyps-relieved-2 main-hyps-relieved-3
           main-hyps-relieved-4 main-hyps-relieved-5 main-hyps-relieved-6)))

|#

;; So, we have left the goals MAIN-HYPS-RELIEVED-5 and
;; MAIN-HYPS-RELIEVED-6.  Let us start with the first.  Opening up
;; GEN-SETTING-SUBSTITUTIONS gives us a number of subgoals.

;; The first two cases below do not require knowledge about DOMAIN-1
```

```
;; (or G, P, FREE, or NEW-G), but simply following from the validity
;; of the state (GENERALIZE SG STATE).  Disabling GENERALIZE is very
;; useful (probably not necessary, though I didn't let the prover run
;; long enough to find out).

;; << 34 >>
```

THEOREM: main-hyps-relieved-5-lemma-1
let *s* **be** witnessing-instantiation (generalize (*sg*, *state*))
in
let *s1* **be** restrict (*s*, *domain-1*),
s2 **be** apply-to-subst (nullify-subst (*sg*), co-restrict (*s*, *domain-1*))
in
valid-state (generalize (*sg*, *state*))
→ (var-substp (*s1*) ∧ var-substp (*s2*)) **endlet endlet**

```
;; The next case is trivial.

;; << 35 >>
```

THEOREM: main-hyps-relieved-5-lemma-2
generalize-okp (*sg*, *state*) → var-substp (*sg*)

```
;; The next case is also quite trivial; notice how abstracted it is.

;; << 36 >>
```

THEOREM: main-hyps-relieved-5-lemma-3
let *s1* **be** restrict (*s*, *domain-1*),
s2 **be** apply-to-subst (nullify-subst (*sg*), co-restrict (*s*, *domain-1*))
in
disjoint (domain (*s1*), domain (*s2*)) **endlet**

```
;; For the next two cases we first observe that (DOMAIN S) is disjoint
;; from (DOMAIN SG), and then we use SUBSETP-DISJOINT-1 where X is the
;; domain of S1 or S2, Y is the domain of S, and Z is the domain of
;; SG:
;;             (IMPLIES (AND (SUBSETP X Y) (DISJOINT Y Z))
;;                      (DISJOINT X Z))

;; << 37 >>
```

THEOREM: witnessing-instantiation-is-disjoint-from-generalizing-substitution
let *s* **be** witnessing-instantiation (generalize (*sg*, *state*))
in
(generalize-okp (*sg*, *state*) ∧ valid-state (generalize (*sg*, *state*)))
→ disjoint (domain (*s*), domain (*sg*)) **endlet**

```
;; In the next case we abstract away DOMAIN-1 (and hence G, P, FREE,
;; and NEW-G).  Incidentally, a similar phenomenon occurred here to
;; the one reported just above the statement above of MAIN-THEOREM-1-CASE-3:
;; final polishing resulted in the need for another lemma.  That extra
;; lemma is DISJOINT-SET-DIFF-SUFFICIENCY in this case, to be found
;; in "sets.events".

;; << 38 >>
```

THEOREM: main-hyps-relieved-5-lemma-4

let *s* **be** witnessing-instantiation (generalize (*sg*, *state*))
in
let *s1* **be** restrict (*s*, *domain-1*),
    s2 **be** apply-to-subst (nullify-subst (*sg*), co-restrict (*s*, *domain-1*))
in
(generalize-okp (*sg*, *state*)
 ∧ valid-state (generalize (*sg*, *state*)))
→ (disjoint (domain (*s1*), domain (*sg*))
    ∧ disjoint (domain (*s2*), domain (*sg*))) **endlet endlet**

```
;; The lemma MAIN-HYPS-RELIEVED-5-LEMMA-5-WIT is true because the
;; domain of s is contained in the free variables of the generalized
;; state (by choice, i.e. definition, of witnessing-instantiation),
;; which is disjoint from the intersection of the indicated
;; GEN-CLOSURE with the variables in the range of sg.  I'll use a
;; trick that I learned from Ken Kunen (definable Skolem function is
;; all, really) to reduce disjointness considerations to membership
;; considerations.

;; << 39 >>
```

THEOREM: main-hyps-relieved-5-lemma-5-wit

let *g* **be** caar (*state*),
    p **be** cdar (*state*),
    free **be** cdr (*state*),
    s **be** witnessing-instantiation (generalize (*sg*, *state*))
in
let *new-g* **be** subst (**t**, invert (*sg*), *g*)
in
let *domain-1* **be** gen-closure (cons (*new-g*, *p*),
                                free,
                                all-vars (**t**, *new-g*))
in
let *s1* **be** restrict (*s*, *domain-1*)
in

(generalize-okp (*sg*, *state*)
 ∧ valid-state (generalize (*sg*, *state*))
 ∧ (*wit* ∈ all-vars (**f**, range (*sg*)))
 ∧ (*wit* ∈ domain (*s*)))
→ (*wit* ∉ *domain-1*) **endlet endlet endlet endlet**

```
;; << 40 >>
```

THEOREM: main-hyps-relieved-5-lemma-5
let *g* **be** caar (*state*),
 p **be** cdar (*state*),
 free **be** cdr (*state*),
 s **be** witnessing-instantiation (generalize (*sg*, *state*))
in
let *new-g* **be** subst (**t**, invert (*sg*), *g*)
in
let *domain-1* **be** gen-closure (cons (*new-g*, *p*),
 free,
 all-vars (**t**, *new-g*))
in
let *s1* **be** restrict (*s*, *domain-1*)
in
(generalize-okp (*sg*, *state*)
 ∧ valid-state (generalize (*sg*, *state*)))
→ disjoint (all-vars (**f**, range (*sg*)), domain (*s1*)) **endlet endlet endlet endlet**

```
;; << 41 >>
```

THEOREM: main-hyps-relieved-5-lemma-6
let *g* **be** caar (*state*),
 p **be** cdar (*state*),
 free **be** cdr (*state*),
 s **be** witnessing-instantiation (generalize (*sg*, *state*))
in
let *new-g* **be** subst (**t**, invert (*sg*), *g*)
in
let *domain-1* **be** gen-closure (cons (*new-g*, *p*),
 free,
 all-vars (**t**, *new-g*))
in
let *s2* **be** apply-to-subst (nullify-subst (*sg*),
 co-restrict (*s*, *domain-1*))
in
(generalize-okp (*sg*, *state*)
 ∧ valid-state (generalize (*sg*, *state*)))
→ disjoint (all-vars (**f**, range (*s2*)), domain (*sg*)) **endlet endlet endlet endlet**

```
;; << 42 >>
```

THEOREM: main-hyps-relieved-5
let *g* **be** caar (*state*),
    p **be** cdar (*state*),
    free **be** cdr (*state*),
    s **be** witnessing-instantiation (generalize (*sg*, *state*))
in
let *new-g* **be** subst (**t**, invert (*sg*), *g*)
in
let *domain-1* **be** gen-closure (cons (*new-g*, *p*),
                                free,
                                all-vars (**t**, *new-g*))
in
let *s1* **be** restrict (*s*, *domain-1*),
    s2 **be** apply-to-subst (nullify-subst (*sg*),
                                co-restrict (*s*, *domain-1*))
in
(generalize-okp (*sg*, *state*)
 ∧ valid-state (generalize (*sg*, *state*)))
→ gen-setting-substitutions (*s1*, *s2*, *sg*) **endlet endlet endlet endlet**

```
;; Now all that is left is MAIN-HYPS-RELIEVED-6.  Actually, this lemma
;; doesn't have anything to do with inverse substitutions or even with
;; generalization, really.  The idea is simply that one takes a valid
;; state and wishes to split its witnessing substitution into two
;; parts.  The parts are the respective restriction and co-restriction
;; of the original substitution to some set that is ``closed'' in the
;; appropriate sense.  Actually, the co-restriction is allowed to have
;; a substitution applied to it, whose domain is disjoint from the
;; goals ``outside'' that closure.  Below we give the lemmas and the
;; proof of MAIN-HYPS-RELIEVED-6 from those lemmas.  But first let us
;; introduce the necessary notions.

;; << 43 >>
```

DEFINITION:
all-vars-disjoint-or-subsetp (*goals*, *free*, *x*)
= **if** listp (*goals*)
  then (subsetp (intersection (*free*, all-vars (**t**, car (*goals*))), *x*)
        ∨ disjoint (intersection (*free*, all-vars (**t**, car (*goals*))), *x*))
       ∧ all-vars-disjoint-or-subsetp (cdr (*goals*), *free*, *x*)
  else t endif

```
;; Our plan will be to show that (CDR STATE) has the above property
```

```
;; with respect to the free variables of the generalized state and the
;; appropriate gen-closure.  In cases where one applies a substitution
;; of the form (append s1 s2) to such a list of goals, where the
;; domain of s1 is contained in the intersection of those free
;; variables with that closure and the domain of s2 is disjoint from
;; that intersection, we'll show that the result is a theorem-list iff
;; each of the following are theorem-lists:  apply s1 to the goals
;; whose vars intersect its domain, and apply s2 to the rest.
;; Reduction rules about applying restrictions etc. will then finish
;; the job.

;; Notice the similarity of the following definition with new-gen-vars.
;; Think of vars as the closure variables, and free as the free variable
;; set within which this all "takes place".

;; << 44 >>
```

DEFINITION:
goals-intersecting-vars (*goals*, *free*, *vars*)
= **if** listp (*goals*)
then let *current-free-vars* **be** intersection (*free*,
all-vars (**t**, car (*goals*)))
in
if disjoint (*current-free-vars*, *vars*)
then goals-intersecting-vars (cdr (*goals*), *free*, *vars*)
else cons (car (*goals*),
goals-intersecting-vars (cdr (*goals*),
free,
vars)) **endif endlet**
else nil endif

```
;; << 45 >>
```

DEFINITION:
goals-disjoint-from-vars (*goals*, *free*, *vars*)
= **if** listp (*goals*)
then let *current-free-vars* **be** intersection (*free*,
all-vars (**t**, car (*goals*)))
in
if disjoint (*current-free-vars*, *vars*)
then cons (car (*goals*),
goals-disjoint-from-vars (cdr (*goals*), *free*, *vars*))
else goals-disjoint-from-vars (cdr (*goals*), *free*, *vars*) **endif endlet**
else nil endif

```
;; Now we begin the remaining goal, MAIN-HYPS-RELIEVED-6.  The idea is
;; to show that the appropriate goal list is a theorem-list by showing
;; separately that the first and the rest are theorems, since the
;; reasons are slightly different.  The first is a theorem because its
;; free vars are all in domain-1, hence in the domain of s1; so, s2
;; can be dropped from the APPEND.  The rest all have the property
;; that their free vars are contained in or disjoint from domain-1,
;; and for those disjoint from it, they do not contain variables from
;; the domain of sg.  Notice that the new current goal may violate the
;; latter requirement, since it may have no free vars at all but
;; contain vars from the domain of sg, and that's why we have to make
;; a special case out of it.

#|

(add-axiom main-hyps-relieved-6-first (rewrite)
  (let ((g (caar state))
        (p (cdar state))
        (free (cdr state))
        (s (witnessing-instantiation (generalize sg state))))
    (let ((new-g (subst t (invert sg) g)))
      (let ((domain-1
             (gen-closure (cons new-g p) free (all-vars t new-g))))
        (let ((s1 (restrict s domain-1))
              (s2 (apply-to-subst (nullify-subst sg)
                                  (co-restrict s domain-1))))
          (implies (and (generalize-okp sg state)
                        (valid-state (generalize sg state)))
                   (theorem (subst t (append s1 s2)
                                   (subst t (invert sg) g)))))))))

(add-axiom main-hyps-relieved-6-rest (rewrite)
  (let ((g (caar state))
        (p (cdar state))
        (free (cdr state))
        (s (witnessing-instantiation (generalize sg state))))
    (let ((new-g (subst t (invert sg) g)))
      (let ((domain-1
             (gen-closure (cons new-g p) free (all-vars t new-g))))
        (let ((s1 (restrict s domain-1))
              (s2 (apply-to-subst (nullify-subst sg)
                                  (co-restrict s domain-1))))
          (implies (and (generalize-okp sg state)
                        (valid-state (generalize sg state)))
```

```
                  (theorem-list (subst f (append s1 s2) p))))))))

(prove-lemma main-hyps-relieved-6 (rewrite)
  (let ((g (caar state))
        (p (cdar state))
        (free (cdr state))
        (s (witnessing-instantiation (generalize sg state))))
    (let ((new-g (subst t (invert sg) g)))
      (let ((domain-1
             (gen-closure (cons new-g p) free (all-vars t new-g))))
        (let ((s1 (restrict s domain-1))
              (s2 (apply-to-subst (nullify-subst sg)
                                  (co-restrict s domain-1))))
          (implies (and (generalize-okp sg state)
                        (valid-state (generalize sg state)))
                   (theorem-list (subst f (append s1 s2)
                                        (cons (subst t (invert sg) g) p))))))))
  ((disable-theory t)
   (enable-theory ground-zero)
   (enable main-hyps-relieved-6-first main-hyps-relieved-6-rest subst theorem-list)))

|#

;; The first is true because the free vars in new-g are all in the
;; domain of s1, since they are all in domain-1.  By the way, the
;; proof-checker was useful here; I dove to the subst term (after
;; adding abbreviations and promoting hypotheses) and saw that I
;; wanted to rewrite with SUBST-APPEND-NOT-OCCUR-2.  I also notice the
;; need for GEN-CLOSURE-CONTAINS-THIRD-ARG during the attempt to prove
;; a goal.

;; First, we only want to open up GENERALIZE when we are looking at
;; goals, not when we are simply asking about the witnessing
;; substitution.

;; << 46 >>
```

THEOREM: car-generalize
car (generalize (*sg*, *state*))
= cons (subst (**t**, invert (*sg*), caar (*state*)), cdar (*state*))

```
;; << 47 >>
```

EVENT: Disable generalize.

```
;; Inspection of the proof of a subgoal of MAIN-HYPS-RELIEVED-6-FIRST
;; suggests that we need the following lemma.  Actually, before the
;; final polishing it was the case that the following version sufficed.
;; But final polishing led me to prove a "better" version, as well
;; as the lemma DISJOINT-SET-DIFF-GENERAL in "sets.events".

#|
(prove-lemma gen-closure-contains-third-arg (rewrite)
  (implies (subsetp domain free)
           (subsetp (intersection domain free-vars-so-far)
                    (gen-closure goals free free-vars-so-far))))
|#

;; << 48 >>
```

THEOREM: gen-closure-contains-third-arg
subsetp (*x*, intersection (*free*, *free-vars-so-far*))
→ subsetp (*x*, gen-closure (*goals*, *free*, *free-vars-so-far*))

```
;; << 49 >>
```

THEOREM: main-hyps-relieved-6-first
let *g* **be** caar (*state*),
p **be** cdar (*state*),
free **be** cdr (*state*),
s **be** witnessing-instantiation (generalize (*sg*, *state*))
in
let *new-g* **be** subst (**t**, invert (*sg*), *g*)
in
let *domain-1* **be** gen-closure (cons (*new-g*, *p*),
free,
all-vars (**t**, *new-g*))
in
let *s1* **be** restrict (*s*, *domain-1*),
s2 **be** apply-to-subst (nullify-subst (*sg*),
co-restrict (*s*, *domain-1*))
in
(generalize-okp (*sg*, *state*)
∧ valid-state (generalize (*sg*, *state*)))
→ theorem (subst (**t**, append (*s1*, *s2*), *new-g*)) **endlet endlet endlet endlet**

```
;; Now all that remains is MAIN-HYPS-RELIEVED-6-REST.

#|
```

```
;; I originally forgot the (TERMP F P) hypothesis below, but it wasn't
;; very hard to back up and fix this.

(add-axiom main-hyps-relieved-6-rest-generalization (rewrite)
  (let ((s1 (restrict s domain-1))
        (s2 (apply-to-subst (nullify-subst sg)
                            (co-restrict s domain-1))))
    (implies (and (var-substp sg)
                  (var-substp s)
                  (subsetp (domain s) new-free)
                  (termp f p)
                  (theorem-list (subst f s p))
                  (disjoint (domain sg)
                            (all-vars f (goals-disjoint-from-vars
                                         p new-free domain-1)))
                  (all-vars-disjoint-or-subsetp p new-free domain-1))
             (theorem-list (subst f (append s1 s2) p)))))

(add-axiom main-hyps-relieved-6-rest-lemma-1 (rewrite)
  (let ((g (caar state))
        (p (cdar state))
        (free (cdr state))
        (s (witnessing-instantiation (generalize sg state))))
    (let ((new-g (subst t (invert sg) g)))
      (let ((domain-1
             (gen-closure (cons new-g p) free (all-vars t new-g))))
        (let ((s1 (restrict s domain-1))
              (s2 (apply-to-subst (nullify-subst sg)
                                  (co-restrict s domain-1))))
          (implies (and (generalize-okp sg state)
                        (valid-state (generalize sg state)))
                   (disjoint (domain sg)
                             (all-vars f (goals-disjoint-from-vars
                                          p (cdr (generalize sg state)) domain-1)))))))))

;; Minor note:  I used the BREAK-LEMMA feature of NQTHM to realize
;; that I needed the following lemma.

(add-axiom main-hyps-relieved-6-rest-lemma-2 (rewrite)
  (let ((g (caar state))
        (p (cdar state))
        (free (cdr state))
        (s (witnessing-instantiation (generalize sg state))))
    (let ((new-g (subst t (invert sg) g)))
```

```
      (let ((domain-1
             (gen-closure (cons new-g p) free (all-vars t new-g))))
        (let ((s1 (restrict s domain-1))
              (s2 (apply-to-subst (nullify-subst sg)
                                  (co-restrict s domain-1))))
          (implies (and (generalize-okp sg state)
                        (valid-state (generalize sg state)))
                   (all-vars-disjoint-or-subsetp p (cdr (generalize sg state)) domain-1)))))

(prove-lemma main-hyps-relieved-6-rest (rewrite)
  (let ((g (caar state))
        (p (cdar state))
        (free (cdr state))
        (s (witnessing-instantiation (generalize sg state))))
    (let ((new-g (subst t (invert sg) g)))
      (let ((domain-1
             (gen-closure (cons new-g p) free (all-vars t new-g))))
        (let ((s1 (restrict s domain-1))
              (s2 (apply-to-subst (nullify-subst sg)
                                  (co-restrict s domain-1))))
          (implies (and (generalize-okp sg state)
                        (valid-state (generalize sg state)))
                   (theorem-list (subst f (append s1 s2) p)))))))
  ((disable-theory t)
   (enable-theory ground-zero)
   (enable theorem-list subst car-generalize ;; so that we can get at p from (car state)
           ;; relieving hyps of main-hyps-relieved-6-rest-generalization:
           main-hyps-relieved-6-rest-lemma-1 main-hyps-relieved-6-rest-lemma-2
           ;; to relieve the (termp f p) hypothesis in main-hyps-relieved-6-rest-generalizat
           statep termp-list-cons
           generalize-okp valid-state-opener main-hyps-relieved-6-rest-generalization)))

;; At this point I did a sanity check and sure enough, the pushed
;; lemmas all go through at this point:  main-hyps-relieved-6,
;; main-hyps-relieved, main-theorem-1-case-4, main-theorem-1, and
;; generalize-is-correct.

|#

;; It remains to prove MAIN-HYPS-RELIEVED-6-REST-LEMMA-1,
;; MAIN-HYPS-RELIEVED-6-REST-LEMMA-2, and
;; MAIN-HYPS-RELIEVED-6-REST-GENERALIZATION.

;; For the first of these we need the following trivial observation.
```

```
;; << 50 >>
```

THEOREM: goals-disjoint-from-vars-subsetp
subsetp (goals-disjoint-from-vars (*goals*, *free*, *vars*), *goals*)

```
;; Unfortunately the observation above doesn't quite suffice, because
;; of a technical problem with free variables in hypotheses.  The
;; following consequence does, though.

;; << 51 >>
```

THEOREM: disjoint-all-vars-goals-disjoint-from-vars
disjoint (*x*, all-vars (**f**, *goals*))
→ disjoint (*x*, all-vars (**f**, goals-disjoint-from-vars (*goals*, *free*, *vars*)))

```
;; << 52 >>
```

THEOREM: main-hyps-relieved-6-rest-lemma-1
let *g* **be** caar (*state*),
p **be** cdar (*state*),
free **be** cdr (*state*),
s **be** witnessing-instantiation (generalize (*sg*, *state*))
in
let *new-g* **be** subst (**t**, invert (*sg*), *g*)
in
let *domain-1* **be** gen-closure (cons (*new-g*, *p*),
free,
all-vars (**t**, *new-g*))
in
let *s1* **be** restrict (*s*, *domain-1*),
s2 **be** apply-to-subst (nullify-subst (*sg*),
co-restrict (*s*, *domain-1*))
in
(generalize-okp (*sg*, *state*)
∧ valid-state (generalize (*sg*, *state*)))
→ disjoint (domain (*sg*),
all-vars (**f**,
goals-disjoint-from-vars (*p*,
cdr (generalize (*sg*,
state)),
domain-1))) **endlet endlet endlet endlet**

```
;; The next goal, MAIN-HYPS-RELIEVED-6-REST-LEMMA-2, needs the lemma
;; ALL-VARS-DISJOINT-OR-SUBSETP-GEN-CLOSURE below.  That lemma's
```

```
;; mechanical proof depends on the trivial observation
;; DISJOINT-INTERSECTION3-MIDDLE in file sets.events.

;; << 53 >>
```

THEOREM: all-vars-disjoint-or-subsetp-gen-closure
subsetp (*domain*, *free*)
→ all-vars-disjoint-or-subsetp (*goals*,
domain,
gen-closure (cons (*g*, *goals*), *free*, *vars*))

```
;; << 54 >>
```

THEOREM: main-hyps-relieved-6-rest-lemma-2
let *g* **be** caar (*state*),
p **be** cdar (*state*),
free **be** cdr (*state*),
s **be** witnessing-instantiation (generalize (*sg*, *state*))
in
let *new-g* **be** subst (**t**, invert (*sg*), *g*)
in
let *domain-1* **be** gen-closure (cons (*new-g*, *p*),
free,
all-vars (**t**, *new-g*))
in
let *s1* **be** restrict (*s*, *domain-1*),
s2 **be** apply-to-subst (nullify-subst (*sg*),
co-restrict (*s*, *domain-1*))
in
(generalize-okp (*sg*, *state*)
∧ valid-state (generalize (*sg*, *state*)))
→ all-vars-disjoint-or-subsetp (*p*,
cdr (generalize (*sg*,
state)),
domain-1) **endlet endlet endlet endlet**

```
;; Finally, all that's left is
;; MAIN-HYPS-RELIEVED-6-REST-GENERALIZATION.  An attempted proof by
;; induction of that theorem results in 11 goals, all but one of which
;; goes through automatically.  The remaining one is as follows.

#|

(IMPLIES
 (AND
```

```
  (DISJOINT NEW-FREE
            (INTERSECTION DOMAIN-1
                          (ALL-VARS T X)))
  (THEOREM-LIST
   (SUBST F
          (APPEND (RESTRICT S DOMAIN-1)
                  (APPLY-TO-SUBST (NULLIFY-SUBST SG)
                                  (CO-RESTRICT S DOMAIN-1)))
          Z))
  (MAPPING SG)
  (VARIABLE-LISTP (DOMAIN SG))
  (TERMP F (RANGE SG))
  (MAPPING S)
  (VARIABLE-LISTP (DOMAIN S))
  (TERMP F (RANGE S))
  (SUBSETP (DOMAIN S) NEW-FREE)
  (TERMP T X)
  (TERMP F Z)
  (THEOREM (SUBST T S X))
  (THEOREM-LIST (SUBST F S Z))
  (DISJOINT (DOMAIN SG) (ALL-VARS T X))
  (DISJOINT (DOMAIN SG)
            (ALL-VARS F
                      (GOALS-DISJOINT-FROM-VARS Z NEW-FREE DOMAIN-1)))
  (ALL-VARS-DISJOINT-OR-SUBSETP Z NEW-FREE DOMAIN-1))
 (THEOREM (SUBST T
                 (APPEND (RESTRICT S DOMAIN-1)
                         (APPLY-TO-SUBST (NULLIFY-SUBST SG)
                                         (CO-RESTRICT S DOMAIN-1)))
                 X)))

|#

;; Let us attempt to prove this goal with the proof-checker, thus
;; seeing why the rewriter can't handle it automatically.  We would
;; like to rewrite with SUBST-APPEND-NOT-OCCUR-1 to replace the
;; conclusion with:

#|
(THEOREM (SUBST T
                (APPLY-TO-SUBST (NULLIFY-SUBST SG)
                                (CO-RESTRICT S DOMAIN-1))
                X))
|#
```

```
;; However, in order to do that we need to observe that under the
;; hypotheses, the following holds.

#|
(DISJOINT (ALL-VARS F
                   (DOMAIN (RESTRICT S DOMAIN-1)))
          (ALL-VARS T X))
|#

;; This is one of those cases of a problem with free variables in
;; hypotheses that are so annoying.  The lemma DOMAIN-RESTRICT has
;; been put in alists.events to help with this.  But then we lose the
;; fact that the first ALL-VARS in the goal above may be removed.  The
;; lemma VARIABLE-LISTP-INTERSECTION has been added to terms.events to
;; take care of that.

;; Now it looks like the rewrite using SUBST-APPEND-NOT-OCCUR-1 should
;; succeed, since all hypotheses are relieved by rewriting alone.
;; Just to make sure, we back up in the proof checker and see if BASH
;; uses this rule on our original goal.  Sure enough, it does.

;; Now our conclusion is the one displayed above, i.e.

#|
(THEOREM (SUBST T
                (APPLY-TO-SUBST (NULLIFY-SUBST SG)
                                (CO-RESTRICT S DOMAIN-1))
                X))
|#

;; Since (as we already know) (NULLIFY-SUBST SG) has the same domain
;; as does SG, and since the hypotheses imply that (DOMAIN SG) is
;; disjoint from the variables of X, the SUBST expression in this
;; conclusion should simplify to:

#|
(SUBST T (NULLIFY-SUBST SG)
       (SUBST T (CO-RESTRICT S DOMAIN-1)
              X))
|#

;; We therefore need the lemma SUBST-APPLY-TO-SUBST-ELIMINATOR below
;; (which is used under the substitution where S gets (CO-RESTRICT S
```

```
;; DOMAIN-1) and SG gets (NULLIFY-SUBST SG)).  However, we'll
;; immediately derive the desired consequence and then disable this
;; lemma, since it appears that it would loop with
;; COMPOSE-PROPERTY-REVERSED.

;; << 55 >>
```

THEOREM: subst-apply-to-subst-eliminator
(variable-listp (domain (*sg*))
 ∧ variable-listp (domain (*s*))
 ∧ termp (**t**, *x*)
 ∧ disjoint (domain (*sg*), all-vars (**t**, *x*)))
→ (subst (**t**, apply-to-subst (*sg*, *s*), *x*) = subst (**t**, *sg*, subst (**t**, *s*, *x*)))

```
;; << 56 >>
```

THEOREM: theorem-subst-apply-to-subst-with-disjoint-domain
(var-substp (*sg*)
 ∧ var-substp (*s*)
 ∧ termp (**t**, *x*)
 ∧ disjoint (domain (*sg*), all-vars (**t**, *x*))
 ∧ theorem (subst (**t**, *s*, *x*)))
→ theorem (subst (**t**, apply-to-subst (*sg*, *s*), *x*))

```
;; << 57 >>
```

EVENT: Disable subst-apply-to-subst-eliminator.

```
;; The proof of the remaining goal should go through now, one might
;; think.  However, we need one more observation first, because we
;; need to apply the following lemma.

#|
(PROVE-LEMMA SUBST-CO-RESTRICT
             (REWRITE)
             (IMPLIES (AND (DISJOINT X
                                     (INTERSECTION (DOMAIN S)
                                                   (ALL-VARS FLG TERM)))
                           (VARIABLE-LISTP (DOMAIN S))
                           (TERMP FLG TERM))
                      (EQUAL (SUBST FLG (CO-RESTRICT S X) TERM)
                             (SUBST FLG S TERM))))
|#
```

```
;; but the first hypothesis of this lemma needs special handling because of
;; free variables.  The lemma DISJOINT-SUBSETP-HACK was proved at this point,
;; and appears now in sets.events.

;; And finally, we finish.  During polishing I suddenly needed the
;; lemma SUBSETP-INTERSECTION-MONOTONE-2, which is now included in
;; "sets.events", and which in turn suggested
;; SUBSETP-INTERSECTION-COMMUTER there.

;; << 58 >>
```

THEOREM: main-hyps-relieved-6-rest-generalization
let *s1* **be** restrict (*s*, *domain-1*),
    s2 **be** apply-to-subst (nullify-subst (*sg*), co-restrict (*s*, *domain-1*))
in
(var-substp (*sg*)
 ∧ var-substp (*s*)
 ∧ subsetp (domain (*s*), *new-free*)
 ∧ termp (**f**, *p*)
 ∧ theorem-list (subst (**f**, *s*, *p*))
 ∧ disjoint (domain (*sg*),
 all-vars (**f**,
 goals-disjoint-from-vars (*p*,
 new-free,
 domain-1)))
 ∧ all-vars-disjoint-or-subsetp (*p*, *new-free*, *domain-1*))
→ theorem-list (subst (**f**, append (*s1*, *s2*), *p*)) **endlet**

```
;; Now to clean up the goals that have been pushed above:

;; << 59 >>
```

THEOREM: main-hyps-relieved-6-rest
let *g* **be** caar (*state*),
    p **be** cdar (*state*),
    free **be** cdr (*state*),
    s **be** witnessing-instantiation (generalize (*sg*, *state*))
in
let *new-g* **be** subst (**t**, invert (*sg*), *g*)
in
let *domain-1* **be** gen-closure (cons (*new-g*, *p*),
 free,
 all-vars (**t**, *new-g*))
in
let *s1* **be** restrict (*s*, *domain-1*),

s2 **be** apply-to-subst (nullify-subst (*sg*),
co-restrict (*s*, *domain-1*))
in
(generalize-okp (*sg*, *state*)
∧ valid-state (generalize (*sg*, *state*)))
→ theorem-list (subst (**f**, append (*s1*, *s2*), *p*)) **endlet endlet endlet endlet**

```
;; << 60 >>
```

THEOREM: main-hyps-relieved-6
(generalize-okp (*sg*, *state*) ∧ valid-state (generalize (*sg*, *state*)))
→ theorem-list (subst (**f**,
append (restrict (witnessing-instantiation (generalize (*sg*,
state)),
gen-closure (cons (subst (**t**,
invert (*sg*),
caar (*state*)),
cdar (*state*)),
cdr (*state*),
all-vars (**t**,
subst (**t**,
invert (*sg*),
caar (*state*))))),
apply-to-subst (nullify-subst (*sg*),
co-restrict (witnessing-instantiation (generalize (*sg*,
state)),
gen-closure (cons (subst (**t**,
invert (*sg*),
caar (*state*)),
cdar (*state*)),
cdr (*state*),
all-vars (**t**,
subst (**t**,
invert (*sg*),
caar (*state*)))))))),
cons (subst (**t**, invert (*sg*), caar (*state*)), cdar (*state*))))

```
;; << 61 >>
```

THEOREM: main-hyps-relieved
(generalize-okp (*sg*, *state*) ∧ valid-state (generalize (*sg*, *state*)))
→ main-hyps (restrict (witnessing-instantiation (generalize (*sg*, *state*)),
gen-closure (cons (subst (**t**, invert (*sg*), caar (*state*)),
cdar (*state*)),
cdr (*state*),

all-vars (**t**,
subst (**t**,
invert (*sg*),
caar (*state*)))),
apply-to-subst (nullify-subst (*sg*),
co-restrict (witnessing-instantiation (generalize (*sg*,
state)),
gen-closure (cons (subst (**t**,
invert (*sg*),
caar (*state*)),
cdar (*state*)),
cdr (*state*),
all-vars (**t**,
subst (**t**,
invert (*sg*),
caar (*state*))))),
sg,
caar (*state*),
cdar (*state*))

```
;; << 62 >>
```

THEOREM: main-theorem-1-case-4
(generalize-okp (*sg*, *state*) ∧ valid-state (generalize (*sg*, *state*)))
→ theorem-list (subst (**f**, gen-inst (*sg*, *state*), car (*state*)))

```
;; << 63 >>
```

THEOREM: main-theorem-1
let *wit* **be** gen-inst (*sg*, *state*)
in
(generalize-okp (*sg*, *state*) ∧ valid-state (generalize (*sg*, *state*)))
→ (statep (*state*)
∧ var-substp (*wit*)
∧ subsetp (domain (*wit*), cdr (*state*))
∧ theorem-list (subst (**f**, *wit*, car (*state*)))) **endlet**

```
;; << 64 >>
```

THEOREM: generalize-is-correct
(generalize-okp (*sg*, *state*) ∧ valid-state (generalize (*sg*, *state*)))
→ valid-state (*state*)

Index