```
;; 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.
```

EVENT: Start with the library "terms".

```
#| 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 $(\text{car}\,(x)) \wedge$ theorem-list $(\text{cdr}\,(x))$
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*)
= (listp (*state*) ∧ termp (**f**, car (*state*)) ∧ variable-listp (cdr (*state*)))

```
;; << 5 >>
```

DEFINITION:
valid-state (*state*)
↔ (statep (*state*)
∧ ∃ *witnessing-instantiation* (var-substp (*witnessing-instantiation*)
∧ subsetp (domain (*witnessing-instantiation*),
cdr (*state*))
∧ theorem-list (subst (**f**,
witnessing-instantiation,
car (*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
 (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*)
$\wedge$ **let** *witnessing-instantiation* **be** witnessing-instantiation (*state*)
in
var-substp (*witnessing-instantiation*)
$\wedge$ subsetp (domain (*witnessing-instantiation*),
cdr (*state*))
$\wedge$ 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*) $\wedge$ valid-state (generalize (*sg*, *state*)))
$\rightarrow$ 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*))
$\rightarrow$ 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*))
∧ disjoint (all-vars (*flg*, *p*), domain (*sg*)))
→ (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*)
→ theorem-list (subst (**f**,
apply-to-subst (apply-to-subst (*s2*, *sg*), append (*s1*, *s2*)),
p))

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

THEOREM: main-hyps-suffice
(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))

```
;; 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 $new\text{-}g$ **be** subst (**t**, invert (sg), g)
in
let $domain\text{-}1$ **be** gen-closure (cons ($new\text{-}g$, p),
$free$,
all-vars (**t**, $new\text{-}g$))
in
let $s1$ **be** restrict (s, $domain\text{-}1$)
in
(generalize-okp (sg, $state$)
$\wedge$ valid-state (generalize (sg, $state$))
$\wedge$ ($wit \in$ all-vars (**f**, range (sg)))
$\wedge$ ($wit \in$ domain (s)))
$\rightarrow$ ($wit \notin domain\text{-}1$) **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 $new\text{-}g$ **be** subst (**t**, invert (sg), g)
in
let $domain\text{-}1$ **be** gen-closure (cons ($new\text{-}g$, p),
$free$,
all-vars (**t**, $new\text{-}g$))
in
let $s1$ **be** restrict (s, $domain\text{-}1$)
in
(generalize-okp (sg, $state$)
$\wedge$ valid-state (generalize (sg, $state$)))
$\rightarrow$ 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-generalization:
   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*)

EVENT: Make the library `"generalize"`.

Index