```
;; Requires sets and alists libraries.
```

EVENT: Start with the library "alists".

```
;; 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
$(x \simeq \mathbf{nil}) \rightarrow (\text{variable-listp}\,(x) = (x = \mathbf{nil}))$

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 (*flg*, x)
= **if** *flg*
then if variablep (x) **then t**
elseif listp (x)
then function-symbol-p (car (x)) $\wedge$ termp (**f**, cdr (x))
else f endif
elseif listp (x) **then** termp (**t**, car (x)) $\wedge$ termp (**f**, cdr (x))
else x = **nil endif**

THEOREM: termp-list-cons
termp (**f**, cons (a, x)) = (termp (**t**, a) $\wedge$ termp (**f**, x))

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

THEOREM: termp-t-cons
flg $\rightarrow$ (termp (*flg*, cons (a, x)) = (function-symbol-p (a) $\wedge$ termp (**f**, x)))

THEOREM: termp-t-nlistp
(*flg* $\wedge$ ($\neg$ listp (x))) $\rightarrow$ (termp (*flg*, x) = variablep (x))

EVENT: Disable termp.

THEOREM: termp-list-implies-properp
termp (**f**, x) $\rightarrow$ properp (x)

DEFINITION:
all-vars (*flg*, x)
= **if** *flg*
then if variablep (x) **then** list (x)
elseif listp (x) **then** all-vars (**f**, cdr (x))
else nil endif
elseif listp (x) **then** append (all-vars (**t**, car (x)), all-vars (**f**, cdr (x)))
else nil endif

THEOREM: properp-all-vars
properp (all-vars (*flg*, x))

THEOREM: all-vars-list-cons
all-vars (**f**, cons (a, x)) = append (all-vars (**t**, a), all-vars (**f**, x))

THEOREM: all-vars-t-cons
flg → (all-vars (*flg*, cons (a, x)) = all-vars (**f**, x))

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

THEOREM: all-vars-subsetp-append-hack
(*flg1* ∧ *flg2*)
→ (subsetp (all-vars (*flg1*, x), append (all-vars (*flg2*, x), y))
∧ subsetp (all-vars (*flg1*, x), append (y, all-vars (*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
flg → (all-vars (*flg*, x) = all-vars (**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) → variable-listp (set-diff (x, y))

THEOREM: all-vars-variablep
(*flg* ∧ variablep (x)) → (all-vars (*flg*, x) = list (x))

THEOREM: member-variable-listp-implies-variablep
(($a \in x$) ∧ variable-listp (x)) → 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*) ∨ variable-listp (*y*))
→ variable-listp (intersection (*x*, *y*))

THEOREM: variable-listp-domain-restrict
variable-listp (domain (*s*)) → variable-listp (domain (restrict (*s*, *x*)))

THEOREM: variable-listp-domain-co-restrict
variable-listp (domain (*s*)) → variable-listp (domain (co-restrict (*s*, *x*)))

THEOREM: termp-range-restrict
termp (**f**, range (*s*)) → termp (**f**, range (restrict (*s*, *x*)))

THEOREM: termp-range-co-restrict
termp (**f**, range (*s*)) → termp (**f**, range (co-restrict (*s*, *x*)))

THEOREM: member-preserves-disjoint-all-vars-lemma
(disjoint (*y*, all-vars (**f**, *x*)) ∧ (*g* ∈ *x*)) → disjoint (*y*, all-vars (**t**, *g*))

THEOREM: member-preserves-disjoint-all-vars
(*flg* ∧ disjoint (*y*, all-vars (**f**, *x*)) ∧ (*g* ∈ *x*))
→ disjoint (*y*, all-vars (*flg*, *g*))

THEOREM: member-all-vars-subsetp
(*flg* ∧ (*a* ∈ *x*)) → subsetp (all-vars (*flg*, *a*), all-vars (**f**, *x*))

THEOREM: all-vars-f-monotone
subsetp (*x*, *y*) → subsetp (all-vars (**f**, *x*), all-vars (**f**, *y*))

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

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

DEFINITION:
subst (*flg*, *s*, *x*)
= **if** *flg*
then if *x* ∈ 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*)

```
;; Later we may wish to prove correctness of one-way-unify
```

DEFINITION:
instance (*flg*, *term1*, *term2*)
↔ ∃ *one-way-unifier* (var-substp (*one-way-unifier*)
∧ (*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* ≃ **nil**) → (subst (**f**, *s*, *x*) = **nil**)

THEOREM: subst-t-variablep
(*flg* ∧ variablep (*x*))
→ (subst (*flg*, *s*, *x*)
= **if** *x* ∈ domain (*s*) **then** value (*x*, *s*)
else *x* **endif**)

THEOREM: subst-t-non-variablep
flg → (subst (*flg*, *s*, cons (*fn*, *x*))
= **if** cons (*fn*, *x*) ∈ domain (*s*) **then** value (cons (*fn*, *x*), *s*)
else cons (*fn*, subst (**f**, *s*, *x*)) **endif**)

THEOREM: all-vars-subst-lemma
(*flg* ∧ (*x* ∈ domain (*s*)))
→ subsetp (all-vars (*flg*, value (*x*, *s*)), all-vars (**f**, range (*s*)))

THEOREM: all-vars-subst
termp (*flg*, *x*)
→ subsetp (all-vars (*flg*, subst (*flg*, *s*, *x*)),
append (all-vars (*flg*, *x*), all-vars (**f**, 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
$\mathit{flg} \rightarrow (((\text{subst}(\mathit{flg}, s, x) = \text{subst}(\mathbf{t}, s, x)) = \mathbf{t})$
$\qquad \wedge \quad ((\text{subst}(\mathbf{t}, s, x) = \text{subst}(\mathit{flg}, s, x)) = \mathbf{t}))$

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

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

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

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

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

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

THEOREM: var-substp-apply-to-subst
(termp (**f**, range (*s*)) ∧ termp (**f**, range (*sg*)))
→ termp (**f**, range (apply-to-subst (*sg*, *s*)))

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

THEOREM: non-variablep-not-member-of-variable-listp
(variable-listp (*d*) ∧ (¬ variablep (*term*))) → (*term* ∉ *d*)

THEOREM: compose-property
(variable-listp (domain (*s2*)) ∧ termp (*flg*, *x*))
→ (subst (*flg*, compose (*s1*, *s2*), *x*) = subst (*flg*, *s2*, subst (*flg*, *s1*, *x*)))

THEOREM: compose-property-reversed
(variable-listp (domain (*s2*)) ∧ termp (*flg*, *x*))
→ (subst (*flg*, *s2*, subst (*flg*, *s1*, *x*)) = subst (*flg*, 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*))

EVENT: Disable instance.

```
;;;(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: instance, var-substp, compose, apply-to-subst, subst, nullify-subst.

EVENT: Make the library `"terms"`.

Index