```
#|

 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.
|#
```

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

```
;  This file contains examples that illustrate the use
;  of the new events CONSTRAIN and FUNCTIONALLY-INSTANTIATE.

;  Example 1.  Here we simply introduce three new function symbols of
;  1 argument with no constraints on them.
```

CONSERVATIVE AXIOM: p-q-r-intro
**t**

Simultaneously, we introduce the new function symbols $p$, $q$, and $r$.

```
;  Example 2.  Here we introduce the function h, which has the strange
;  property we call ``commutativity2'' and show that one can ``map''
;  with such a function in a primitive recursive way (using pr-h) or
;  by using an accumulator (using pr-ac); but either way one gets the
;  same result.  We then can FUNCTIONALLY-INSTANTIATE this result
```

```
;  using times instead of h.
```

CONSERVATIVE AXIOM: intro-h
$\mathrm{h}(x, \mathrm{h}(y, z)) = \mathrm{h}(y, \mathrm{h}(x, z))$

Simultaneously, we introduce the new function symbol $h$.

DEFINITION:
$\text{pr-h}(l, z)$
$= \quad \textbf{if } l \simeq \textbf{nil} \textbf{ then } z$
$\quad\quad \textbf{else } \mathrm{h}(\mathrm{car}(l), \text{pr-h}(\mathrm{cdr}(l), z)) \textbf{ endif}$

DEFINITION:
$\text{ac-h}(l, z)$
$= \quad \textbf{if } l \simeq \textbf{nil} \textbf{ then } z$
$\quad\quad \textbf{else } \text{ac-h}(\mathrm{cdr}(l), \mathrm{h}(\mathrm{car}(l), z)) \textbf{ endif}$

THEOREM: pr-is-ac
$\text{ac-h}(l, z) = \text{pr-h}(l, z)$

DEFINITION:
$\text{pr-times}(l, z)$
$= \quad \textbf{if } l \simeq \textbf{nil} \textbf{ then } z$
$\quad\quad \textbf{else } \mathrm{car}(l) * \text{pr-times}(\mathrm{cdr}(l), z) \textbf{ endif}$

DEFINITION:
$\text{ac-times}(l, z)$
$= \quad \textbf{if } l \simeq \textbf{nil} \textbf{ then } z$
$\quad\quad \textbf{else } \text{ac-times}(\mathrm{cdr}(l), \mathrm{car}(l) * z) \textbf{ endif}$

THEOREM: pr-times-is-ac-times
$\text{ac-times}(l, z) = \text{pr-times}(l, z)$

```
;  Example 3.  This example is somewhat similar to the last one in
;  spirit.  This time we constrain the function lt to have one of the
;  properties of a simple order, define a sort on top of it, prove the
;  sort correct, and then instantiate the result with lessp for lt.
```

CONSERVATIVE AXIOM: lt-intro
$\mathrm{lt}(z, v) \rightarrow (\neg\, \mathrm{lt}(v, z))$

Simultaneously, we introduce the new function symbol $lt$.

DEFINITION:
ordered-lt $(l)$
$=$ **if** listp $(l)$
**then if** listp (cdr $(l)$)
**then if** lt (cadr $(l)$, car $(l)$) **then f**
**else** ordered-lt (cdr $(l)$) **endif**
**else t endif**
**else t endif**

DEFINITION:
addtolist-lt $(x,\ l)$
$=$ **if** listp $(l)$
**then if** lt $(x$, car $(l))$ **then** cons $(x,\ l)$
**else** cons (car $(l)$, addtolist-lt $(x$, cdr $(l)$)) **endif**
**else** list $(x)$ **endif**

DEFINITION:
sort-lt $(l)$
$=$ **if** listp $(l)$ **then** addtolist-lt (car $(l)$, sort-lt (cdr $(l)$))
**else nil endif**

THEOREM: ordered-sort-lt
ordered-lt (sort-lt $(l)$)

DEFINITION:
ordered-lessp $(l)$
$=$ **if** listp $(l)$
**then if** listp (cdr $(l)$)
**then if** cadr $(l) <$ car $(l)$ **then f**
**else** ordered-lessp (cdr $(l)$) **endif**
**else t endif**
**else t endif**

DEFINITION:
addtolist-lessp $(x,\ l)$
$=$ **if** listp $(l)$
**then if** $x <$ car $(l)$ **then** cons $(x,\ l)$
**else** cons (car $(l)$, addtolist-lessp $(x$, cdr $(l)$)) **endif**
**else** list $(x)$ **endif**

DEFINITION:
sort-lessp $(l)$
$=$ **if** listp $(l)$ **then** addtolist-lessp (car $(l)$, sort-lessp (cdr $(l)$))
**else nil endif**

THEOREM: ordered-sort-lessp
ordered-lessp (sort-lessp $(l)$)

```
;  Example 4.  We here define the familiar map function, show that it
;  distributes over append, and instantiate the result with a LAMBDA
;  that has a free variable.
```

CONSERVATIVE AXIOM: fn-intro
**t**

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

DEFINITION:
map-fn $(x)$
= **if** $x \simeq$ **nil then nil**
**else** cons (fn (car $(x)$), map-fn (cdr $(x)$)) **endif**

THEOREM: map-distributes-over-append
map-fn (append $(u,\ v)$) = append (map-fn $(u)$, map-fn $(v)$)

DEFINITION:
map-plus-y $(x,\ y)$
= **if** $x \simeq$ **nil then nil**
**else** cons (car $(x)$ + $y$, map-plus-y (cdr $(x)$, $y$)) **endif**

THEOREM: map-plus-y-distributes-over-append
map-plus-y (append $(u,\ v)$, $z$) = append (map-plus-y $(u,\ z)$, map-plus-y $(v,\ z)$)

```
;  Example 5.  Here we follow the lead of Goodstein in his book
;  Primitive Recursive Arithmetic and of McCarthy with his recursion
;  induction.  We show, using FUNCTIONALLY-INSTANTIATE, that the
;  associativity of append can be proved without explicit appeal to
;  induction.  Of course there are inductions hidden all over the
;  place, e.g. in the type-set analysis for true-rec and in the proof
;  of the metatheorem that justifies FUNCTIONALLY-INSTANTIATE.  Still,
;  this is a startling development to those who regard the
;  associativity of append as the first theorem requiring an inductive
;  proof.
```

DEFINITION:
true-rec $(x)$
= **if** $x \simeq$ **nil then t**
**else** true-rec (cdr $(x)$) **endif**

THEOREM: true-rec-is-true
true-rec $(x)$

DEFINITION:
app $(x,\ y)$
$=$ **if** $x \simeq$ **nil** **then** $y$
**else** cons (car $(x)$, app (cdr $(x)$, $y$)) **endif**

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

```
;  Example 6.  We illustrate that one can prove theorems using
;  CONSTRAIN and FUNCTIONALLY-INSTANTIATE that resemble proofs in
;  first-order predicate calculus with quantifiers.
```

CONSERVATIVE AXIOM: all-x-p-x-intro
ALL-X-P-X $\rightarrow$ p $(x)$

Simultaneously, we introduce the new function symbol *all-x-p-x*.

CONSERVATIVE AXIOM: all-x-not-p-x-into
ALL-X-NOT-P-X $\rightarrow$ ($\neg$ p $(x)$)

Simultaneously, we introduce the new function symbol *all-x-not-p-x*.

THEOREM: all-x-not-p-x-into-converse
p $(x)$ $\rightarrow$ ($\neg$ ALL-X-NOT-P-X)

DEFINITION: SOME-X-P-X $=$ ($\neg$ ALL-X-NOT-P-X)

THEOREM: all-implies-some
ALL-X-P-X $\rightarrow$ SOME-X-P-X

```
;  Example 7.  We illustrate a CONSTRAIN that expresses that a
;  function is ``fair'' in the sense that it is infinitely often true
;  and false.
```

DEFINITION:
even $(x)$
$=$ **if** $x \simeq 0$ **then t**
**elseif** $x = 1$ **then** FALSE
**else** $\neg$ even $(x - 1)$ **endif**

Conservative Axiom: fair-intro

$$\begin{aligned} & \text{fair}\,(\text{fair-true-witness}\,(n)) \\ \wedge\quad & (\neg\ \text{fair}\,(\text{fair-false-witness}\,(n))) \\ \wedge\quad & (\text{fair-true-witness}\,(n) \not< n) \\ \wedge\quad & (\text{fair-false-witness}\,(n) \not< n) \end{aligned}$$

Simultaneously, we introduce the new function symbols *fair*, *fair-true-witness*, and *fair-false-witness*.

```
;  Example 8.  We illustrate the idea of ``stubbing'' functions.  We
;  define interp to call an undefined, but constrained, function num.
;  Later we prove a result about instantiation of interp by
;  FUNCTIONALLY-INSTANTIATE.
```

Conservative Axiom: num-intro

$\text{num}\,(x) \in \mathbf{N}$

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

Definition:

$$\begin{aligned} & \text{interp}\,(x) \\ = \quad & \textbf{if } x \not\simeq \texttt{0} \textbf{ then } x * x \\ & \textbf{else } \text{num}\,(x) \textbf{ endif} \end{aligned}$$

Theorem: interp-is-numeric

$\text{interp}\,(x) \in \mathbf{N}$

Definition:

$$\begin{aligned} & \text{interp2}\,(x) \\ = \quad & \textbf{if } x \not\simeq \texttt{0} \textbf{ then } x * x \\ & \textbf{else } x + x \textbf{ endif} \end{aligned}$$

Theorem: interp2-is-numeric

$\text{interp2}\,(x) \in \mathbf{N}$

```
;  Example 9.  We here illustrate the fact that add-axioms
;  are correctly tracked.
```

Definition: $\text{p-alias}\,(x) = \text{p}\,(x)$

Axiom: even-p-alias

$\text{even}\,(\text{p-alias}\,(x))$

Theorem: even-p

$\text{even}\,(\text{p}\,(x))$

```
#|

;  Because this step fails, we comment it out.  The failure
;  stems from our having provided a substitution for the
;  apparently irrelevant function q-alias.  However, providing
;  an analogue to q-alias explose the real problem, the necessity
;  of anopther add-axiom.  The key point is that p-alias
;  is not irrelevant when trying to FUNCTIONALLY-INSTANTIATE
;  even-p because p is ancestral in even-p-alias.

(functionally-instantiate even-q nil
  (even (q x))
  even-p
  ((p q)))

|#

;  Example 10.  It is necessary for soundness that we check that
;  the variables in the constraints do not intersect the free variables
;  in the FUNCTIONALLY-INSTANTIATE substitutions.  Otherwise,
;  the following sequence would lead to unsoundness.
```

Conservative Axiom: pp-intro
$(y = 0) \rightarrow (\text{PP} = y)$

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

Theorem: pp-is-0
$0 = \text{PP}$

```
#|

;  Because this last step fails, we comment it out.  Note that if we
;  did not catch this, relieving the constraint would amount to
;  checking merely that (implies (equal y 0) (equal y y)).

(functionally-instantiate anything-is-0 (rewrite)
  (equal 0 y)
  pp-is-0
  ((pp (lambda () y))))

|#

;  Example 11.  Some from ``higher logic.''
```

Conservative Axiom: fn-commutative
$\text{fn2}\,(x,\ y) = \text{fn2}\,(y,\ x)$

Simultaneously, we introduce the new function symbol *fn2*.

Definition:
$\text{foldr-fn}\,(lst,\ r)$
$=$ **if** $\text{listp}\,(lst)$ **then** $\text{fn2}\,(\text{car}\,(lst),\ \text{foldr-fn}\,(\text{cdr}\,(lst),\ r))$
**else** $r$ **endif**

Definition:
$\text{foldl-fn}\,(lst,\ r)$
$=$ **if** $\text{listp}\,(lst)$ **then** $\text{foldl-fn}\,(\text{cdr}\,(lst),\ \text{fn2}\,(r,\ \text{car}\,(lst)))$
**else** $r$ **endif**

Definition:
$\text{reverse}\,(x)$
$=$ **if** $\text{listp}\,(x)$ **then** $\text{append}\,(\text{reverse}\,(\text{cdr}\,(x)),\ \text{list}\,(\text{car}\,(x)))$
**else nil endif**

Theorem: foldl-is-foldr
$\text{foldr-fn}\,(lst,\ r) = \text{foldl-fn}\,(\text{reverse}\,(lst),\ r)$

Theorem: times-add1
$(x * (1 + y)) = (x + (x * y))$

Theorem: times-comm
$(x * y) = (y * x)$

Definition:
$\text{foldr-times}\,(lst,\ r)$
$=$ **if** $\text{listp}\,(lst)$ **then** $\text{car}\,(lst) * \text{foldr-times}\,(\text{cdr}\,(lst),\ r)$
**else** $r$ **endif**

Definition:
$\text{foldl-times}\,(lst,\ r)$
$=$ **if** $\text{listp}\,(lst)$ **then** $\text{foldl-times}\,(\text{cdr}\,(lst),\ r * \text{car}\,(lst))$
**else** $r$ **endif**

Theorem: foldl-times-is-foldr-times
$\text{foldr-times}\,(lst,\ r) = \text{foldl-times}\,(\text{reverse}\,(lst),\ r)$

# Index