EVENT: Start with the initial \texttt{nqthm} theory.

; This file contains examples that illustrate the use
; of the new events \texttt{CONSTRAIN} and \texttt{FUNCTIONALLY-INSTANTIATE}.

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

\textbf{Conservative Axiom: p-q-r-intro} \texttt{t}

Simultaneously, we introduce the new function symbols \texttt{p}, \texttt{q}, and \texttt{r}.

; Example 2. Here we introduce the function \texttt{h}, which has the strange
; property we call \textquote{commutativity2} and show that one can \textquote{map}
; with such a function in a primitive recursive way (using \texttt{pr-h}) or
; by using an accumulator (using \texttt{pr-ac}); but either way one gets the
; same result. We then can \texttt{FUNCTIONALLY-INSTANTIATE} this result
Conservative Axiom: intro-h
\[ h(x, h(y, z)) = h(y, h(x, z)) \]

Simultaneously, we introduce the new function symbol \( h \).

Definition:
\[
pr-h(l, z) = \begin{cases} 
  z & \text{if } l \simeq \text{nil} \\
  h(\text{car}(l), pr-h(\text{cdr}(l), z)) & \text{else}
\end{cases}
\]

Definition:
\[
ac-h(l, z) = \begin{cases} 
  z & \text{if } l \simeq \text{nil} \\
  ac-h(\text{cdr}(l), h(\text{car}(l), z)) & \text{else}
\end{cases}
\]

Theorem: pr-is-ac
\[ ac-h(l, z) = pr-h(l, z) \]

Definition:
\[
pr-times(l, z) = \begin{cases} 
  z & \text{if } l \simeq \text{nil} \\
  \text{car}(l) \ast pr-times(\text{cdr}(l), z) & \text{else}
\end{cases}
\]

Definition:
\[
ac-times(l, z) = \begin{cases} 
  z & \text{if } l \simeq \text{nil} \\
  ac-times(\text{cdr}(l), \text{car}(l) \ast z) & \text{else}
\end{cases}
\]

Theorem: pr-times-is-ac-times
\[ ac-times(l, z) = pr-times(l, z) \]

; 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
\[ lt(z, v) \rightarrow (\neg lt(v, z)) \]

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

Simultaneously, we introduce the new function symbol fn.

Definition:
map-fn (x) = if x ≃ 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 ≃ 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 ≃ nil then t
               else true-rec (cdr (x)) endif
Theorem: true-rec-is-true
true-rec (x)

Definition:
app (x, y) = if x ≃ 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 → 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 → (¬ p (x))

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

Theorem: all-x-not-p-x-into-converse
p (x) → (¬ ALL-X-NOT-P-X)

Definition: some-x-p-x = (¬ ALL-X-NOT-P-X)

Theorem: all-implies-some
all-x-p-x → 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 ≃ 0 then t
elseif x = 1 then false
else ¬ even (x − 1) endif
**Conservative Axiom: fair-intro**

\begin{align*}
& \text{fair (fair-true-witness (n))} \\
& \land (\neg \text{fair (fair-false-witness (n)))} \\
& \land (\text{fair-true-witness (n) } \not< n) \\
& \land (\text{fair-false-witness (n) } \not< n)
\end{align*}

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**

\begin{align*}
& \text{num (x)} \in \mathbb{N}
\end{align*}

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

**Definition**: 

\begin{align*}
\text{interp (x)} &= \begin{cases} 
\text{x} \times \text{x} & \text{if } x \not\equiv 0 \\
\text{num (x)} & \text{else}
\end{cases}
\end{align*}

**Theorem**: interp-is-numeric

\begin{align*}
\text{interp (x)} \in \mathbb{N}
\end{align*}

**Definition**: 

\begin{align*}
\text{interp2 (x)} &= \begin{cases} 
\text{x} \times \text{x} & \text{if } x \not\equiv 0 \\
\text{x} + \text{x} & \text{else}
\end{cases}
\end{align*}

**Theorem**: interp2-is-numeric

\begin{align*}
\text{interp2 (x)} \in \mathbb{N}
\end{align*}

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

**Definition**: 

\begin{align*}
p\text{-alias (x)} &= p (x)
\end{align*}

**Axiom**: even-p-alias

\begin{align*}
even (p\text{-alias (x)})
\end{align*}

**Theorem**: even-p

\begin{align*}
even (p (x))
\end{align*}
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 expose the real problem, the necessity of another 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) → (pp = y)

Simultaneously, we introduce the new function symbol pp.

Theorem: pp-is-0
0 = pp

(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
\[ fn2(x, y) = fn2(y, x) \]

Simultaneously, we introduce the new function symbol \( fn2 \).

**Definition:**
\[ \text{foldr-fn}(lst, r) = \begin{cases} \text{fn2(car(lst), foldr-fn(cdr(lst), r))} & \text{if listp(lst)} \\ r & \text{else} \end{cases} \]

**Definition:**
\[ \text{foldl-fn}(lst, r) = \begin{cases} \text{foldl-fn(cdr(lst), fn2(r, car(lst)))} & \text{if listp(lst)} \\ r & \text{else} \end{cases} \]

**Definition:**
\[ \text{reverse}(x) = \begin{cases} \text{append(reverse(cdr(x)), list(car(x)))} & \text{if listp(x)} \\ \text{nil} & \text{else} \end{cases} \]

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

**Theorem:** times-add1
\[ (x \times (1 + y)) = (x + (x \times y)) \]

**Theorem:** times-comm
\[ (x \times y) = (y \times x) \]

**Definition:**
\[ \text{foldr-times}(lst, r) = \begin{cases} \text{car(lst) \times foldr-times(cdr(lst), r)} & \text{if listp(lst)} \\ r & \text{else} \end{cases} \]

**Definition:**
\[ \text{foldl-times}(lst, r) = \begin{cases} \text{foldl-times(cdr(lst), r \times car(lst))} & \text{if listp(lst)} \\ r & \text{else} \end{cases} \]

**Theorem:** foldl-times-is-foldr-times
\[ \text{foldr-times}(lst, r) = \text{foldl-times(reverse(lst), r)} \]
## Index

<table>
<thead>
<tr>
<th>ac-h</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>ac-times</td>
<td>2</td>
</tr>
<tr>
<td>addtolist-lessp</td>
<td>3</td>
</tr>
<tr>
<td>addtolist-lt</td>
<td>3</td>
</tr>
<tr>
<td>all-implies-some</td>
<td>5</td>
</tr>
<tr>
<td>all-x-not-p-x</td>
<td>5</td>
</tr>
<tr>
<td>all-x-not-p-x-into</td>
<td>5</td>
</tr>
<tr>
<td>all-x-not-p-x-into-converse</td>
<td>5</td>
</tr>
<tr>
<td>all-x-p-x</td>
<td>5</td>
</tr>
<tr>
<td>all-x-p-x-intro</td>
<td>5</td>
</tr>
<tr>
<td>app</td>
<td>5</td>
</tr>
<tr>
<td>assoc-of-app</td>
<td>5</td>
</tr>
<tr>
<td>even</td>
<td>5, 6</td>
</tr>
<tr>
<td>even-p</td>
<td>6</td>
</tr>
<tr>
<td>even-p-alias</td>
<td>6</td>
</tr>
<tr>
<td>fair</td>
<td>6</td>
</tr>
<tr>
<td>fair-false-witness</td>
<td>6</td>
</tr>
<tr>
<td>fair-intro</td>
<td>6</td>
</tr>
<tr>
<td>fair-true-witness</td>
<td>6</td>
</tr>
<tr>
<td>fn</td>
<td>4</td>
</tr>
<tr>
<td>fn-commutative</td>
<td>8</td>
</tr>
<tr>
<td>fn-intro</td>
<td>4</td>
</tr>
<tr>
<td>fn2</td>
<td>8</td>
</tr>
<tr>
<td>foldl-fn</td>
<td>8</td>
</tr>
<tr>
<td>foldl-is-foldr</td>
<td>8</td>
</tr>
<tr>
<td>foldl-times</td>
<td>8</td>
</tr>
<tr>
<td>foldl-times-is-foldr-times</td>
<td>8</td>
</tr>
<tr>
<td>foldr-fn</td>
<td>8</td>
</tr>
<tr>
<td>foldr-times</td>
<td>8</td>
</tr>
<tr>
<td>h</td>
<td>2</td>
</tr>
<tr>
<td>interp</td>
<td>6</td>
</tr>
<tr>
<td>interp-is-numeric</td>
<td>6</td>
</tr>
<tr>
<td>interp2</td>
<td>6</td>
</tr>
<tr>
<td>interp2-is-numeric</td>
<td>6</td>
</tr>
<tr>
<td>intro-h</td>
<td>2</td>
</tr>
<tr>
<td>lt</td>
<td>2, 3</td>
</tr>
<tr>
<td>lt-intro</td>
<td>2</td>
</tr>
<tr>
<td>map-distributes-over-append</td>
<td>4</td>
</tr>
<tr>
<td>map-fn</td>
<td>4</td>
</tr>
<tr>
<td>map-plus-y</td>
<td>4</td>
</tr>
<tr>
<td>map-plus-y-distributes-over-append</td>
<td>4</td>
</tr>
<tr>
<td>num</td>
<td>6</td>
</tr>
<tr>
<td>num-intro</td>
<td>6</td>
</tr>
<tr>
<td>ordered-lessp</td>
<td>3, 4</td>
</tr>
<tr>
<td>ordered-lt</td>
<td>3</td>
</tr>
<tr>
<td>ordered-sort-lessp</td>
<td>4</td>
</tr>
<tr>
<td>ordered-sort-lt</td>
<td>3</td>
</tr>
<tr>
<td>p</td>
<td>5, 6</td>
</tr>
<tr>
<td>p-alias</td>
<td>6</td>
</tr>
<tr>
<td>p-q-r-intro</td>
<td>1</td>
</tr>
<tr>
<td>pp</td>
<td>7</td>
</tr>
<tr>
<td>pp-intro</td>
<td>7</td>
</tr>
<tr>
<td>pp-is-0</td>
<td>7</td>
</tr>
<tr>
<td>pr-h</td>
<td>2</td>
</tr>
<tr>
<td>pr-is-ac</td>
<td>2</td>
</tr>
<tr>
<td>pr-times</td>
<td>2</td>
</tr>
<tr>
<td>pr-times-is-ac-times</td>
<td>2</td>
</tr>
<tr>
<td>reverse</td>
<td>8</td>
</tr>
<tr>
<td>some-x-p-x</td>
<td>5</td>
</tr>
<tr>
<td>sort-lessp</td>
<td>3, 4</td>
</tr>
<tr>
<td>sort-lt</td>
<td>3</td>
</tr>
<tr>
<td>times-add1</td>
<td>8</td>
</tr>
<tr>
<td>times-comm</td>
<td>8</td>
</tr>
<tr>
<td>true-rec</td>
<td>4, 5</td>
</tr>
<tr>
<td>true-rec-is-true</td>
<td>5</td>
</tr>
</tbody>
</table>