Consistently Adding Primitive Recursive Definitions in ACL2

by

John Cowles
University of Wyoming
defpun

A macro for consistently introducing “partial functions” into CAL2.


One of many cases handled by defpun is when the “defining equation” is tail recursive.
Tail Recursion

Let test, base, and st be arbitrary unary functions.

There always is at least one ACL2 function $f$ that satisfies

\[(\text{equal } (f \, x))\]
\[(\text{if } (\text{test } x))\]
\[\quad (\text{base } x)\]
\[\quad (f \, (\text{st } x)))\].
Tail Recursion Construction

Pete & J construct a *tail recursive* function $f$ in ACL2:

1. Define $\text{stn}$ so that $(\text{stn} \ x \ n)$ computes $(\text{st}^n \ x)$.

2. Use defchoose to define a Skolem (witnessing) function $\text{fch}$ so that

   $$(\text{fch} \ x)$$

   is an $n$ such that $(\text{test} \ (\text{stn} \ x \ n))$ holds, if such an $n$ exists.

   If no such $n$ exists, then ACL2 knows nothing about the value of $(\text{fch} \ x)$.

   If $(\text{test} \ (\text{stn} \ x \ (\text{fch} \ x)))$ holds, then $(\text{fch} \ x)$ need not be the smallest $n$ such that $(\text{test} \ (\text{stn} \ x \ n))$ holds.
Tail Recursion Construction

3. Define a version of \( f \), called \( fn \), with an extra “clock-like” input parameter, \( n \), that ensures termination:

\[
\begin{align*}
\text{(defun fn (x n)} \\
\text{ (declare (xargs :measure (nfix n)))} \\
\text{ (if (or (zp n) (test x))} \\
\text{ (base x)} \\
\text{ (fn (st x) (1- n))))}.
\end{align*}
\]

4. Finally define \( f \):

\[
\begin{align*}
\text{(defun f (x)} \\
\text{ (if (test (stn x (fch x)))} \\
\text{ (fn x (fch x))} \\
\text{ nil))}
\end{align*}
\]

Any constant would do in place of \( \text{nil} \) in this definition.
Tail Recursion Construction

(defun f (x)
  (if (test (stn x (fch x)))
      (fn x (fch x))
      nil))

ACL2 verifies that this f satisfies the tail recursive equation

(equal (f x)
  (if (test x)
      (base x)
      (f (st x)))).
Primitive Recursion

Let $h$ be a binary function.

A function $f$ satisfying an equation of the form

$$(\text{equal } (f \ x))$$
$$(\text{if } (\text{test } x))$$
$$(\text{base } x)$$
$$(h \ x \ (f \ (\text{st } x))))$$

is called \textit{primitive recursive}. 
Primitive Recursion

This definition of primitive recursive is inspired by the primitive recursive definitions studied in Theory of Computation courses:

For previously defined functions, $k$ and $h$, on the non-negative integers, define $f$ by

$$
f(\vec{x}, 0) = k(\vec{x})$$
$$f(\vec{x}, t + 1) = h(t, f(\vec{x}, t), \vec{x}).$$

Here $\vec{x} = x_1, \ldots, x_n$. 
Primitive Recursion

Extend Pete & J's tail recursive construction to many, but not all, primitive recursive defining equations.
Primitive Recursion

There are h's for which no ACL2 function f satisfies the primitive recursive defining equation:

\[
\text{(equal (f x)} \\
\text{(if (test x)} \\
\text{\hspace{1cm} (base x)} \\
\text{\hspace{1cm} (h x (f (st x))))}).
\]
Example

No ACL2 function \( g \) satisfies this \textit{primitive recursive} equation

\[
\text{(equal } (g \ x) \\
\text{(if } (\text{equal } x \ 0) \\
\text{nil \\
\text{(cons nil } (g \ (- \ x \ 1))))).
\]

Here

\bullet (\text{test } x) \text{ is } (\text{equal } x \ 0),

\bullet (\text{base } x) \text{ is } \text{nil},

\bullet (h \ x \ y) \text{ is } (\text{cons nil } y), \text{ and}

\bullet (st \ x) \text{ is } (- \ x \ 1).
Primitive Recursion

\[(\text{equal } (f \ x)) \]
\[(\text{if } (\text{test } x) \]
\[\quad (\text{base } x) \]
\[\quad (h \ x \ (f \ (\text{st } x))))]].\]

A sufficient (but not necessary) condition on \(h\) for the existence of \(f\) is that \(h\) have a right fixed point.

That is, there is some \(c\) such that \((h \ x \ c) = c\).
Primitive Recursion Construction

Modify Pete & J’s tail recursion construction.

Construct a *primitive recursive* function \( f \) in ACL2:

1. Define \( \text{stn} \) so that \( (\text{stn} \ x \ n) \) computes \( (\text{st}^n \ x) \).

   *(Same as for tail recursion.)*

2. Use \texttt{defchoose} to define a Skolem (witnessing) function \( \text{fch} \) so that

   \( (\text{fch} \ x) \) is an \( n \) such that \( (\text{test} \ (\text{stn} \ x \ n)) \) holds, if such an \( n \) exists.

   *(Same as for tail recursion.)*
Primitive Recursion Construction

3. Define a version of \( f \), called \( fn \), with an extra “clock-like” input parameter, \( n \), that ensures termination:

\[
\text{(defun fn (x n)}
\quad \text{(declare (xargs :measure (nfix n)))}
\quad \text{(if (or (zp n) (test x)))}
\quad \text{(base x)}
\quad \text{(h x (fn (st x) (1- n))))}.
\]

4. Finally define \( f \):

Here \((h\text{-fix})\) is a right fixed point for \( h \).

\[
\text{(defun f (x)}
\quad \text{(if (test (stn x (fch x)))}
\quad \text{(fn x (fch x))}
\quad \text{(h-fix)))}
\]

11-a
Primitive Recursion Construction

\[
\text{(defun f (x)}
\begin{align*}
\text{ (if (test (stn x (fch x)))} \\
\text{ (fn x (fch x))} \\
\text{ (h-fix)))}
\end{align*}
\]

ACL2 verifies that this \( f \) satisfies the \textit{primitive recursive} equation

\[
\text{(equal (f x)}
\begin{align*}
\text{ (if (test x)} \\
\text{ (base x) } \\
\text{ (h x (f (st x))))})
\end{align*}
\].
Example

A right fixed point for $h$ is *not necessary* for some primitive recursive definitions.

The ACL2 function $\text{fix}$ *satisfies* this *primitive recursive* equation

\[
\text{(equal } (\text{fix } x) \text{)} \\
\quad \text{(if } (\text{equal } x 0) \\
\quad \quad 0 \\
\quad \quad (+ 1 (\text{fix } (- x 1))))),
\]

Here

- $(\text{test } x)$ is $(\text{equal } x 0)$,
- $(\text{base } x)$ is $0$,
- $(h \ x \ y)$ is $(+ 1 \ y)$ [*no fixed point*], and
- $(\text{st } x)$ is $( - x 1)$. 


defpr

A macro for consistently introducing primitive recursive equations into ACL2.

In an encapsulate, carry out the Primitive Recursion Construction:

- \( f \) is constrained only by

\[
\text{(deffthm generic-primitive-recursive-f)}
\text{(equal (f x)}
\text{(if (test x)}
\text{(base x)}
\text{(h x (f (st x)))})\text{)}.\]

- \( h \) is constrained to have a right fixed point, (h-fix).

- test, base, and st are unconstrained.
defpr

Given the required fixed point, the defpr macro

- recognizes a primitive recursive definition, and

- generates a *functional instance* of generic-primitive-recursive-f to produce a witness to the desired primitive recursive equation.
Example

No ACL2 function $g$ satisfies this primitive recursive equation

$$(\text{equal} \ (g \ x))$$
$$(\text{if} \ (\text{equal} \ x \ 0)$$
$$\quad \text{nil}$$
$$\quad (\text{cons} \ \text{nil} \ (g \ (- \ x \ 1))))).$$

The problem: $\text{cons}$ has no right fixed point.
Example

The **problem**: \texttt{cons} has no right fixed point.

Provide a right fixed point by the following:

```
(defstub
  cons-fix () => *)
```

```
(defun
  cons$ (x y)
  (if (equal y (cons-fix))
    (cons-fix)
    (cons x y)))
```
Example

(defpr
  g (x)
  (declare (xargs :fixpt (cons-fix)))
  (if (equal x 0)
      nil
      (cons$ nil (g (- x 1)))))

produces an ACL2 solution for g:

(equal (g x)
  (if (equal x 0)
      nil
      (cons$ nil (g (- x 1)))))

Note use of XARGS keyword :fixpt to give the required fixed point.
Example

Any fixed point will do.

Multiplication already has a right fixed point, namely 0:

\((* \times 0) = 0\).

(defpr
  fact (x)
  (declare (xargs :fixpt 0))
  (if (equal x 0)
    1
    (* x (fact (- x 1)))))

produces an ACL2 solution for fact:

(equal (fact x)
  (if (equal x 0)
    1
    (* x (fact (- x 1)))))

16
Note: ACL2 accepts the definition that uses the zero-test idiom (zp x) in place of the test (equal x 0):

```
(defun fact (x)
  (if (zp x)
      1
      (* x (fact (- x 1)))))
```
Example

This succeeds: (a primitive recursive definition)

(defpr
  f (x)
  (declare (xargs :fixpt 0))
  (if (equal x 0)
    1
    (* (f (- x 1))
        (f (- x 1))))

This fails: (not a primitive recursive definition)

(defpr
  f1 (x)
  (declare (xargs :fixpt 0))
  (if (equal x 0)
    1
    (* (f1 (- x 1))
        (f1 (+ x 1)))))
Example
with parameters.

(defpr
  k (a b)
  (declare (xargs :fixpt 0))
  (if (equal b 0)
      1
      (* a b (k a (- b 1)))))

Note: On the non-negative integers

(k a b) = a^b \cdot b!
Example

Tail recursion is a special case.

The function, $\text{Id-2-2}$, defined by

$$(\text{Id-2-2} \ x1 \ x2) = x2$$

is used for $h$.

Any constant can be used for the fixed point.

(defpr
tail-f (x)
  (declare (xargs :fixpt nil))
  (if (tail-test x)
      (tail-base x)
      (Id-2-2 x (tail-f (tail-st x))))
)

defthm
tail-f-is-tail-recursive
equal (tail-f x)
  (if (tail-test x)
      (tail-base x)
      (tail-f (tail-st x))))
Conclusion

Recursive equations of the form

\[(\text{equal} \ (f \ x)\)
\[(\text{if} \ (\text{test} \ x)\)
\[(\text{base} \ x)\)
\[(h \ x \ (f \ (\text{st} \ x))))\]

are satisfiable in ACL2’s logic whenever \(h\) has a right fixed point.

Proving \(h\) has a right fixed point ensures the systematic construction of such a function \(f\).