Flat Domains and Recursive Equations in ACL2

by

John Cowles

University of Wyoming
ACL2 is a logic of total functions.

- Some recursive equations have no satisfying ACL2 functions:
  
  **No** ACL2 function \( g \) satisfies this recursive equation

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

  Theory of flat domains is a rival logic of total functions.

- Every recursive equation has at least one satisfying function.
Flat Domains

From the fix-point theory of program semantics.

A flat domain is a structure

$$< S, \sqsubseteq, \bot >$$

, where

- $S$ is a set,

- $\bot \in S$, and

- $\sqsubseteq$ is the partial order defined by

$$x \sqsubseteq y \iff x = \bot \lor x = y.$$
Graphical representation of a flat domain:

\[ S - \{\bot\} \]
\[ \bullet \ldots \bullet \bullet \bullet \ldots \bullet \]
\[ \bot \]

- Graphical representation of the \( \sqsubseteq \) relation defined by
  \[ x \sqsubseteq y \iff x \subseteq y \land x \neq y. \]

- The “flat part” is depicted by the vertices labeled with \( S - \{\bot\} \).
Extend the partial order, $\sqsubseteq$, \textit{componentwise} to

- tuples from $S \times S \times \cdots \times S$ by
  
  $< x_1, \ldots, x_n > \sqsubseteq < y_1, \ldots, y_n >$
  
  $\iff x_1 \sqsubseteq y_1 \land \cdots \land x_n \sqsubseteq y_n$

- functions $f, g : S \times \cdots \times S \to S$ by
  
  $f \sqsubseteq g \iff (\forall \bar{x} \in S^n)[f(\bar{x}) \sqsubseteq g(\bar{x})]$
Flat Domains

Use total functions to model partial functions.

• Interpret

\[ f(\vec{x}) = \bot \]

as meaning

\[ f(\vec{x}) \text{ is undefined.} \]

• Interpret, for functions \( f \) and \( g \),

\[ f \sqsubseteq g \]

as meaning

whenever \( f(\vec{x}) \) is defined,

\( \circ \) \( g(\vec{x}) \) is also defined, and

\( \circ \) \( f(\vec{x}) = g(\vec{x}). \)
Least Upper Bounds of Chains

Every chain of functions on $S$,

$$f_0 \subseteq f_1 \subseteq \cdots \subseteq f_i \subseteq \cdots,$$

has an unique least upper bound, $\sqcup f_i$.

- $\sqcup f_i$ is a function on $S$,

- for all $j$, $f_j \subseteq \sqcup f_i$ and

- if $f$ is any function such that for all $i$, $f_i \subseteq f$, then $\sqcup f_i \subseteq f$,

- define $\sqcup f_i(\vec{x})$ by cases:

  **Case 1.** $\forall i (f_i(\vec{x}) = \perp)$.  
  Let $\sqcup f_i(\vec{x}) = \perp$.

  **Case 2.** $\exists j (f_j(\vec{x}) \neq \perp)$. 
  Let $\sqcup f_i(\vec{x}) = f_j(\vec{x})$. 
Flat Domains
Recursive Equations

Let $F$ be a function variable and let $\tau[F]$ be a term built by compositions involving $F$ and other functions.

A recursive equation is of the form

$$ F(\vec{x}) = \tau[F](\vec{x}). $$

A solution for such an equation is a function $f$ such that for all $\vec{x}$,

$$ f(\vec{x}) = \tau[f](\vec{x}). $$

Such a solution $f$ is called a fixed point of the term $\tau[F](\vec{x})$. 
Flat Domains

The Kleene Construction

A term $\tau[F]$ is monotonic:

- Whenever $f$ and $g$ are functions such that $f \subseteq g$, then $\tau[f] \subseteq \tau[g]$.

Kleene’s construction:

- When $\tau[F]$ is monotonic,

\[
F(\overrightarrow{x}) = \tau[F](\overrightarrow{x})
\]

always has a solution.
Flat Domains

The Kleene Construction

Kleene’s construction:

• Use the term $\tau[F]$ to recursively define a chain of functions,

\[
\begin{align*}
    f_0(\vec{x}) &= \bot \\
    f_{i+1}(\vec{x}) &= \tau[f_i](\vec{x}).
\end{align*}
\]

• Since $\tau[F]$ is monotonic,

\[
f_0 \sqsubseteq f_1 \sqsubseteq \cdots \sqsubseteq f_i \sqsubseteq \cdots
\]

• Then,

\[
\sqcup f_i = \tau[\sqcup f_i].
\]

That is, $\sqcup f_i$ is a solution for the recursive equation $F(\vec{x}) = \tau[F](\vec{x})$. 

9-a
Turn ACL2 data into a flat domain

Impose a partial order, $\leq$, on ACL2 data:

- specify a “least element”, ($\texttt{bottom}$), strictly less than any other ACL2 datum

  \[
  \text{(defstub)} \quad \texttt{bottom} () \Rightarrow * \]

- no other distinct data items are related:

  \[
  \text{(defun)} \quad \texttt{<=} (x y) \\
  \quad (\text{or} \ (\text{equal} \ x \ (\texttt{bottom})) \\
  \quad \quad \ (\text{equal} \ x \ y))
  \]

- ($\texttt{bottom}$) plays the part of $\bot$ and $\leq$ plays the part of $\sqsubseteq$. 
Chains of functions in ACL2

Formalize a chain of functions

\[ f_0 \sqsubseteq f_1 \sqsubseteq \cdots \sqsubseteq f_i \sqsubseteq \cdots. \]

- Treat the index as an additional argument to the function, so \( f_i(x) \) becomes \( (f \ i \ x) \) in ACL2.

- The \( \leq \)-chain of functions is consistently axiomatized by

\[
(\text{implies} \ (\text{and} \ (\text{integerp} \ i) \\
\quad (\geq \ i \ 0)) \\
\quad (\leq \ (f \ i \ x) \\
\quad \quad (f \ (+ \ 1 \ i) \ x))).
\]
Chains of functions in ACL2

Formalize the least upper bound, $\sqcup f_i$, of

$$f_0 \sqsubseteq f_1 \sqsubseteq \cdots \sqsubseteq f_i \sqsubseteq \cdots.$$ 

- Use `defchoose` to pick the appropriate “index” required in the definition of the least upper bound.

- ACL2 verifies this formal least upper bound is, in fact, the least upper bound of the chain.
Which ACL2 terms are monotonic?

Recall:

To ensure that Kleene’s construction always produces

- a solution for the recursive equation
  \[ F(\vec{x}) = \tau[F](\vec{x}), \]

- the term \( \tau[F] \) must be monotonic:
  \[ f \sqsubseteq g \Rightarrow \tau[f] \sqsubseteq \tau[g]. \]
Which ACL2 terms are monotonic?

**Tail Recursion.** Let \( \text{test} \), \( \text{base} \), and \( \text{st} \) be arbitrary unary functions.

Consider a term \( \tau[F] \) of the form

\[
(\text{if} (\text{test} \ x)
  (\text{base} \ x)
  (F (\text{st} \ x)))).
\]

Such *tail recursive terms are always monotonic.*

- This means that tail recursive equations always have solutions.

- Another explanation for Pete & J’s result that any tail recursive equation is satisfiable by some ACL2 function.
Such *tail recursive terms are always monotonic*:

Let $f$ and $g$ be functions such that 
($\leq$ (f x)(g x)), [i.e., $f \sqsubseteq g$].

**Case 1.** (test x) is **not** NIL.

$\tau[f](x) = (\text{base } x) = \tau[g](x)$.

So $\tau[f] \sqsubseteq \tau[g]$.

**Case 2.** (test x) is NIL

Since $\forall y[(f y) \sqsubseteq (g y)]$,

$\tau[f](x) = (f (\text{st } x))$

$\sqsubseteq (g (\text{st } x))$

$= \tau[g](x)$.

Thus $\tau[f] \sqsubseteq \tau[g]$. 

14-a
Which ACL2 terms are monotonic?

**Primitive Recursion.** Let test, base, and st be arbitrary unary functions.

Let \( h \) be a binary function.

Consider a term \( \tau[F] \) of the form

\[
(if \ (test \ x) \\
(base \ x) \\
(h \ x \ (F \ (st \ x))))
\]

Often such terms are **not** monotonic.

Such terms are **are** monotonic

if \( h \) *always preserves* \( \sqsubseteq \) in its second input:

\[
y_1 \sqsubseteq y_2 \Rightarrow (h \ x \ y_1) \sqsubseteq (h \ x \ y_2)
\]
Such primitive recursive terms are monotonic if $h$ always preserves $\sqsubseteq$ in its second input:

Let $f$ and $g$ be functions such that $(\leq (f\ x)\ (g\ x))$, [i.e., $f \sqsubseteq g$].

**Case 1.** (test $x$) is not $\text{NIL}$.  
$\tau[f](x) = (\text{base } x) = \tau[g](x)$.  
So $\tau[f] \sqsubseteq \tau[g]$.

**Case 2.** (test $x$) is $\text{NIL}$  
Since $\forall y[(f\ y) \sqsubseteq (g\ y)]$,  
$(f\ (\text{st } x)) \sqsubseteq (g\ (\text{st } x))$.  
Since $h$ always preserves $\sqsubseteq$ in its second input,  
$\tau[f](x) = (h\ x\ (f\ (\text{st } x)))$  
$\sqsubseteq (h\ x\ (g\ (\text{st } x)))$  
$= \tau[g](x)$.  
Thus $\tau[f] \sqsubseteq \tau[g]$.  

15-a
Such primitive recursive terms are monotonic if \( h \) always preserves \( \sqsubseteq \) in its second input:

\[
y_1 \sqsubseteq y_2 \Rightarrow (h \times y_1) \sqsubseteq (h \times y_2)
\]

From *Consistently Adding Primitive Recursive Definitions in ACL2*,

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

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

\[(h \times c) = c.\]

Restate in the terminology of flat domains:

A sufficient (but not necessary) condition on \( h \) for a primitive recursive term, \( \tau[F] \), to be monotonic is that \( h \) have a right fixed point.
Use: Such primitive recursive terms are monotonic

    if \( h \) always preserves \( \sqsubseteq \) in its second input:

\[
y_1 \sqsubseteq y_2 \Rightarrow (h \times y_1) \sqsubseteq (h \times y_2)
\]

To Prove: A sufficient (but not necessary) condition on \( h \) for a primitive recursive term, \( \tau[F] \), to be monotonic is that \( h \) have a right fixed point, \( c \).

Proof. Use the right fixed point \( c \) to build a flat domain:

- Use \( c \) for \( \bot \) and

- \( \sqsubseteq_c \) for \( \sqsubseteq \) where

\[
x \sqsubseteq_c y \iff x = c \lor x = y.\]

- Then

\[
y_1 \sqsubseteq_c y_2 \Rightarrow (h \times y_1) \sqsubseteq_c (h \times y_2)
\]
Which ACL2 terms are monotonic?

**Nested Recursion.** Let `test`, `base`, and `st` be arbitrary unary functions.

Consider a term \( \tau[F] \) of the form

\[
\text{(if (test x) } \\
\text{ (base x) } \\
\text{ (F (F (st x)))})
\]

Often such terms are not monotonic.

Such terms are monotonic if \( F \) always preserves \( \sqsubseteq \):

\[
y_1 \sqsubseteq y_2 \Rightarrow (F y_1) \sqsubseteq (F y_2)
\]

That is, restrict the variable \( F \) to range only over functions that always preserve \( \sqsubseteq \).
Nested Recursion and Kleene’s Construction

Recall Kleene’s construction:

- Use the term $\tau[F]$ to recursively define a chain of functions,
  $$
  f_0(x) = \bot \\
  f_{i+1}(x) = \tau[f_i](x).
  $$

- Since $\tau[F]$ is monotonic,
  $$
  f_0 \sqsubseteq f_1 \sqsubseteq \cdots \sqsubseteq f_i \sqsubseteq \cdots
  $$

- To ensure $\tau[F]$ is monotonic, the function variable $F$ should range only over functions that always preserve $\sqsubseteq$.

- That is, each $f_i$ should always preserve $\sqsubseteq$. 
Nested Recursion and Kleene’s Construction

To ensure that each $f_i$ always preserves $\sqsubseteq$:

- Clearly, $f_0$, defined by $f_0(x) = \bot$, always preserves $\sqsubseteq$.

- **Require**: Whenever $f$ always preserves $\sqsubseteq$, then $\tau[f]$ is also a function that always preserves $\sqsubseteq$. 
Nested Recursion and Kleene’s Construction

Requirement. Whenever $f$ always preserves $\sqsubseteq$, then $\tau[f]$ is also a function that always preserves $\sqsubseteq$.

Orthodox Solution. Functions, that always preserve $\sqsubseteq$, are closed under composition.

- **Restrict** $\tau[F]$ to compositions involving $F$ and functions that always preserve $\sqsubseteq$.

- So test, base, st, and if should all be functions that always preserve $\sqsubseteq$

\[
(\text{if} \ (\text{test} \ x) \\
\ (\text{base} \ x) \\
\ (F \ (F \ (\text{st} \ x))))
\]

- **Problem.** ACL2’s if does not preserve $\sqsubseteq$. 

16-c
Nested Recursion and Kleene’s Construction

**Problem.** ACL2’s if does not preserve \(\sqsubseteq\).

- Assume \(\bot \neq \text{NIL}\).
- Then \(\bot \sqsubseteq \text{NIL}\), but

\[ (\text{if } \bot 0 1) = 0 \not\sqsubseteq 1 = (\text{if } \text{NIL} 0 1) \]

**Solution.** Replace ACL2’s if with a *sequential* version, sq-if, that always preserves \(\sqsubseteq\).

\[
\begin{align*}
\text{(sq-if } \bot \text{ b c)} &= \bot \\
\text{(sq-if } \text{NIL b c)} &= c \\
\text{(sq-if } a \text{ b c)} &= b \text{ if } a \neq \bot \land a \neq \text{NIL}
\end{align*}
\]
Nested Recursion and Kleene’s Construction

**Requirement.** Whenever $f$ always preserves $\sqsubseteq$, then $\tau[f]$ is also a function that always preserves $\sqsubseteq$.

**Non-Orthodox Solution.** Replace ACL2’s `if` with the sequential version, `sq-if`, and make sure `test` is **strict**.

- A function is *strict* iff the function returns $\bot$ whenever any of its inputs is $\bot$.
- Every strict function always preserves $\sqsubseteq$.
- The function `sq-if` is **not** strict.
Nested Recursion and Kleene’s Construction

**Non-Orthodox Solution.** When test is strict, the term

\[
(sq-if \ (test \ x) \\
(\text{base} \ x) \\
(F \ (F \ (st \ x))))
\]

always produces a strict function, whenever F is replaced by any unary function f.

Every strict function always preserves \( \sqsubseteq \).
Primitive heuristics for ensuring terms are monotonic

For subterms, \( \tau[F] \), of the form

\[
\text{(if (test } x \text{) (then } x \text{) (else } x))
\]

- If \( F \) appears in \( \text{(test } x \text{)} \), then replace \text{if} by \text{sq-if}.

- If \( F \) is nested more than one deep in any of \( \text{(test } x \text{)} \), \( \text{(then } x \text{)} \), or \( \text{(else } x \text{)} \), then replace \text{if} by \text{sq-if} and ensure that \( \text{(test } x \text{)} \) is strict.
Primitive heuristics for ensuring terms are monotonic

- If \( F \) appears in (then \( x \)) or (else \( x \)) then, other function applications appearing in (then \( x \)) or (else \( x \)),

1. need not be applications of functions that always preserve \( \sqsubseteq \), if they contain no applications of \( F \);

2. should be applications of functions that always preserve \( \sqsubseteq \), if they contain any application of \( F \).

**Example.** \( (h \ (F \ (st \ x))) \)
- \( st \) need not preserve \( \sqsubseteq \)
- \( h \) should preserve \( \sqsubseteq \)
**Zero Function.** Construct an ACL2 function $Z$ satisfying the equation

\[
\text{(equal (Z x)} \\
\text{(if (equal x 0)} \\
\text{ 0} \\
\text{(* (Z (- x 1))(Z (+ x 1)))).)}
\]

- The two recursive calls of $Z$ are contained inside the call to $\ast$.

- The heuristics suggest that $\ast$ is the only function required to preserve $\sqsubseteq$.

- Unfortunately, $\ast$ does not preserve $\sqsubseteq$ with respect to the usual ACL2 version of $\bot$, ($\text{bottom}$).
A strict version of * would require

\[(\text{equal } (* (\$bottom\$) x) (\$bottom\$))\]
\[(\text{equal } (* x (\$bottom\$)) (\$bottom\$)).\]

Fortunately, the above two equations do hold if \(\$bottom\$) is replaced by 0,

\[(\text{equal } (* 0 x) 0)\]
\[(\text{equal } (* x 0) 0).\]

Therefore, the entire construction can be carried out using 0 in place of \(\$bottom\$).

This example illustrates that any convenient ACL2 object can be used to play the role of \(\$bottom\$).
**Ackermann’s Function.** Construct an ACL2 function \( f \) satisfying

\[
\text{(equal (f x1 x2))} = \begin{cases} 
\text{(+ x2 1)} & \text{if (equal x1 0)} \\
\text{(f (- x1 1) 1) } & \text{if (equal x2 0)} \\
\text{(f (- x1 1) (f x1 (f (- x2 1))))) } & \text{otherwise}
\end{cases}
\]

The heuristics suggest it should be possible to find \( f \) that satisfies:
\[
\text{(equal (f x1 x2))}
\]
\[
\quad \text{(if (equal x1 0)}
\]
\[
\quad \quad (+ x2 1)
\]
\[
\quad \quad \text{(SQ-IF (LT-ST-EQUAL x2 0)}
\]
\[
\quad \quad \quad \text{(f (- x1 1) 1)}
\]
\[
\quad \quad \quad \text{(f (- x1 1)
\]
\[
\quad \quad \quad \quad \text{(f x1
\]
\[
\quad \quad \quad \quad \quad (- x2 1))))))).
\]

- Here SQ-IF is the monotonic sequential version of if,

- LT-ST-EQUAL is a left-strict version of equal satisfying

\[
\text{(equal (LT-ST-EQUAL 'undef$ y) 'undef$).}
\]

- Here 'undef$ is used in place of ($bottom$).
The heuristics are too primitive. No such ACL2 function was proved to exist. But, experimentation shows it is possible to define an ACL2 function $f$ satisfying

$$(\text{equal } (f \ x1 \ x2))$$

$$(\text{if } (\text{equal } x1 \ 0))$$

$$(\text{LT-ST}++ \ x2 \ 1)$$

$$(\text{sq-if } (\text{lt-st-equal } x2 \ 0))$$

$$((f \ (- \ x1 \ 1) \ 1)$$

$$((f \ (- \ x1 \ 1)$$

$$((f \ x1$$

$$((- \ x2 \ 1))))))$).

- Here LT-ST++ is a left-strict version of (binary) $+$ satisfying

$$(\text{equal } (\text{LT-ST}++ \ '\text{undef}$ y) '\text{undef$)}).$$
Of course any function $f$ satisfying this last equation may not satisfy the original equation. However, ACL2 can verify the following, showing that any such $f$ can fail to satisfy the original equation only when the second input is `undef$:

\[
(\text{implies} \ (\not \ (\text{equal} \ x2 \ \text{'undef$})) \ \\
(\text{equal} \ (f \ x1 \ x2) \ \\
(\text{if} \ (\text{equal} \ x1 \ 0) \ \\
\quad (+ \ x2 \ 1) \ \\
\quad (\text{if} \ (\text{equal} \ x2 \ 0) \ \\
\quad \quad (f \ (- \ x1 \ 1) \ 1) \ \\
\quad \quad (f \ (- \ x1 \ 1) \ \\
\quad \quad \quad (f \ x1 \ \\
\quad \quad \quad \quad (- \ x2 \ 1))))) \).
\]