On enumeration of monadic predicates and n-ary relations

Harsh Raju Chamarthi
Northeastern University

November 30, 2012
Problem

Goal
Find counterexamples to a given formula in ACL2.
\((\text{and } hyp_1 \cdots hyp_n) \rightarrow \text{concl}\)
Problem

Goal
Find counterexamples to a given formula in ACL2.
\[(\text{and } hyp_1 \cdots hyp_n) \rightarrow concl\]
Given a set of \(hyp_i\) enumerate all satisfying assignments
Problem

Goal
Find counterexamples to a given formula in ACL2.
\[(\text{and } \text{hyp}_1\cdots\text{hyp}_n) \rightarrow \text{concl}\]
Given a set of \(\text{hyp}_i\) enumerate all satisfying assignments
Background - Type sets

- 14 Primitive types
- Boolean combinations
- Represented as Bit strings
- Limited Expressibility
Background - Defdata framework

Defdata adds

- Product types
  - Constructors: cons, /, complex
- Inductive types
Background - Defdata framework

Defdata adds

- Product types
  - Constructors: cons, /, complex

- Inductive types

foo is a defdata type iff

1. predicate foop is defined and
2. either enumerator nth-foo or *foo-values* is defined

Examples

Product type -- (defdata bar (cons (/ 1 pos) nat-list))
Inductive type -- (defdata loi (oneof nil (cons integer loi)))
Defdata adds

- Product types
  - Constructors: cons, /, complex
- Inductive types

- \textit{NOT}, \textit{AND} combinations not supported
- Better, but still limited expressibility

\textbf{Definition}

\textit{enum expression} gives an enumerating characterization of a variable. \textit{enum set} is a disjunction of enumerator expressions, whose meaning is the union of the respective type domains characterized by the \textit{enum} expressions.
Generative/Inductive Types (Different representation)

As much as possible express each type as an Inductive type with base elements and a finite set of generators.

\[\text{posp} : \text{Base} = \{1\} \quad \text{Gen} = \{S\}\]
\[\text{evenp} : \text{Base} = \{0\} \quad \text{Gen} = \{S \circ S\}\]
\[\text{/3p} : \text{Base} = \{0\} \quad \text{Gen} = \{S \circ S \circ S\}\]
\[\text{string-listp} : \text{Base} = \{\text{nil}\} \quad \text{Gen} = \{\lambda x. (\text{cons} a x) \mid (\text{stringp} a)\}\]

- A type \(P : [\text{Base} : a, \text{Gen} : f]\) can be enumerated by listing members in a manner reminiscent of Herbrand Universe i.e.
  \[
  \{a, fa, ffa, ffffa, fffffa \ldots\}
  \]
- Clearly an enumerator for \(P\) can be easily derived:
  \[
  (\text{nth-P n}) = \text{if} \ (\text{zp} n) \ a \ (f \ (\text{nth-P}(1-n)))
  \]
Generative Types (continued ...)

The \([Base, Gen]\) representation helps in deriving \textit{AND} combinations.

\[
evenp \land /3p \equiv Base = \{0\} \quad Gen = S \circ S \circ S \circ S \circ S
\]

Heuristic - Take intersection of bases and the LCM of the generators.

\[\]
\textit{NOT} still not so amenable.

Source predicate - push the negation all the way inside i.e.

\[
\text{(\texttt{~str-listp } x)} = \text{if (endp } x) \text{ (not (equal } x \text{ nil))} \text{ (or (not (strp (car } x)))} \text{ (\texttt{~str-listp} (cdr } x)))
\]

\[
Base = ATOM \setminus \{\text{nil}\} \cup (\text{cons } a L) \setminus (\text{strp } a) \\
Gen = (\text{cons } a) \setminus (\text{strp } a)
\]
Monadic Recursive Predicates

- Foregoing language for "types" still not expressive enough.
  e.g. orderedp, no-duplicatesp

  (no-duplicatesp X) =
  if (endp X)
    T
  (and (not (in (car X) (cdr X)))
   (no-duplicatesp (cdr X)))

- Dependent Recursion

  \[ no\text{-}duplicatesp : \text{Base} = \{ \text{nil} \}, \text{Gen} = \lambda x. (\text{cons} a x) \mid a \notin x \]

  To characterize no-duplicatesp, need to know a enumerating characterization of n-ary relations!!
Binary Relations

(in a X)

- Find all < a, X > pairs that satisfy (in a X) ≠ nil
- Given X, find all a that satisfy (in a X) ≠ nil
- Given a, find all X that satisfy (in a X) ≠ nil
Binary Relations

(in a X)

\[ a \mid_{a \in X} \]

Natively supported.

\[ X \mid_{a \in X} \]

Use a FIXing Rule to obtain an enum expression!

\[ X = (\text{insert} \ a \ X') \]
Binary Relations

\[(\text{in } a \ X)\]

\[a \ |_{a \in X}\]

Natively supported.

\[X \ |_{a \in X}\]

Use a FIXing Rule to obtain an enum expression!

\[x = (\text{insert } a \ X')\]

\[x \ |_{x < y}\]

\[y \ |_{x < y}\]
Binary Relations

\((\text{in } a \ X)\)

\(a \mid_{a \in X}\)

Natively supported.

\(X \mid_{a \in X}\)

Use a FIXing Rule to obtain an enum expression!

\(X = (\text{insert } a \ X')\)

\(x \mid_{x<y}\)

Use \(x = (y - z) \mid_{z>0}\)

\(y \mid_{x<y}\)

Use \(y = (x + z) \mid_{z>0}\)
Binary Relations

\[(\text{in } a \ X)\]
\[\text{Natively supported.}\]

Use a FIXing Rule to obtain an enum expression!
\[X = (\text{insert } a \ X')\]

Fix Rules
Like \textit{Elim} rules. (defthm in-fix2 (in a (insert a X))) Eliminate a relation in favor of enum expressions
e.g. \(= f(\ldots)\) or inf(\ldots).
Enumerating Relations

\[ \langle X, Y, Z \rangle \frac{\langle y, z, z \rangle}{R(X, Y, Z)} = X \frac{\text{es: } D}{\text{such that } x \in R^{-1}(y, z)} \text{ or } x = R^{-1}(y, z, n) \]

\[ = X \frac{y = R^{-1}(y, z, n)}{T} \cdot \frac{\langle y, z \rangle}{Q(Y, Z)} \]

\[ = \text{Staged Enumeration!} \]

\[ = \text{Monadic} \]

\[ \frac{\text{es: } D}{P(Z)} \]
On deriving $R^{-1}$

\[
\text{(subsetp } X \text{ } Y) = (\text{if (endp } X) \text{ T)}
\]

\[
\text{(and (in (car } X \text{ } Y) \text{)} (\text{subsetp (cdr } X \text{ } Y)))}
\]

\[
\text{(subsetp}^{-2} \text{ n } X) = (\text{if (endp } X) \text{ (nth-all n) (insert (car } X) \text{)} (\text{subsetp}^{-2} \text{ n (cdr } X)))}
\]

\[
\text{(subsetp}^{-1} \text{ n } Y) = (\text{if (zp n) nil)} (\text{cons (nth* n1 Y) (subsetp}^{-1} \text{ p n2 Y)})
\]

Probably doable, but more elegant to let user specify ELIM rules

\[
\text{elim for } X : \text{ } X = Y - Z
\]

\[
\text{elim for } Y : \text{ } Y = X \cup Z
\]
Monadic Predicates (continued...)

\[
\text{def} f = x \quad \text{tep} \quad \text{blp} \\
\begin{array}{c|c|c}
\text{or} (\text{endp } x) & \text{consp } x & \text{orderedp } (\text{cdr } x) \\
\text{endp } (\text{cdr } x) & \text{consp } (\text{cdr } x) & \leq (\text{car } x) (\text{cdr } x) \\
\end{array}
\]

\[
\text{or} \equiv x \\
\begin{array}{c|c|c|c}
\text{tep} \text{ nil} & \text{tep} & \text{tep} & \\
\text{endp } x & \text{cons} x & \text{orderedp } (\text{cdr } x) & \\
\text{T} & \text{endp } (\text{cdr } x) & & \\
\end{array}
\]

\[
\text{and } x \equiv x \\
\begin{array}{c|c|c|c}
\text{nil} & (\text{cons } a \text{ nil}) & (\text{cons } a (\text{cons b } x3)) & \\
\text{T} & \text{T} & a \leq b & \text{orderedp } (\text{cons b } x3) \\
\end{array}
\]

\[
\text{Orderedp} : \text{ Base } = \{ \text{n}il, [a] \} \quad \text{Gen} = \lambda y. (\text{cons } a (\text{cons } b y)) |_{a \leq b}
\]
Monadic Predicates (continued...)

\[
\begin{align*}
x & \quad \text{tlp} \quad \text{tlp} \\
\{ \text{defy} \} & \equiv x \quad \text{tlp} \\
& \quad \text{tlp} \\
& \quad \text{tlp} \\
& \quad \text{tlp} \\
\text{ordy} & \equiv x \\
& \quad \text{tlp} \\
& \quad \text{tlp} \\
& \quad \text{tlp} \\
\text{ordz} & \equiv x \\
& \quad \text{tlp} \\
& \quad \text{tlp} \\
& \quad \text{tlp} \\
\end{align*}
\]

\[
\text{orderedp } x = \text{if (or (endp x) (endp (cdr x)))} \\
\quad \text{T} \\
\quad \text{(and (<= (car x) (cadr x)) (orderedp (cadr x)))}
\]

Fixing Rule
\[
\text{orderedp (sort x)}
\]

Orderedp:
Base = \{nil, [a]\}
Gen = \lambda y. (\text{cons a (cons b y)}) |_{a \leq b}
Monadic Pred (AND)

\[ X \quad \frac{\text{str-listp } X}{\text{no-dup } X} \quad \frac{\text{orderedp } X}{\text{Apply Fix rules}} \]

or, Thread the 3 functions i.e take LCM

\[ \equiv X \quad \frac{= \text{nil}}{= \text{cons a nil}} \quad \frac{= \text{cons a (cons b x3)}}{\text{base} \quad \text{base}} \]

\[ T \quad (\text{strp a}) \quad \begin{cases} a \leq b \\ a \neq \text{cons b x3} \\ b \notin x3 \\ \text{strp a, b} \end{cases} \]

\[ \text{stlp no-dup no-ordp x3} \quad \text{recursive} \]

\[ \text{stlp no-dup no-ordp} \quad \text{Base} = \{ \text{nil}, (\text{cons a nil}) \mid (\text{strp a}) \} \]

Conjunction predicate

\[ \text{Gen} = \lambda y. (\text{cons a} \quad (\text{cons b y})) \mid \text{strp a, b} \quad a < b \quad b \notin y \]
Monadic Predicates (more complex)

Ques: Can all monadic predicates be represented in [Base, Gen] form?
Consider squarep and primep

(squarep x) = (sq1 x x)
(sq1 b x) = if (zp b) nil
                      if b*b = x T
                      (sq1 b-1 x)

Base = ? Gen = ??
(nth-square n) = (* n n)
Fix Rule
(posp x) => (squarep (* x x))

(primep x) = (nd X) = 2
            = (Pr1 x x-1)
(Pr1 x y) = if y = 1 T
                      (and (not (div x y))
                        (pr! x y-1))

Fix Rule ??

(nth-prime n) = ...
Fix Rule ??
Monadic Predicates (more complex)

Ques: Can all monadic predicates be represented in \([Base, Gen]\) form?

Consider \texttt{squarep} and \texttt{primep}

\[
\begin{align*}
\text{(squarep } x) &= (\text{sq1 } x \ x) \\
\text{(sq1 } b \ x) &= \text{if } (\text{zp } b) \\
& \quad \text{nil} \\
& \quad \text{if } b*b = x \\
& \quad T \\
& \quad (\text{sq1 } b-1 \ x)
\end{align*}
\]

\[
\begin{align*}
\text{(primep } x) &= (\text{nd } X) = 2 \\
& = (\text{Pr1 } x \ x-1)
\end{align*}
\]

\[
\begin{align*}
\text{(Pr1 } x \ y) &= \text{if } y = 1 \\
& \quad T \\
& \quad (\text{and } (\text{not } (\text{div } x \ y)) \\
& \quad (\text{pr! } x \ y-1))
\end{align*}
\]

Base = ? Gen = ??

\[
\begin{align*}
\text{(nth-square } n) &= (* \ n \ n)
\end{align*}
\]

Fix Rule

\[
\begin{align*}
\text{(posp } x) \Rightarrow (\text{squarep } (* \ x \ x))
\end{align*}
\]

\[
\begin{align*}
\text{(nth-prime } n) &= \ldots \\
\text{Fix Rule ??}
\end{align*}
\]

\[
R(x, f(x), g(x)) \text{ is a problem to enumerate } ...
\]
Equations and Inverses

From

\[ X \mid g(x) = y \]

we would like to derive the es: \( g^{-1}(y) \)

From \((\text{append } X \ Y) = Z\) we would like to derive es: \((\text{difference } Z \ Y)\big|_{Y \subset Z}\)
Mechanizable?

\[ L = (\text{zip } l1 \ l2) = (\text{if } (\text{or } (\text{endp } l1) (\text{end p } l2)) nil (\text{cons } (\text{cons } (\text{car } l1) (\text{car } l2)) (\text{zip } \text{cdr } l1 \ (\text{cdr } l2)))) \]

\[ (l1, l2) = (\text{unzip } L) = (\text{if } (\text{endp } L) (\text{mv nil nil}) (\text{mv-let } (l1 \ l2) (\text{unzip } (\text{cdr } L)) (\text{b* } ((\text{cons } a \ b) \ (\text{car } L)) (\text{mv } (\text{cons } a1 \ l1) \ (\text{cons } b \ l2)))))) \]
Mechanizable?

L = (zip l1 l2) = (if (or (endp l1) (endp l2))
    nil
    (cons (cons (car l1) (car l2))
      (zip cdr l1) (cdr l2)))

(l1, l2) = (unzip L) = (if (endp L)
    (mv nil nil)
    (mv-let (l1 l2)
      (unzip (cdr L))
      (b* ((cons a b) (car L))
        (mv (cons a1 l1) (cons b l2))))))

Inverse/Elim Rule for zip
(zip (strip-cars L) (strip-cdrs L)) = L
A ternary relation

(shufflep x y z) =
(if (endpz)
  x = y = z = nil
(if (endp x)
  y = z
(if (endp y)
  x = z
(or
  (and (car x) = (car z)
    (shufflep x' y z'))
  (and ((car y) = (car z)
    (shufflep x y' z')))))

Ques: Under what circumstances can this derivation be mechanized?
Interesting example...

(adj-listp G) =
(and (symbol-alistp G)
 (adj-listlp G (strip-cars G))

(adj-listlp G dom) =
(if (end G)
 T
 (and (consp (car G))
 (subsetp (cdar G) dom)
 (adj-list) P (cdr G) dom)

Method 1: Thread and derive

\[ Base = \{nil\} \quad Gen = \lambda g.(c (c a b) g) \big|_{b \subseteq dom} \]

Method 2: Staged enumeration. Apply Fix Rules...

Method 3: Rewrite \( G = (zip \ dom \ edges-list) \). Derive \( dom \) is symbol-listp. Then derive:

\[ (R \ el \ dom) = (if \ (endp \ el)
 T
 (and (subsetp (car el) dom)
 (R (cdr el) dom)) \]
Negation and Conjunction of Monadic Predicates

(no-duplicatesp X) => (orderep X)

Counterexamples: Enumerate (and (no-dup X) (not (ordered X)))

(no-dup X) =
(if (endp X) T
 (if (endp (cdr X)) T
 (if (> (car X) (cadr X))
 (and (not (in (car X) (cdr X)))
 (no-dup (cdr X)))
 (and (not (in (car X) (cdr X)))
 (no-dup X'))))))

Match the IF structure and merge the two predicates to get:

Negate!

~(orderedp X) =
(if (endp X) nil
 (if (endp (cdr X)) nil
 (if (car X) > (cadr X) T
 (~orderedp (cdr X))))))

29 / 1
Negation and Conjunction of Monadic Predicates

(no-duplicatesp X) => (orderep X)

Counterexamples: Enumerate (and (no-dup X) (not (ordered X)))

(no-dup X) =
(if (endp X)
 T
(if (endp (cdr X))
 T
 (if (> (car X) (cadr X))
 (and (not (in (car X) X'))
 (no-dup X'))
 (and (not (in (car X) X'))
 (no-dup X'))))))

Negate!
~(orderedp X) =
(if (endp X)
 nil
 (if (endp (cdr X))
 nil
 (if (car X) > (cadr X)
 T
 (~orderedep X'))) )))

Match the IF structure and merge the two predicates to get:

(|no-dup & ~orderedep| X) =
(if (endp X)
 (and T nil)
 (if (endp (cdr X))
 (and T nil)
 (if (> (car X) (cadr X))
 (and (not (in (car X) (cadr X)))
 (no-dup (cadr X)))
 (and (not (in (car X) (cadr X)))
 (|no-dup & ~orderedep| (cadr X)))))))
Recap

- Generative types
  - Base, Gen representation
  - AND
  - NOT

- Richer Types
  - Monadic Predicates build on n-ary relations
  - Instances of relations $R(x, x)$ are hard...

- Need Elim/Fix/Inverse Rules from the user to program the Cgen capability

- Staged Enumeration (Dependency graph dictated by Rules above)
Finally...

- Find a corresponding "The Method" for CGen framework.
- Interactive Non-Theorem Disproving
  same philosophy as ACL2, more integration with ACL2.
- Fundamental Questions.

Applications

- Lemma generation
- Internal Heuristics (Generalize, Induction)
- Counterexample generation
Finally...

- Find a corresponding "The Method" for CGen framework.
- Interactive Non-Theorem Disproving
  - same philosophy as ACL2, more integration with ACL2.
- Fundamental Questions.

Applications

- Lemma generation
- Internal Heuristics (Generalize, Induction)
- Counterexample generation

Thank You!