```
;; The first four lines of this file were added by Dracula.
;; They tell DrScheme that this is a Dracula Modular ACL2 program.
;; Leave these lines unchanged so that DrScheme can properly load this file.
#reader(planet "reader.ss" ("cce" "dracula.plt") "modular" "lang")
(require "../interfaces/Ibasiclex.lisp")

(module Mbasiclex
  (include-book "list-utilities" :dir :teachpacks)

  ; ds = delimiters to look for (list)
  ; xs = object to search in    (list)
  ; (split-at-delimiter ds xs) = (before at+)
  ;   where before = longest prefix of xs with no values from ds (list)
  ;         at+    = rest of xs (list)
  (defun split-at-delimiter (ds xs)
    (if (or (endp xs) (member-equal (car xs) ds))
        (list nil xs)
        (let* ((cdr-split (split-at-delimiter ds (cdr xs))))
          (list (cons (car xs) (car cdr-split))
                (cadr cdr-split)))))

  ; ps = list of signals to pass by (no constraints on signals)
  ; xs = list of signals (no constraints on signals)
  ; (span ps xs) = list of two elements
  ;  1 longest prefix of xs containing only signals from ps
  ;  2 rest of xs
  (defun span (ps xs)
    (if (or (endp xs) (not (member-equal (car xs) ps)))
        (list nil xs)
        (let* ((cdr-span (span ps (cdr xs))))
          (list (cons (car xs) (car cdr-span))
                (cadr cdr-span)))))

  ; ps = prefix to look for  (list)
  ; xs = object to search in (list)
  ; (splitoff-prefix ps xs) = (ps-matching ps-af-match xs-af-match)
  ;  where ps-matching = longest ps prefix matching xs prefix (list)
  ;        ps-af-match = rest of ps                           (list)
  ;        xs-af-match = non-matching suffix of xs            (list)
  ; Note: ps-af-match = nil means ps is a prefix of xs
  (defun splitoff-prefix (ps xs)
    (if (and (consp ps)
             (consp xs)
             (equal (car ps) (car xs)))
        (let* ((3way (splitoff-prefix (cdr ps) (cdr xs)))
               (ps-matching (car   3way))
               (ps-af-match (cadr  3way))
               (xs-af-match (caddr 3way)))
          (list (cons (car ps) ps-matching) ps-af-match xs-af-match))
        (list nil ps xs)))

  ; ps = prefix to look for  (list of standard, upper-case characters)
  ; xs = object to search in (list)
  ; (splitoff-prefix-upr ps xs) = (ps-matching ps-af-match xs-af-match)
  ;  where ps-matching = longest ps prefix matching xs prefix (list)
  ;        ps-af-match = rest of ps                           (list)
  ;        xs-af-match = non-matching suffix of xs            (list)
  ; Notes: 1. search is not sensitive to case of letters in xs
  ;        2. ps-af-match = nil means ps is a prefix of xs
  ; Implementation issue: combining general and case-insensitive search
  ;   in one function simplifies maintenance, but complicates
  ;   formulation of software properties and their proofs
  (defun splitoff-prefix-upr (ps xs)
    (if (and (consp ps)
             (consp xs)
             (equal (car ps)
                    (let* ((x (car xs)))
```

```
                    (if (standard-char-p x)
                        (char-upcase x)
                        x))))
      (let* ((3way (splitoff-prefix-upr (cdr ps) (cdr xs)))
             (ps-matching (car   3way))
             (ps-af-match (cadr  3way))
             (xs-af-match (caddr 3way)))
        (list (cons (car ps) ps-matching) ps-af-match xs-af-match))
      (list nil ps xs)))

; tok-str = characters to look for (string, standard characters)
; chrs    = object to search in
(defun splitoff-prefix-chr (tok-str xs)
  (splitoff-prefix-upr (str->chrs(string-upcase tok-str)) xs))

; tok = object to search for (list)
; xs  = object of search     (list)
; (split-on-token-gen tok xs) = (prefix match suffix)
;   where
;   prefix = elems of xs before 1st sublist matching tok (list)
;          = xs  if no match
;   match  = tok if match                                (list)
;          = nil if no match
;   suffix = elems of xs after 1st sublist matching tok  (list)
;          = nil if no match
(defun split-on-token-gen (tok xs)
  (if (endp xs)
      (list nil nil nil)
      (let* ((splitoff-3way (splitoff-prefix tok xs))
             (matching?     (null (cadr splitoff-3way)))
             (aftr-tok      (caddr splitoff-3way)))
        (if matching?
            (list nil tok aftr-tok)
            (let* ((3way (split-on-token-gen tok (cdr xs)))
                   (bfor-tok (car   3way))
                   (at-tok   (cadr  3way))
                   (aftr-tok (caddr 3way)))
              (list (cons (car xs) bfor-tok)
                    at-tok
                    aftr-tok))))))

; tok = object to search for (list of upper-case standard characters)
; xs  = object to search in  (list containing no non-standard chars)
; Note: matching is not case-sensitive
; (split-on-token-chr tok xs) = (prefix match suffix)
;   where
;   prefix = elems of xs before 1st sublist matching tok (list)
;          = xs  if no match
;   match  = tok if match                                (list)
;          = nil if no match
;   suffix = elems of xs after 1st sublist matching tok  (list)
;          = nil if no match
(defun split-on-token-chr (tok xs)
  (if (endp xs)
      (list nil nil nil)
      (let* ((splitoff-3way (splitoff-prefix-upr tok xs))
             (matching?     (null (cadr splitoff-3way)))
             (aftr-tok      (caddr splitoff-3way)))
        (if matching?
            (list nil tok aftr-tok)
            (let* ((3way (split-on-token-chr tok (cdr xs)))
                   (bfor-tok (car   3way))
                   (at-tok   (cadr  3way))
                   (aftr-tok (caddr 3way)))
              (list (cons (car xs) bfor-tok)
                    at-tok
                    aftr-tok))))))
```

```
; tok = object to search for (string or list)
; xs  = object to search in (list, no non-standard chars if tok is string)
; Note: search is not case-sensitive if tok is a string
; Warning! Neither tok nor xs may contain non-standard characters
;          if tok is a string
; Implementation issue: combining general and case-insensitive search
;   in one function simplifies maintenance, but complicates
;   formulation of software properties and their proofs
(defun split-on-token (tok xs)
  (if (stringp tok)
      (split-on-token-chr (str->chrs(string-upcase tok)) xs)
      (split-on-token-gen tok xs)))

(export Ibasiclex))
```