Branch: master **boogie-friends** / emacs / **dafny-mode.el**

cpitclaudel 11 days ago dafny: fix attribute snippets and add utility function

1 contributor

515 lines (436 sloc) 23.665 kB Raw Blame History

```
;;; dafny-mode.el --- Support for the Dafny programming language -*- lexical-binding: t -*-

;; Copyright (C) 2015 Clément Pit--Claudel
;; Author: Clément Pit--Claudel <clement.pitclaudel@live.com>
;; URL: https://github.com/boogie-org/boogie-friends/

;; Keywords: convenience, languages

;; This file is not part of GNU Emacs.

;; Permission is hereby granted, free of charge, to any person obtaining a copy
;; of this software and associated documentation files (the "Software"), to deal
;; in the Software without restriction, including without limitation the rights
;; to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
;; copies of the Software, and to permit persons to whom the Software is
;; furnished to do so, subject to the following conditions:

;; The above copyright notice and this permission notice shall be included in
;; all copies or substantial portions of the Software.

;; THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
;; IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
;; FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
;; AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
;; LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
;; OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
;; SOFTWARE.

;;; Commentary:

;; See boogie-friends.el

;;; Code:

;; This file contains the implementation of the Dafny part of the boogie-friends
;; package

(require 'boogie-friends)
(require 'boogie-mode)
(require 'dafny-docs)
(require 'inferior-dafny)

(defconst dafny-defuns '("class" "codatatype" "colemma" "constructor" "copredicate" "datatype" "function"
                         "iterator" "lemma" "method" "newtype" "predicate" "trait" "type"
                         "function method" "predicate method"))

(defconst dafny-specifiers '("decreases" "ensures" "free" "invariant" "modifies" "reads" "requires"))

(defconst dafny-modifiers '("inductive" "abstract" "ghost" "protected" "static"))

(defconst dafny-builtins '("as" "default" "extends" "import" "include" "module" "opened" "refines" "returns" "yields"))

(defconst dafny-keywords '("assert" "assume" "break" "calc" "case" "else" "exists" "false" "forall" "fresh" "if"
                           "in" "label" "match" "modify" "new" "null" "old" "print" "return" "then" "this"
                           "true" "var" "where" "while" "yield"))

(defconst dafny-types '("array" "array2" "array3" "bool" "char" "imap" "int" "iset" "map" "multiset" "nat" "object"
                        "real" "seq" "set" "string"))

(defconst dafny-block-heads '("calc" "else" "if" "match" "while"))

(defconst dafny-all-keywords (cl-loop for source in '(dafny-defuns dafny-specifiers dafny-modifiers
                                                      dafny-builtins dafny-keywords dafny-types)
                                      append (mapcar (lambda (kwd) (propertize kwd 'source source)) (symbol-value source))))
```

```
64
65
66  (defconst dafny-defuns-regexp     (regexp-opt dafny-defuns 'symbols))
67  (defconst dafny-specifiers-regexp (regexp-opt dafny-specifiers 'symbols))
68  (defconst dafny-modifiers-regexp  (regexp-opt dafny-modifiers 'symbols))
69  (defconst dafny-builtins-regexp   (regexp-opt dafny-builtins 'symbols))
70  (defconst dafny-keywords-regexp   (regexp-opt dafny-keywords 'symbols))
71  (defconst dafny-types-regexp      (regexp-opt dafny-types 'symbols))
72
73  (defconst dafny-extended-defun-regexp (concat "\\s-*\\(" dafny-modifiers-regexp "\\)*\\s-*" dafny-defuns-regexp))
74
75  (defconst dafny-extended-block-head-regexp (concat "\\s-*\\(" dafny-modifiers-regexp "\\)*"
76                                                     "\\s-*"    (regexp-opt (append dafny-block-heads dafny-defuns) 'symbols)))
77
78  (defgroup dafny nil
79    "IDE extensions for the Dafny programming language."
80    :group 'boogie-friends)
81
82  (defcustom dafny-snippets-repo "etc/dafny-snippets"
83    "Name of file holding Dafny snippets."
84    :group 'dafny)
85
86  (defcustom dafny-attributes-repo "etc/dafny-attributes"
87    "Name of file holding Dafny attributes."
88    :group 'dafny)
89
90  (defcustom dafny-verification-backend 'cli
91    "Which interface to use for on-the-fly verification.
92
93  One of `cli', `server', or nil.
94
95  * `cli' uses the standard Dafny binary (found at
96    `flycheck-dafny-executable'), which does not suport caching.
97
98  * `server' uses a background Dafny server process (found at
99    `flycheck-inferior-dafny-executable') and supports caching.
100
101 * nil disables on-the-fly verification."
102   :tag "Prover interface"
103   :type '(radio (const :tag "CLI (slow, stable)" cli)
104                 (const :tag "Server (fast, experimental)" server)
105                 (const :tag "None" nil))
106   :group 'dafny)
107
108 (defcustom dafny-prover-args '("/compile:0" "/nologo")
109   "Arguments to pass to Dafny when checking a file.
110 The name of the file itself is added last.  You can override all
111 arguments here, or use `dafny-prover-custom-args' to add just a
112 few extra flags in addition to the default ones."
113   :group 'dafny)
114
115 (defcustom dafny-prover-custom-args '()
116   "Extra arguments to pass to Dafny.
117 These come in addition to `dafny-prover-args'."
118   :group 'dafny)
119
120 (defcustom dafny-prover-background-args '("/timeLimit:20")
121   "Extra arguments to pass to Dafny for background verification.
122 These come in addition to `dafny-prover-args' and
123 `dafny-prover-custom-args'."
124   :group 'dafny)
125
126 (defcustom dafny-prover-local-args '()
127   "Extra arguments to pass to Dafny when checking a file.
128 These come in addition to `dafny-prover-args' and
129 `dafny-prover-custom-args'."
130   :group 'dafny)
131
132 (defcustom dafny-prover-alternate-args '("/compile:3")
133   "Extra arguments to pass to Dafny when compiling with a prefix arg.
134 Added to `dafny-prover-basic-args', `dafny-prover-local-args',
135 and `dafny-prover-custom-args' when manually launching
136 verification (\\[boogie-friends-verify]) with a prefix arg."
137   :group 'dafny)
138
139 (defvar dafny-snippets nil
140   "Cache of all known Dafny snippets, loaded from `dafny-snippets-repo'.")
141
142 (defvar dafny-attributes nil
143   "Cache of all known Dafny snippets, loaded from `dafny-snippets-repo'.")
```

```
144
145 (defun dafny-load-snippets-collection (repo place force-reload interactive)
146   "Load snippets from REPO into PLACE.
147 Loading happens only is PLACE's value is nil, or if FORCE-RELOAD
148 in non-nil."
149   (unless (and (not force-reload) (symbol-value place))
150     (let* ((docs-fname (expand-file-name repo boogie-friends-directory))
151            (snippets (with-temp-buffer (insert-file-contents docs-fname) (buffer-string))))
152       (set place (cl-loop for index = 0 then (1+ index)
153                           for line in (split-string snippets "\n\n" t)
154                           for trimmed = (replace-regexp-in-string "\\(\\`\n+\\|\n+\\'\\)" "" line)
155                           for cleaned = (boogie-friends-cleanup-snippet trimmed)
156                           collect (propertize cleaned 'index index 'snippet trimmed)))))
157   (if interactive
158       (message "Loaded %d snippets" (length (symbol-value place)))
159     (symbol-value place)))
160
161 (defun dafny-init-snippets (&optional force-reload interactive)
162   "Initialize and return `dafny-snippets'.
163 Reloading only happens if `dafny-snippets' is nil or if
164 FORCE-RELOAD is non-nil.  A non-nil INTERACTIVE value suppresses
165 the return value."
166   (interactive '(t t))
167   (dafny-load-snippets-collection dafny-snippets-repo 'dafny-snippets force-reload interactive))
168
169 (defun dafny-init-attributes (&optional force-reload interactive)
170   "Initialize and return `dafny-attributes'.
171 Reloading only happens if `dafny-attributes' is nil or if
172 FORCE-RELOAD is non-nil.  A non-nil INTERACTIVE value suppresses
173 the return value."
174   (interactive '(t t))
175   (dafny-load-snippets-collection dafny-attributes-repo 'dafny-attributes force-reload interactive))
176
177 (defconst dafny-font-lock-keywords ;; FIXME type constraints
178   (list
179    (list #'boogie-friends-mark-font-lock-assignment-chain
180          1 font-lock-variable-name-face)
181    (list (concat "\\(?:" dafny-defuns-regexp "\\s-+\\)+" boogie-friends-font-lock-var)
182          2 font-lock-function-name-face)
183    (list (concat boogie-friends-font-lock-var "\\s-*" ":" "\\s-*" boogie-friends-font-lock-type)
184          '(1 font-lock-variable-name-face) '(2 font-lock-type-face))
185    (cons dafny-defuns-regexp font-lock-builtin-face)
186    (cons dafny-modifiers-regexp font-lock-preprocessor-face)
187    (cons dafny-specifiers-regexp font-lock-doc-face)
188    (cons dafny-builtins-regexp font-lock-builtin-face)
189    (cons "!\\_<in\\_>" font-lock-keyword-face) ;; Needed because '!' is not part of a symbol, so adding '!in' to keywords doesn't
190    (cons dafny-keywords-regexp font-lock-keyword-face)
191    (cons dafny-types-regexp font-lock-type-face)
192    (list "\\(!\\)\\([^=i!]\\|$\\)" 1 font-lock-negation-char-face)
193    (list "\\(\\_<forall\\_>\\).*?::"
194          '(1 (compose-region (match-beginning 1) (match-end 1) ?∀))
195          '(1 font-lock-keyword-face append)))
196   "Font lock specifications for `dafny-mode'.")
197
198 (defun dafny-ignore-event (_e)
199   "Swallow an event E.
200 Useful to ignore mouse-up events handled mouse-down events."
201   (interactive "e"))
202
203 (defvar dafny-mode-map
204   (let ((map (boogie-friends-make-keymap t)))
205     (define-key map (kbd "C-c C-a") 'boogie-friends-translate)
206     (define-key map (kbd "C-c C-b") 'dafny-insert-attribute)
207     (define-key map (kbd "C-c C-j") 'dafny-jump-to-boogie)
208     (define-key map (kbd "C-c C-?") 'dafny-docs-open) ;; TODO enable by default?
209     (define-key map (kbd "<C-mouse-1>") 'dafny-ignore-event)
210     (define-key map (kbd "<C-S-mouse-1>") 'dafny-ignore-event)
211     (define-key map (kbd "<C-down-mouse-1>") 'dafny-click-find-definition)
212     (define-key map (kbd "<C-S-down-mouse-1>") 'dafny-click-jump-to-boogie)
213     map)
214   "Keybindings for `dafny-mode'.")
215
216 (defconst dafny-mode-syntax-table
217   (let ((tbl (make-syntax-table)))
218     (modify-syntax-entry ?'  "w" tbl)
219     (modify-syntax-entry ?_  "w" tbl)
220     ;; Comments
221     (modify-syntax-entry ?\n ">" tbl)
222     (modify-syntax-entry ?/  "  124" tbl)
223     (modify-syntax-entry ?*  "  23bn" tbl)
```

```
    tbl)
  "Syntax table for `dafny-mode'.")

(defconst dafny-translation-proc-name "*dafny-to-boogie*"
  "Name of the Dafny → Boogie process.")

(defconst dafny-translation-extension ".bpl"
  "Extension of generated Boogie files.")

(defconst dafny-translation-target-mode 'boogie-mode
  "Mode of generated Boogie files.")

(defun dafny-translation-prover-args-fn (dest-fname)
  "Extra arguments to translate to lower level source"
  (list "/nologo" "/noVerify" (concat "/print:" dest-fname)))

(defun dafny-profiler-prepare-fn (use-alternate callback)
  "Prepare a boogie source buffer before launching the profiler"
  ;; The callback is invoked in the context of the new, translated buffer
  (boogie-friends-translate
   use-alternate (lambda (_source-buffer translated-buffer)
                   (with-current-buffer translated-buffer
                     (funcall callback buffer-file-name)))))

(defun dafny-line-props ()
  "Classifies the current line (for indentation)."
  (save-excursion
    (beginning-of-line)
    (cons (cond ((or (comment-beginning) (looking-at-p "\\s-*/[/*]")) 'comment)
                ((looking-at-p "\\s-*\\(case\\|else\\)")              'case)
                ((looking-at-p ".*{\\s-*\\(//.*\\)?$")                'open)
                ((looking-at-p ".*}\\s-*\\(//.*\\)?$")                'close)
                ((looking-at-p ".*;\\s-*\\(//.*\\)?$")                'semicol)
                ((looking-at-p dafny-extended-defun-regexp)           'defun)
                (t                                                    'none))
          (current-indentation))))

(defun dafny-indent ()
  "Indent current line."
  (interactive)
  (beginning-of-line)
  (let* ((pprev-type  (car-safe (save-excursion (and (boogie-friends-backward-line) (boogie-friends-backward-line) (dafny-line-pro
         (prev-props  (save-excursion (and (boogie-friends-backward-line) (dafny-line-props))))
         (prev-type   (car-safe prev-props))
         (prev-offset (or (cdr-safe prev-props) 0))
         (is-defun    (looking-at-p dafny-extended-defun-regexp))
         (is-close    (looking-at-p "[^{\n]*}"))
         (is-lonely-open (looking-at-p "[ \t]*{"))
         (is-case    (looking-at-p "[ \t]*case"))
         (is-else    (looking-at-p "[ \t]*else"))
         (comment-beg (save-excursion (comment-beginning))))
    (indent-line-to
     (cond (comment-beg (if (< comment-beg (point-at-bol)) ;; Multiline comment; indent to '*' or beginning of text
                            (let ((incr (if (looking-at-p "\\s-*\\*") 1 3)))
                              (save-excursion (goto-char comment-beg) (+ (current-indentation) incr)))
                          prev-offset))
           ((or is-close is-lonely-open)
            (save-excursion
              (when is-close
                (backward-up-list))
              ;; Find beginning of block head (the head can span multiple lines)
              (let ((bound (save-excursion (ignore-errors (backward-up-list) (point)))))
                ;; The bound ensures that brackets headerless blocks are indented properly
                (re-search-backward (concat "^\\s-*}?" dafny-extended-block-head-regexp) bound t))
              (current-indentation)))
           (is-defun (if (memq prev-type '(open)) (indent-next-tab-stop prev-offset) prev-offset))
           (is-case (-if-let (parent (save-excursion (when (re-search-backward "^\\s-*match" nil t) (current-indentation))))
                        (indent-next-tab-stop parent)
                      prev-offset))
           (is-else (or (save-excursion (when (re-search-backward "^\\s-*if" nil t) (current-indentation)))
                        prev-offset))
           (t (pcase prev-type
                (`comment prev-offset)
                (`case    (indent-next-tab-stop prev-offset))
                (`open    (indent-next-tab-stop prev-offset))
                (`close   prev-offset)
                (`semicol prev-offset)
                (`defun   (indent-next-tab-stop prev-offset))
                (`none    (if (memq pprev-type '(none defun comment case)) prev-offset (indent-next-tab-stop prev-offset)))
                (_        prev-offset))))))
```

```
  (skip-chars-forward " "))

(defun dafny-indent-keep-position ()
  "Indent current line, minimally moving point.
That is, leaves the point in place if it is already beyond the
first non-blank character of that line, and moves it to the first
character in the line otherwise."
  (interactive)
  (let ((position (save-excursion (dafny-indent) (point))))
    (when (> position (point))
      (goto-char position))))

(defun dafny-jump-to-boogie-internal (line &optional buffer)
  "Jump to translation of LINE in boogie buffer BUFFER.
Attemps to guess the right buffer if BUFFER is nil.  If unable to
find references to LINE, look for references to neighbouring
lines."
  (-when-let* ((buffer (or buffer
                           (-when-let* ((bpl-fname (boogie-friends-translated-fname)))
                             (find-buffer-visiting bpl-fname))))
               (window (display-buffer buffer))
               ((dest . delta) (with-current-buffer buffer
                                 (let ((case-fold-search t))
                                   (save-excursion
                                     (cl-loop for delta in '(0 -1 -2 -3 -4 -5 1 2 3 4 5)
                                              do (goto-char (point-max))
                                              for pos = (search-backward (format ".dfy(%d," (+ line delta)) nil t)
                                              thereis (when pos (cons pos delta)))))))))
    (with-current-buffer buffer
      (with-selected-window window
        (goto-char dest)
        (boogie-highlight-current-line (= 0 delta))
        (recenter)))
    delta))

(defvar-local dafny-jump-overlay nil
  "Temporary highlighting of a line matching a Boogie position.
See `dafny-jump-to-boogie'.")

(defun dafny-jump-to-boogie (line &optional buffer)
  "Jump to the Boogie location matching LINE.
Interactively, LINE is the current line.  BUFFER is the Boogie
buffer to search.  Since not all lines have a direct counterpart
in the Boogie file, the line actually matched is briefly
highlighted."
  (interactive (list (save-restriction (widen) (line-number-at-pos (point))) nil))
  (boogie-friends-clean-overlay 'dafny-jump-overlay)
  (-if-let* ((delta (dafny-jump-to-boogie-internal line buffer)))
      (progn (when (/= 0 delta)
               (message "No location found for line %d. Showing the closest available reference, %d line(s) %s."
                        line (abs delta) (if (> 0 delta) "above" "below")))
             (setq dafny-jump-overlay (save-excursion (forward-line delta)
                                                      (make-overlay (point-at-bol) (point-at-eol))))
             (overlay-put dafny-jump-overlay 'face 'highlight)
             (run-with-timer 0.5 nil #'boogie-friends-clean-overlay 'dafny-jump-overlay (current-buffer)))
    (error "No location found for line %d" line)))

(defun dafny-click-find-definition (event)
  "Find definitions of symbol under mouse pointer.
Symbol at point must be a function name. Search is restricted to
open Dafny buffers."
  (interactive "e") ;; FIXME would be much better to only show the lines below the definition
  (boogie-friends-with-click event 'dafny-mode nil
    (-when-let* ((fun-name (thing-at-point 'word)))
      (occur-1 (concat "^" dafny-extended-defun-regexp "\\s-*\\<" (regexp-quote fun-name) "\\>") 3
               (cl-loop for b being the buffers when (string-match-p "\\.dfy\\'" (buffer-name b)) collect b))
      (-when-let* ((buf (get-buffer "*Occur*")))
        (with-current-buffer buf
          (face-remap-set-base 'match '(:weight bold :inverse-video t)))))))

(defun dafny-click-jump-to-boogie (event)
  "Call `dafny-jump-to-boogie' on line under mouse."
  (interactive "e")
  (boogie-friends-with-click event 'dafny-mode t
    (dafny-jump-to-boogie (line-number-at-pos (point)) nil)))

(defun dafny-snippets-doc-buffer (arg)
  "Show documentation for snippet ARG."
  (-when-let* ((doc-buffer (dafny-docs-open))
               (doc-window (get-buffer-window doc-buffer)))
```

```
    (with-current-buffer doc-buffer
      (with-selected-window doc-window
        (save-match-data
          (when (cl-loop for regexp in '("\\_<\\(\\(?:\\w\\|\\s-\\)+\\)\\_>" "\\_<\\(\\w+\\)\\_>")
                         for needle = (when (string-match regexp arg) (match-string-no-properties 0 arg))
                         when needle thereis (progn (goto-char (point-min))
                                                    (or (re-search-forward (concat "\n  +" (regexp-quote needle)) nil t)
                                                        (search-forward needle nil t))))
            (beginning-of-line)
            (recenter))))
      (current-buffer))))

(defun dafny-file-exists-or-error (fname &optional if-nil if-missing no-exists-error)
  "Return FNAME, unless it does not exist as a file."
  (if fname
      (if (or no-exists-error (file-exists-p fname))
          fname
        (error "%s" (or if-missing (format "Not found: %s" fname))))
    (error "%s" (or if-nil "No file found"))))

(defun dafny-test-suite-paths (dfy-name &optional no-err-for-expect)
  (-when-let* ((dfy    (dafny-file-exists-or-error dfy-name "No file at point"))
               (expect (dafny-file-exists-or-error (concat dfy-name ".expect") nil nil no-err-for-expect))
               (output (dafny-file-exists-or-error
                        (expand-file-name (concat (file-name-nondirectory dfy-name) ".tmp")
                                          (expand-file-name "Output" (file-name-directory dfy-name))))))
    (list dfy expect output)))

;;;###autoload
(defun dafny-test-suite-open-diff (dfy-name)
  (interactive (list (progn (require 'ffap) (with-no-warnings (ffap-file-at-point)))))
  (-when-let* (((dfy expect output) (dafny-test-suite-paths dfy-name)))
    (diff expect output)))

;;;###autoload
(defun dafny-test-suite-accept-diff (dfy-name)
  (interactive (list (progn (require 'ffap) (with-no-warnings (ffap-file-at-point)))))
  (-when-let* (((dfy expect output) (dafny-test-suite-paths dfy-name t)))
    (copy-file output expect t)
    (message "%s accepted" output)))

(defun dafny-verify-false ()
  (interactive)
  (save-excursion
    (goto-char (point-min))
    (let ((re (concat "^" dafny-extended-defun-regexp "\\>")))
      (while (re-search-forward re nil t)
        (replace-match "\\& {:verify false}" t)))))

(defun dafny-verify-true ()
  (interactive)
  (save-excursion
    (goto-char (point-min))
    (let ((re (concat "^\\(" dafny-extended-defun-regexp "\\>\\)\\s-*{:verify\\s-*\\(?:true\\|false\\)\\s-*}")))
      (while (re-search-forward re nil t)
        (replace-match "\\1")))))

(defun dafny-attribute-prefix ()
  (when (save-excursion (skip-syntax-backward "w_")
                        (looking-back (regexp-quote "{:") (- (point) 2)))
    (company-grab-symbol)))

(defun dafny-attributes-backend (command &optional arg &rest ignored)
  "A boogie-mode backend for attributes."
  (interactive (list 'interactive))
  (pcase command
    (`interactive (company-begin-backend 'dafny-attributes-backend))
    (`prefix (dafny-attribute-prefix))
    (`candidates (boogie-friends-candidates-snippet arg (dafny-init-attributes)))
    (`match (get-text-property 0 'match arg))
    (`ignore-case t)
    (`sorted t)
    (`annotation "<snip>")
    (`post-completion (boogie-friends-insert-snippet arg))
    (`no-cache t)
    (`require-match 'never)))

(defun dafny-insert-attribute ()
  (interactive)
  (insert "{:}")
  (backward-char 1)
```

```
464
465   (call-interactively #'dafny-attributes-backend))
466
467 (defun dafny-predicate ()
468   (or (eq dafny-verification-backend 'cli)
469       boogie-friends--prover-running-in-foreground-p))
470
471 (defun dafny-error-filter (errs)
472   (boogie-friends-cleanup-errors (flycheck-increment-error-columns errs)))
473
474 (flycheck-def-executable-var dafny "dafny")
475
476 (flycheck-define-command-checker 'dafny
477   "Flycheck checker for the Dafny programming language."
478   :command '("dafny" (eval (boogie-friends-compute-prover-args)) source-inplace)
479   :error-patterns boogie-friends-error-patterns
480   :error-filter #'dafny-error-filter
481   :predicate #'dafny-predicate
482   :modes '(dafny-mode))
483
484 (add-to-list 'flycheck-checkers 'dafny)
485
486 ;;;###autoload
487 (add-to-list 'auto-mode-alist '("\\.dfy\\'" . dafny-mode))
488
489 (defvar dafny--flycheck-extra nil
490   "Extra text to append to Flycheck's status.")
491
492 (defun dafny-mode-flycheck-mode-line ()
493   "Construct a string to replace the default Flycheck modeline."
494   (concat (flycheck-mode-line-status-text)
495           (or dafny--flycheck-extra "")))
496
497 ;;;###autoload
498 (define-derived-mode dafny-mode prog-mode "Dafny"
499   "Major mode for editing Dafny programs.
500
501 \\{dafny-mode-map}"
502   :syntax-table dafny-mode-syntax-table
503   (setq-local boogie-friends-symbols-alist (append '(("in" . ?∈) ("!in" . ?∉) ("!!" . ?‼))
504                                                    boogie-friends-symbols-alist))
505   (boogie-friends-mode-setup)
506   (add-to-list 'company-backends #'dafny-attributes-backend)
507   (set (make-local-variable 'flycheck-mode-line) '(:eval (dafny-mode-flycheck-mode-line)))
508   (set (make-local-variable 'indent-line-function) #'dafny-indent-keep-position)
509   (set (make-local-variable 'indent-region-function) nil)
510   (add-to-list (make-local-variable 'font-lock-extra-managed-props) 'composition)
511   (electric-indent-local-mode 1))
512
513 (provide 'dafny-mode)
514 ;;; dafny-mode.el ends here
```