SPARK Ada mode for GNU/Emacs

Gaétan Allaert

July 10, 2011

Contents

1 Introduction

This is a major mode to edit SPARK Ada source code in Emacs. This mode is based on the Ada mode already provided with Emacs.

SPARK is a formally-defined computer programming language based on the Ada programming language, intended to be secure and to support the development of high integrity software used in applications and systems where predictable and highly reliable operation is essential either for reasons of safety (e.g., avionics in aircraft/spacecraft, or medical systems and process control software in nuclear powerplants) or for business integrity (for example financial software for banking and insurance companies).

2 Installation

The SPARK Ada mode is not a stand alone mode, it requires the Ada mode.

```
ADA_MODE_VERSION=4.01
SPARK_MODE_VERSION=1.1
```

- Get version `<ADA_MODE_VERSION>` of the Ada mode from http://stephe-leake.org/emacs/ada-mode/emacs-ada-mode.html. Check the version of the Ada mode provided with your Emacs, it could be the relevant one!
- Install the Ada mode if needed :

```
mkdir <PREFIX>/ada-mode/
cd <PREFIX>/ada-mode/
tar zxvf ada-mode-${ADA_MODE_VERSION}.tar.gz
cd -
```

- Unpack the SPARK Ada mode :

  ```
  tar zxvf spark-ada-${SPARK_MODE_VERSION}.tar.gz
  ```

  The directory `spark-ada-<SPARK_MODE_VERSION>` will be created.

- Overwrite the Ada mode by the SPARK Ada mode :

  ```
  cp -f <PREFIX>/spark-ada-${SPARK_MODE_VERSION}/*.el <PREFIX>/ada-mode/
  ```

- Make the SPARK Ada mode visible for Emacs by adding in your `.emacs` :

  ```
  (add-to-list 'load-path "<PREFIX>/ada-mode")
  ```

- By default the SPARK Ada mode is not activated. To activate the SPARK Ada mode, use the `Ada` → `Customize` menu from the Ada mode and update the entry : "Ada Spark Mode". This will update your `.emacs` by adding :

  ```
  (custom-set-variables
    '(ada-spark-mode t)
  )
  ```

- When opening an Ada file in Emacs, the SPARK Ada mode will be used.

- The specific SPARK Ada mode functionalities are described in the usage section of `spark-mode.el`.

3 Configuration

The SPARK mode provides the following configuration parameters:

ada-fill-spark-annotation	`#`	Text inserted in the first columns when filling a SPARK annotation paragraph.
ada-spark-examiner-command	`spark`	Name of the SPARK Examiner and the options to use.
ada-spark-sparksimp-command	`sparksimp -t`	Name of the SPARKSimp and the options to use. Don't provide the `-p=` option.
ada-spark-simplifier-command	`spadesimp`	Name of the Simplifier and the options to use.
ada-spark-pogs-command	`pogs`	Name of the POGS and the options to use.
ada-spark-mode	`nil`	Set SPARK Ada mode.
ada-spark-number-of-jobs	2	Specifies the number of jobs (commands) to run simultaneously.

4 Functionalities

The SPARK mode provides the following functionalities:

`TAB`	indent SPARK annotations
`ENTER`	add the SPARK annotation `--#` when you insert a newline inside the SPARK annotations + indent the new line
TYPE	adjust casing for identifier in SPARK annotations
`ESC-/` or `META-/`	auto-completion in SPARK annotation
`C-c C-f`	reformat proof functions
`C-c C-i`	reformat/sort inherit annotations
`C-c C-g`	reformat/sort own/global annotations
`C-c C-h`	reformat/compress/sort derives annotation
Ada → SPARK → SPARK current file (spark−current−file)	SPARK the current file
Ada → SPARK → SPARK metafile (spark−metafile)	SPARK the metafile
Ada → SPARK → SPARK/Proof current body (spark−proof−current−body)	SPARK and Simplify the current body (without the nested subprograms) + open the related SIV file
Ada → SPARK → SPARK/Proof current subprogram (spark−proof−current−subprogram)	SPARK, Simplify and POGS the current subprogram (with the nested subprograms) + open the related SIV file and SUM file if nested subprograms
Ada → SPARK → SPARK/Proof current file (spark−proof−current−file)	SPARK, Simplify and POGS the current file + open the top level SUM file
Ada → SPARK → SPARK/Proof metafile (spark−proof−metafile)	SPARK, Simplify and POGS the metafile + open the top level SUM file
Ada → SPARK → Proof all (spark−proof)	Simplify all and POGS + open the top level SUM file
(spark−fix−error)	automatic fix of SPARK semantic, flow and capitalisation errors
`C-c C-d`	`Goto Declaration/Body` cross navigation in SPARK annotations
(ada−find−any−references)	`List References` cross navigation in SPARK annotations using gnatfind or gps
AUTO	syntax highlighting of SPARK keywords in annotation
(ada−rename−identifier)	renames identifier in Ada source code and in SPARK annotations
(spark−pretty−print)	pretty print the SPARK source code
(ada−format−call−paramlist)	reformat subprogram call or aggregate

4.1 Indentation

To indent a line in a SPARK annotation, the cursor must be located after the `--#`. If the cursor is located before the `--#`, only the `--#` will be indented as a normal Ada comment. The corresponding SPARK annotation on the same line will not be indented.

4.2 Reformatting

All the SPARK reformatting functionalities are performed in the standard way of the Emacs Ada mode like `C-c C-f` to reformat subprograms parameters list : put the cursor in the area that has to be reformatted and run the relevant reformatting command.

- Put the cursor in the parameters list of a proof function and type `C-c C-f`.

- Put the cursor in a `inherit` annotation and type `C-c C-i`.
- Put the cursor in a `own` annotation and type `C-c C-g`.
- Put the cursor in a `global` annotation and type `C-c C-g`.
- Put the cursor in a `derives` annotation and type `C-c C-h`.

The SPARK reformatting functionalities don't use `sparkformat`. So, `sparkformat` has not to be present or visible.

4.3 SPARK/Proof

- SPARK/Proof current body : The cursor must be located in the body of a SPARK source code. SPARK the current file; simplify the VCG file for the selected body; open the SIV file for this specific body.
- SPARK/Proof current subprogram : The cursor must be located in the body of a SPARK source code. SPARK the current file; simplify the VCG file for the selected body; if they are nested subprograms, simplify all the subprograms recursively and run POGS on all the nested subprograms; open the SIV file for this specific body and open the SUM file for the nested subprograms.
- SPARK/Proof current file : SPARK the current file; simplify everything from the top level and run POGS from the top level; open the top level SUM file.
- SPARK/Proof metafile : SPARK the given metafile; simplify everything from the top level and run POGS from the top level; open the top level SUM file.

The Ada SPARK mode uses the ada−fill−spark−annotation configuration parameter to SPARK a file or a metafile.

The Ada SPARK mode uses the ada−spark−sparksimp−command and ada−spark−number−of−jobs configuration parameters to simplify a set of subprograms recursively. The last one (ada−spark−number−of−jobs) is used to specify the number of simplifiers to run simultaneously.

The Ada SPARK mode uses the ada−spark−simplifier−command configuration parameter to simplify a specific VCG file.

The Ada SPARK mode uses the ada−spark−pogs−command configuration parameter to run POGS.

4.4 SPARK Fix Error

SPARK Fix Error works in a similar way than GNAT Fix Error but for SPARK annotations. To fix normal Ada errors in the SPARK source code, it is better to use GNAT Fix Error and not SPARK Fix Error. The goal of SPARK Fix Error is not to duplicate GNAT Fix Error but to fix specific SPARK errors. For this reason, SPARK Fix Error should be use in combination with GNAT Fix Error.

Before fixing any SPARK errors, make sure that SPARK Fix Error is available for Emacs. The `.emacs` must contain the following line: (**require** 'spark−fix−error).

To fix a SPARK error:

1. Run the Examiner on the SPARK source code using Ada → SPARK → SPARK current file or Ada → SPARK → SPARK metafile.
2. Select the SPARK error that has to be fixed.
3. The related SPARK source code will be loaded and the cursor will be located at the corresponding line/column number.

4. Most of the time, the cursor is not located where the SPARK error must be fixed. For example, the cursor can be located at the end of the subprogram body and the SPARK error must be fixed in the annotation of the subprogram declaration. The cursor has to be moved manually to the relevant location in the SPARK annotation. The cursor may be placed in the `global` or the `derives` annotation.

5. Call the function (spark−fix−error) by typing `M-x spark-fix-error`.

6. SPARK Fix Error will try to fix as many error as it can. SPARK Fix Error will stop when the next error is located in a different file or at a different location in the same file or if the next error is related with a different category of SPARK errors. The following categories are defined:

   - Semantic error;
   - Flow error;
   - Stop after the error for isolated SPARK error.

The list of SPARK errors that can be fixed automatically is defined in the file `spark-fix-error.el` in the function (spark−fix−one−error).

4.5 SPARK Pretty Print

SPARK Pretty Print reformats the Ada source code and the SPARK annotations as well. To get better result, SPARK Pretty Print can be run after `gnatpp -c5`. The `-c5` option must be used to left the SPARK annotations untouched.

To run SPARK Pretty Print in batch mode,

- an Emacs lisp script has to be written to call (spark−pretty−print) in the good context. Lets call this script `spark-pretty-print.el` and the content of the file is as follow:

  ```
  (defun spark-pretty-print-batch ()
    (add-to-list 'load-path "<PREFIX>/ada-mode/")
    (custom-set-variables
     '(ada-spark-mode t))
    (load-file "<PREFIX>/ada-mode/ada-mode.el")
    (ada-mode)
    (spark-pretty-print)
    (save-buffers-kill-emacs t))
  ```

- then, the following command can be issued:

  ```
  emacs --batch --load spark-pretty-print.el --find-file ${SPARK_FILE} \
     --funcall spark-pretty-print-batch
  ```

5 Licence