



Tool Development and Support

SPARK Toolset Release Note – Release 7.3

EXM/RN
Issue: 2.0
Status: Definitive
10 January 2006

Originator

SPARK Team

Approver

SPARK Team Line Manager



Copyright

The contents of this manual are the subject of copyright and all rights in it are reserved. The manual may not be copied, in whole or in part, without the written consent of Praxis High Integrity Systems Limited.

The software tools referred to in this manual are the subject of copyright and all rights in them are reserved. The rights in these tools are owned by Praxis High Integrity Systems Limited, and it may not be copied, in whole or in part, without the written consent of this company, except for reasonable back-up purposes. The same proprietary and copyright notices must be affixed to any permitted copies as were affixed to the original. This exception does not allow copies to be made for others, whether or not sold, and none of the material purchased may be sold, given or loaned to another person or organisation. Under law copying includes translating into another language or format.

©1991-2006 Praxis High Integrity Systems Limited, 20 Manvers St, Bath BA1 1PX

Limited Warranty

Praxis High Integrity Systems Limited save as required by law makes no warranty or representation, either express or implied, with respect to this software, its quality, performance, merchantability or fitness for a purpose. As a result, the licence to use this software is sold 'as is' and you, the purchaser, are assuming the entire risk as to its quality and performance.

Praxis High Integrity Systems Limited accepts no liability for direct, indirect, special or consequential damages nor any other legal liability whatsoever and howsoever arising resulting from any defect in the software or its documentation, even if advised of the possibility of such damages. In particular Praxis High Integrity Systems Limited accepts no liability for any programs or data stored or processed using Praxis High Integrity Systems Limited products, including the costs of recovering such programs or data.

SPADE is a trademark of Praxis High Integrity Systems Limited

Note: The SPARK programming language is not sponsored by or affiliated with SPARC International Inc. and is not based on SPARC™ architecture.



Contents

1	Introduction	6
2	Contact Information	7
3	SPARK language changes	8
3.1	Static constraint checking of real expressions	8
3.2	Mixed mode refinements of abstract own variables	8
4	Other significant Examiner changes	11
4.1	Support of 64-bit Floating-point	11
4.2	Handling of VC generation in presence of semantic errors	11
4.3	Handling of VC generation in presence of data flow errors	11
4.4	Error Explanations option	12
5	Miscellaneous Examiner improvements	13
5.1	Reported illegal re-declaration of record fields.	13
5.2	Use of quantifiers of Boolean type	13
5.3	Non-local record subtypes	13
5.4	Syntax tree limit increased	13
5.5	Pragma on private subprogram	13
5.6	Unprovable VCs for functions which access external variables	13
5.7	Improved warnings for potentially invalid values	13
5.8	Qualification rule for modular literals relaxed	14
5.9	Improved validity warning on externals and unchecked conversions	14
5.10	User-defined hidden exception handlers	15
5.11	Addition of “false” VC for subprograms with a semantic error	15
5.12	Suppression of R-values in VCG where subprogram has a data flow error	15
5.13	Static constraint error on real numbers in SPARK 83	15
5.14	Improved algorithm for $A*B$	15
5.15	Generation of proof rules for ‘Size	15
5.16	Pragma import and variables	16
5.17	Semantic Error 156	16
5.18	Global “nolistings” option	17
5.19	Brief mode briefer	17
5.20	/nowarning_file override	17
5.21	Better diagnosis of common syntax errors	17
5.22	Elimination of SPARK_ERRORS environment variable	17
5.23	Error counting	18
5.24	Creation of ‘megaspark’	18
5.25	Derives null and /flow=data mode	18
5.26	FDL declaration order	18
6	SPADE Simplifier	19
6.1	Correction to simplification of VCs with Records	19
6.2	Improved detection of Simplifier failures	19
6.3	Speed of contradiction hunter	19
6.4	Memory exhaustion in Simplifier 2.17 and above	19
6.5	Common non-linear arithmetic cases	19



6.6	Use of Examiner-generated proof rules	19
6.7	Use of rational inequalities	20
6.8	Simplifier performance with 'Pos and 'Val rules	20
6.9	User-defined proof rules	20
7	SPADE Proof Checker	21
7.1	Plain output mode added	21
7.2	Checker unification rules 2-9 are usable	21
7.3	Rerunning of scripts containing execute commands	21
7.4	Addition of overwrite_warning flag	21
7.5	New rules in MODULAR rule family	21
7.6	Load command	22
8	POGS	23
9	SPARKFormat	24
10	SPARKMake	25
11	Backward incompatibilities	26
11.1	Examiner	26
11.2	SPADE Simplifier and Checker	26
12	User manual change summary	27
12.1	SPARK95 – The SPADE Ada95 Kernel	27
12.2	SPARK83 – The SPADE Ada Kernel	27
12.3	Examiner User Manual	27
12.4	Generation of RTCs for SPARK Programs	27
12.5	Generation of VCs for SPARK Programs	27
12.6	Installation manuals – SUN, Windows and VAX/VMS	27
12.7	Generation PFs for SPARK Programs	27
12.8	Checker User Manual	27
12.9	Checker Rules	28
12.10	SPARKFormat User Manual	28
12.11	SPARKMake User Manual	28
12.12	POGS User Manual	28
12.13	Simplifier User Manual	28
13	Limitations and known errors	29
13.1	Tool limitations	29
13.2	Known error summary	31
14	Change Summary from Release 2.0	32
14.1	Release 2.0 - November 1995	32
14.2	Release 2.1 - July 1996	32
14.3	Release 2.5 - March 1997	32
14.4	Release 3.0 - September 1997	32
14.5	Release 4.0 - December 1998	33
14.6	Release 5.0 - June 2000	33
14.7	Release 6.0 - November 2001	34
14.8	Release 6.1 - June 2002	34
14.9	Release 6.3 - December 2002	36



14.10 Release 7.0 – July 2003	36
14.11 Release 7.2 – December 2004	36
15 Operating system compatibility	38
15.1 VAX/VMS	38
15.2 SPARC/Solaris	38
15.3 Windows NT, 2000 and XP	38
15.4 Intel/Linux	38
Document Control and References	39
Changes history	39
Changes forecast	40
Document references	40
File under	40



1 Introduction

Continued development of the SPARK Toolset has resulted in the development of a number of useful features, which Praxis High Integrity Systems wish to make available before the next planned major release. These developments have been incorporated into Toolset Release 7.3.

This document describes changes in the behaviour of all variants of the SPARK Toolset Release 7.3 compared to Release 7.2.



2 Contact Information

For further information about this document please contact Praxis High Integrity Systems:

By phone: +44 (0)1225 823829 (direct line), +44 (0)1225 466991 (exchange)

By FAX: +44 (0)1225 469006

By email: sparkinfo@praxis-his.com



3 SPARK language changes

3.1 Static constraint checking of real expressions

The semantics of static expression evaluation changed between Ada 83 and Ada 95. In the latter case, static expressions are evaluated to “infinite precision” before being converted to the target type. In the former, they are evaluated using machine arithmetic. The Examiner respected this difference when performing static constraint checks on assignments to real-number objects. An apparent out of range value was reported as a *warning* in SPARK 83 mode (because we cannot be sure of the *exact* value of the expression since implementation arithmetic has been used) and as an *error* in SPARK 95.

We now conclude that this attempt to be faithful to Ada semantics, in this particular case, raises more problems than it solves. In particular, it raises the risk that an important static constraint error may be overlooked because it “only generated a warning”. With Release 7.3, static expressions are checked for range violations using infinite precision arithmetic in all cases. Note that this change may, in extreme cases, render existing programs illegal.

For example:

```
type R is digits 5 range 0.0 .. 100.0;
C : constant R := 100.0 + 0.00001;
-- The above would have raised a warning prior to Release 7.3 but is now regarded as an error;
-- this is despite the fact that the code may compile correctly because the value might be in range
-- when evaluated using machine arithmetic.
```

The warning control file keyword `static_real_constraints` is no longer available.

3.2 Mixed mode refinements of abstract own variables

Strictly speaking, this section does not describe a SPARK language change. It does clarify an existing SPARK rule that was wrongly implemented in SPARK Examiners prior to Release 7.3. The description is placed here rather than in sections 4 or 5 because it is important and because it may have backward compatibility issues for existing code.

3.2.1 Background

SPARK permits an unmoded abstract own variable to be refined on to an unrestricted mixture of unmoded constituents and external variables with mode in or mode out. An example of such mixed-mode refinement can be found in Figure 3 of Section 7 of the SPARK language definition.

Where the abstract own variable is marked with an “initializes” clause, SPARK requires that *all* of its unmoded constituents must be initialized and *none* of its moded constituents may be; this initialization can be at declaration or during package elaboration. The rule is designed to eliminate subtle elaboration order dependencies arising from the reading and writing of external ports during program start up.



Prior to release 7.3, the Examiner wrongly applied this same initialization rule to a *procedure* which updated the own variable: if the procedure unconditionally updated all of the unmoded constituents, it was regarded as wholly defining the own variable at the abstract level.

3.2.2 Example

Consider a package `P` which has an abstract own variable `State`. `State` is refined onto 3 concrete constituents: an simple, unmoded variable `Flag`; an external variable, `Inputs`, of mode `in`; and an external variable, `Outputs`, of mode `out`.

A procedure, `Transfer`, that copies data from `Inputs` to `Outputs` clearly has the concrete annotation “`derives Outputs from Inputs`” and this clearly is equivalent to “`derives State from State`” at the abstract level.

A procedure, `Reset`, that simply clears the `Flag` variable would have the concrete annotation “`derives Flag from ;`”. The rules for checking abstract and refined annotation consistency are at section 7.2.1 of the SPARK language definition. Item (2) from the first enumerated list in that section is applicable to the operation `Reset`. A refinement constituent of `State`, in the form of `Flag`, appears in `Reset`’s annotation with mode `out` (i.e. it is a pure export). Furthermore, other refinement constituents of `State` (`Inputs` and `Outputs`) are missing from the annotation of `Reset`. The rules therefore requires us to give `State`, in the abstract annotation, the mode `in out` and add a self dependency. The correct abstract annotation of `Reset` is therefore “`derives State from State`”. *Prior to Release 7.3, the Examiner wrongly regarded is as being “`derives State from ;`”.*

Confusing flow analysis can occur when an external user constructs an operation that combines the `Transfer` and `Reset` operations. For example:

```
procedure TransferAndReset
--# global P.State;
--# derives ...; -- see notes below
is
begin
  P.Transfer;
  P.Reset;
end TransferAndReset;
```

This procedure causes an flow of data from one constituent of `State` to another followed by the resetting of one part only of `State`. Clearly the overall, abstract description of this operation is “`derives P.State from P.State`”. With the previous incorrect Examiner behaviour, users would be required to change it to the incorrect “`derives State from ;`” in order to eliminate flow error messages.

3.2.3 Consequences and backward compatibility

There are two considerations for users arising from this Examiner correction.



Firstly, it is possible that existing code may have been wrongly annotated in a way that matches the Examiner's incorrect analysis. This would leave it free from flow errors but with a wrongly stated flow analysis "contract" leading to possible problems elsewhere in the overall program. The results of re-examining code of this form with Examiner Release 7.3 will be change of flow analysis results that may require some rework. We believe this risk to be fairly small; our attempts to construct silently incorrect examples have generally created the opposite problem: code which it is impossible to annotate in any way that removes Examiner error messages.

The second consequence of the SPARK language rule clarification is for the initialization of mixed mode abstract own variables. Any procedure that updates some or all of the constituents of such a variable will have the abstract annotation "derives State from State". It will either:

- 1 Write to *some but not all* of the constituents and thus create a self dependency under SPARK rule 7.2.1(2); or
- 2 write to *all* of the constituents including any external variables of mode out, which also creates a self-dependency at the abstract level.

It follows that the *only* way of initializing a mixed mode abstract own variable is by initializing its unmoded constituents at declaration or package elaboration and by marking the own variable with an "initializes" annotation.



4 Other significant Examiner changes

This section documents other significant changes to the SPARK Examiner made for release 7.3. Where appropriate, SPARK Examiner Performance Report (SEPR) numbers are given.

4.1 Support of 64-bit Floating-point

The Examiner models all real numbers as rational pairs and evaluates static expressions precisely. Internal static limits on the representation of these rational pairs bound the size of numbers that the Examiner can support. Previous versions of the Examiner only supported IEEE 32-bit Floating-point, Release 7.3 of the Examiner extends the range, and in particular, allows the specification of `Long_Float'First` and `Long_Float'Last` in the Configuration file for typical IEEE 64-bit implementations of `Long_Float`.

4.2 Handling of VC generation in presence of semantic errors

Historically the Examiner has always suppressed VC generation for subprograms which contain static semantic errors. This is consistent with the recommendation to deal with all semantic and flow errors before attempting VC generation. The historical behaviour does, however, carry with it a small risk: if VCs are successfully generated for a subprogram and then a semantic error is introduced to it, then the previously generated VC file is not overwritten and only its date/time stamp gives a clue that it might be obsolete.

With Release 7.3, the Examiner now *always* generates a VC file for each subprogram; however, if semantic errors have been detected then the VC is deliberately made unprovable with a `False` conclusion. A warning message to this effect is echoed to the screen and included as a comment in the VC file. The `False` VC is detected by PoGS and make it harder to overlook the real cause of the error.

Similar behaviour occurs when path functions are generated. In this case, a null path function, signifying that the defective subprogram does nothing, is generated.

Notwithstanding this change, users are reminded of the importance of checking the report file for errors and warnings and not relying solely on the VC generator and Simplifier.

4.3 Handling of VC generation in presence of data flow errors

Release 7.2 of the Examiner exploited a benefit of data flow analysis by assuming that a local variable, anywhere it is referenced in an expression, must be validly in type. The effect of this is to generate additional hypotheses that aid the discharge of VCs. As noted in the Release 7.2 Release Note, the validity of the behaviour depends on the absence of data flow errors in the code for which proof is being attempted. To avoid the risk of an unsound proof being constructed because a data flow error has been accidentally overlooked, Release 7.3 now only generates assumptions about the validity of local variables if the flow analyser has shown the related sequence of statements to be free from conditional and



unconditional data flow errors. The change increases the security of the proof process at the cost of a possible slight decrease in the number of VCs automatically discharged.

Notwithstanding this change, users are reminded of the importance of checking the report file for errors and warnings and not relying solely on the VC generator and Simplifier.

4.4 Error Explanations option

The Examiner now has the option to append a more detailed explanation (as appears in the Examiner User Manual) to each error and warning appearing in the report file, listing files, and on-screen.

A new command-line switch called “error_explanations” (abbreviation “er”) controls this behaviour. Valid values for this switch are “off”, “first_occurrence”, and “every_occurrence”, which may be abbreviated to “o”, “f” and “e” respectively.

If “off” is selected, then no explanations are produced.

If “first_occurrence” is selected, the explanation is appended to the first occurrence of each warning or error message in each of the report file, listing file(s), and screen output.

If “every_occurrence” is selected, an explanation is appended to every warning and error message.

The default value is “off”.

Example:

```
spark /er=f p1.adb
```



5 Miscellaneous Examiner improvements

5.1 Reported illegal re-declaration of record fields.

The Examiner erroneously reported a record field as illegally re-declared if the name used for the record field had been previously used as a name in the scope of the record declaration. A record field name must be unique within the declaration of the record type but can be common with other names declared in the scope of the record declaration (SEPR 1767, SEPR 610). This has been corrected in release 7.3.

5.2 Use of quantifiers of Boolean type

The Examiner now disallows explicit use of a range constraint in a quantified expression when the quantified type is Boolean (SEPR 1779). This gives greater consistency with the language design, since constrained Boolean subtypes are forbidden.

5.3 Non-local record subtypes

Improvements have been made to allow analysis of a record aggregate where the qualifier denotes a local subtype of a non-local record (SEPR 1781).

5.4 Syntax tree limit increased

The Examiner's syntax tree limits have been increased (SEPR 1794)

5.5 Pragma on private subprogram

The Examiner is now able to handle the case where pragma Import appears in a private part, (SEPR 1797).

5.6 Unprovable VCs for functions which access external variables

The tilde (~) notation is now allowed in function return annotations for external variables of mode 'in'. This allows the construction of partial correctness proofs for such functions and their callers.

5.7 Improved warnings for potentially invalid values

Where values are read from external devices or generated by use of instantiations of Unchecked_Conversion, it is possible that invalid representations may result. Such invalid values may cause run-time errors or other erroneous program behaviour. Invalid values are of particular concern in



SPARK because they may undermine the logical framework in which proof, including proof of freedom from run-time errors, is carried out.

The Examiner takes two steps to alert users to the problem of invalid values:

- 1 It warns when an external variable is read or an instantiation of unchecked conversion is used.
- 2 It does not assume that the values obtained by these means are correctly “in type” and so does not generate hypotheses to state that they are. As a result, unprovable VCs result if an attempt is made to use such a value without first validating it in some way.

Since mechanism 2 is only helpful in the case of scalar types (excluding Boolean) it follows that mechanism 1 provides the *only* warning of the potential hazard for invalid Boolean or structured types.

Release 7.3 strengthens the protection provided by the Examiner by producing different levels of warning for activities that might generate invalid values depending on whether additional protection is provided by the run-time check mechanism or not. When the type read from an external variable or generated by an unchecked conversion is scalar (excluding Boolean) then a weaker, suppressible warning is generated; this is identical to earlier Examiner releases. However, when the type is one for which run-time checks provide no further protection (Boolean or structured types), a stronger warning which cannot be suppressed is generated. This distinction reduces the risk of overlooking the more serious cases because of the “noise” generated by the less serious ones.

5.8 Qualification rule for modular literals relaxed

Qualification rules for modular literals below relational operators are no longer required. E.g. If T is a modular type variable:

If $T > 2$ then ...

is allowed.

5.9 Improved validity warning on externals and unchecked conversions

When a potentially invalid value might be introduced i.e. by unchecked conversion, or by reading an external variable, the Examiner issues a warning and suppresses VC hypotheses. This behaviour has been improved to distinguish between cases where the lack of hypotheses gives protection (non Boolean scalars – generates a suppressible warning) and cases where it doesn't (Boolean or non-scalar - non-suppressible warning).



5.10 User-defined hidden exception handlers

SPARK allows for hidden exception handlers in any subprogram_implementation structure. These are considered outside the SPARK boundary, but are allowed syntactically for platforms where this makes sense.

5.11 Addition of “false” VC for subprograms with a semantic error

A subprogram containing a semantic error previously did not generate any VCs. Now an explicit “False” VC is generated, which will show up in the POGS output.

5.12 Suppression of R-values in VCG where subprogram has a data flow error

If a subprogram has an unconditional or conditional data-flow error, the generation of R-value hypotheses in the VCG is suppressed.

5.13 Static constraint error on real numbers in SPARK 83

Previously the Examiner issued warning 201 when it statically detected a constraint error involving a real number in SPARK 83 mode. In all other cases error 402 is raised. The warning was an attempt to deal with Ada 83 semantics, covering extreme cases where a number appears to be out of range but calculates, using machine semantics, to a valid value. For clarity and safety SPARK now diverts from Ada semantics at this point and makes `l` illegal to have an apparently out-of-range value at all.

5.14 Improved algorithm for $A**B$

A faster algorithm has been implemented to improve performance for $A**B$. Previously this was very slow for large values of `B`.

5.15 Generation of proof rules for ‘Size

The Examiner now considers the value specified for ‘Size, and checks that it is a static expression of base type integer. If ‘Size is not explicitly specified the Examiner generates a proof rule that its value is greater than or equal to zero. If it is explicitly specified, it generates a proof rule for this value. The Examiner does not check whether the value is a sensible one, hence the warning “Representation clause - ignored by the SPARK Examiner” continues to be generated.



5.16 Pragma import and variables

The Examiner now allows the use of pragma import after a variable as well as after a subprogram. The pragma must immediately follow the variable declaration.

Attaching a pragma Import to a variable has some important semantic effects. In many ways it is like attaching an address clause to the variable. The Examiner currently makes some consistency checks between the properties asserted of an own variable and those implied by the presence of an address clause; these checks are now extended to include the use of pragma Import. Essentially the checks are to avoid “concealed volatility” of variables caused by them being connected to the environment in some way without that fact being visible to the Examiner.

Unlike an address clause, the use of pragma Import is also regarded as initializing the variable to which it is attached. Indeed, Ada prohibits an explicit initialization of such a variable. Clearly users must ensure that the variable is indeed properly initialized, with a valid value, in the place from which it is being imported.

Connecting local variables of subprograms to the external environment remains strongly deprecated.

Example: a simple input port.

```
package Port
--# own in Inputs;
is
  procedure Read (X : out Integer);
  --# global in Inputs;
  --# derives X from Inputs;
end Port;

package body Port
is
  Inputs : Integer;
  pragma Import (C, Inputs, InputLinkName);

  procedure Read (X : out Integer)
  is
  begin
    X := Inputs; -- validity checking omitted for simplicity
  end Read;
end Port;
```

5.17 Semantic Error 156

The error message for Error 156 (entire variable expected) has been improved by the creation of Error 150 that covers the case when the name of a constant is found in a mandatory annotation such as a global or derives annotation.



5.18 Global “nolistings” option

A `/nolistings` switch has been added which suppresses the generation of all listing files. This is in addition to the ability to specify `/nolisting` for individual files.

e.g. `spark /nolistings sparkfile.adb`

will not generate any listing files, while

`spark /listing_extension=ls_ onespark.ads, onespark.adb, twospark.ads /nolisting`

will generate listing files `onespark.lss` and `onespark.lsb`, but not `twospark.lss`

5.19 Brief mode briefer

The option `/brief` now suppresses everything except for warnings and errors, creating a more concise summary of the Examiner’s findings.

5.20 `/nowarning_file` override

The Examiner will not allow contradictory options to be specified at the same time e.g. `spark /statistics /nostatistics sparkfile.ada`. This holds even if one option is specified in the `spark.sw` file and the other on the commandline. An exception to this is `/nowarning_file` and `/warning_file` which can now override each other. The Examiner will use whichever option is specified last.

In particular, `/nowarning_file` may be given on the command-line even if `/warning_file=XXX` is given in the default switch file. This is useful for sanity checking and “regression analysis” of a project where no warning suppression is required.

5.21 Better diagnosis of common syntax errors

The warnings generated by several common syntax errors have been improved to help the user diagnose the problem more easily.

5.22 Elimination of `SPARK_ERRORS` environment variable

The environment variable `SPARK_ERRORS` is no longer used to locate the `errors.htm` file. Instead, the Examiner expects to find the `errors.htm` file in a directory “`../lib/spark/`” relative to the directory where the Examiner binary is installed.



5.23 Error counting

The total number of errors, warnings, and suppress warnings generated by the Examiner is summarized on screen.

5.24 Creation of ‘megaspark’

Examiner release 7.3 ships with megaspark as standard. This is an Examiner with increased capacity in its internal tables.

5.25 Derives null and /flow=data mode

Spurious ineffective statements were reported when /flow=date and a procedure was called which derives null from a single import. This has now been corrected.

5.26 FDL declaration order

FDL declarations are now in the correct order (SEPR 1025).



6 SPADE Simplifier

SPADE Simplifier version 2.22 ships with toolset release 7.3. This version includes the following improvements over and above version 2.17:

6.1 Correction to simplification of VCs with Records

If a VC included a record with a field name that is overloaded with another name in the scope of the VC then the Simplifier would erroneously substitute the record field name resulting in VCs that do not type check. This has been corrected in version 2.22 (SEPR 1766, SEPR 976)

6.2 Improved detection of Simplifier failures

Increased visibility of Simplifier failures during the running of SPARKSimp. Failure messages are now displayed on the output screen (SEPR 1793).

6.3 Speed of contradiction hunter

The speed of the contradiction hunter when given a large number of hypotheses has been improved (SEPR 1800).

6.4 Memory exhaustion in Simplifier 2.17 and above

There has been a re-design of new arithmetic tactics to eliminate the potential for unlimited recursion. (SEPR 1806).

6.5 Common non-linear arithmetic cases

The Simplifier has improved tactics for dealing with expressions involving the 'mod' operator (SEPR 1846).

6.6 Use of Examiner-generated proof rules

The Simplifier is now capable of taking advantage of the "may_de_deduced", "may_be_deduced_from", and "may_be_replaced_by" rules generated by the Examiner, even if these involve variables and side-conditions. These assist in proofs involving the 'Pos and 'Val attributes, the 'Valid attribute, and the use of composite constants.



6.7 Use of rational inequalities

The capability to multiply out rational inequalities has been added to the Simplifier (SEPR 1866)

6.8 Simplifier performance with ‘Pos and ‘Val rules

The Simplifier is now able to take advantage of proof rules involving ‘Pos and ‘Val generated by the Examiner. (SEPR 1911)

6.9 User-defined proof rules

The Simplifier now has the ability to read and exploit user-defined proof rules. See section 7 of the Simplifier User Manual for details (SEPR 1867).



7 SPADE Proof Checker

SPADE Proof Checker version 2.06 ships with toolset release 7.3. This includes the following improvements over and above version 2.03:

7.1 Plain output mode added

Plain output mode suppresses the checker version number, the date and time stamp, and full path-names in the PLG file.

7.2 Checker unification rules 2-9 are usable

Subgoals of Checker rules unification(2) – unification(9) were incorrect and unusable in release 2.03. These have been corrected in release 2.06.

7.3 Rerunning of scripts containing execute commands

Previously the .cmd file recorded all commands executed by the Checker – hence both the command “execute” and subsequent commands within the executed file were logged, and the resultant script could not be replayed. The .cmd command now records only the first call to execute and recommences logging once that file has been completed. This means that the recorded scripts can be rerun.

7.4 Addition of overwrite_warning flag

The overwrite_warning qualifier switches on a warning to alert you when old command log or proof log files are deleted. If the Checker needs to create a command log file and there is already a file in the directory of that name, it will move it. So, for example, name.cmd is moved to name.cmd-. If both files are already in the directory, the Checker will delete the name.cmd- file first, then move the existing name.cmd file to name.cmd-, leaving it free to create a new name.cmd file for the current command log. The same behaviour exists for the proof log. Overwrite_warning will consult the user before deleting the file rather than deleting it silently.

7.5 New rules in MODULAR rule family

Three useful new rules have been added in the MODULAR rule family. These are:

```
modular(32): N mod N may_be_replaced_by 0 if [ N <> 0 ].  
modular(33): N mod N may_be_replaced_by 0 if [ N > 0 ].  
modular(34): N mod N may_be_replaced_by 0 if [ N < 0 ].
```



7.6 Load command

The `load` command is now obsolete. The `consult` command can be used instead to read in rules.



8 POGS

POGS version 4.4 ships with the toolset release 7.3. This release includes the following improvements:

The subprograms containing undischarged VCs are listed at the beginning of the summary section, along with the number of VCs remaining to be discharged. This enhances the readability and makes it easier to see at a glance where the problems are occurring (SEPR 1782).

Recognition of VCs which have been proved to be false by the Examiner or the Simplifier has been added to the individual listings by VC and the end summary. Any subprograms containing false VCs will be listed at the beginning of the summary section. Previously POGS made no distinction between VCs proved false and unproved VCs.

VCs which have been proved by contradiction, if any, are listed at the beginning of the summary section (SEPR 1791).

POGS has increased capacity to handle long/deep directory trees (SEPR 1783).

Note that POGS is not supplied on VAX/VMS.



9 SPARKFormat

Version 7.3 of SPARKFormat is shipped with the toolset release 7.3.

This version of SPARKFormat provides additional command line options to allow the user to tailor the layout of annotations. The user can print each variable on a separate line and specify the size of the indentation from '-#', or can specify 'inline' which aligns each element (variable or separator) with respect to the previous element.

The new switches are: `global_indent`, `export_indent`, `import_indent`, `separator_indent`. Each takes a number greater than 0, or the word 'inline'.

Information about the tool is now available with the new switches `/help` and `/version`. `Help` provides a listing of all command line options for SPARKFormat, and overrides all other switches. `Version` provides the version number of the tool and overrides all switches except `help`.

Note that SPARKFormat is not supplied on VAX/VMS.



10 SPARKMake

SPARKMake version 7.3 ships with the toolset release 7.3. This release includes the following improvements:

SPARKMake no longer fails if the root file is not in any directory specified, nor in the current directory.

The switch `/duplicates_are_errors` has been added. When this switch is specified if duplicate units are found in different files, SPARKMake will fail. The default behaviour has been changed so that if this switch is not specified and duplicate units are found a warning will be raised, but SPARKMake will not fail.

Information about the tool is now available with the new switches `/help` and `/version`. Help provides a listing of all command line options for SPARKFormat. This replaces the previous method of accessing help by leaving the command line blank. Version provides the version number of the tool. Once either switch is found on the command line they override all other switches.

Note that SPARKMake is not supplied on VAX/VMS.



11 Backward incompatibilities

11.1 Examiner

There are no known backward incompatibilities introduced between Examiner Releases 7.2 and 7.3.

11.2 SPADE Simplifier and Checker

The Simplifier may prove VCs that it previously failed to prove. This may render existing Checker proof scripts for those VCs redundant.



12 User manual change summary

The following user manuals have been updated as part of this release:

12.1 SPARK95 – The SPADE Ada95 Kernel

- Updated to include extension of SPARK95 for the use of pragma Import on variables, and clarification of rules regarding 'Succ and 'Pred.

12.2 SPARK83 – The SPADE Ada Kernel

- Clarification of rules regarding 'Succ and 'Pred.

12.3 Examiner User Manual

- Adds new error messages, warnings, and command-line switches.

12.4 Generation of RTCs for SPARK Programs

- No changes at present.

12.5 Generation of VCs for SPARK Programs

- Updated to reflect use of tilde in function return annotations and explicit range in quantifiers.

12.6 Installation manuals – SUN, Windows and VAX/VMS

- Updates to reflect new structure and additional files installed with release 7.3.

12.7 Generation PFs for SPARK Programs

- No changes at present.

12.8 Checker User Manual

- Updates to reflect new command line options



12.9 Checker Rules

- Updates to reflect additional rules

12.10 SPARKFormat User Manual

- Updated to describe new formatting options and new command line switches.

12.11 SPARKMake User Manual

- Updated to describe new command line switches.

12.12 POGS User Manual

- Updated to describe changes to summary section.

12.13 Simplifier User Manual

- Updated to describe new simplification tactics and the use of user-defined proof rules.



13 Limitations and known errors

13.1 Tool limitations

This section describes limitations of the Examiner tool arising mainly from incomplete implementation of planned features. Where appropriate a SPARK Examiner Performance Report (SEPR) number is given.

13.1.1 General

- 1 The SPARK 95 language definition removes the distinction between initial and later declarative items; this distinction remains in force in the Examiner that requires SPARK 83 declaration orders even in SPARK 95 mode. (SEPR 813)
- 2 The Examiner does not yet permit the use of 8-bit characters in SPARK 95 userdefined identifiers. (SEPR 818)
- 3 Universal expressions in a modular context may sometimes require type qualification. (SEPR 1591)
- 4 The Examiner does not yet permit the use of “use type” following an embedded package specification. (SEPR 747)
- 5 The Examiner does not yet permit the renaming of packages in the same way that subprograms can be renamed. (SEPR 1391)
- 6 The Examiner does not yet allow the 'Base attribute when not used as a prefix. (SEPR 1114)
- 7 The Examiner does not yet allow S'Range where S is scalar. (SEPR 1115)

13.1.2 Verification Condition Generation and Run-time Checks

- 1 Ada string inequality is not modelled. (SEPR 712)
- 2 VCs involving string catenation that includes the character ASCII.NUL will be incorrect. (SEPR 661)
- 3 Aggregates of multi-dimensional arrays cannot be modelled although aggregates of arrays of arrays can. (SEPR 590)
- 4 Verification conditions involving real numbers are evaluated using infinite precision or perfect arithmetic; this allows the correctness of an algorithm to be shown but cannot guard against the effects of cumulative rounding errors for example.
- 5 The Examiner does not generate VCs for package initialization parts. Statically determinable constraint errors will be detected during well-formation checking of package initialization. (SEPR 288)



- 6 The VC Generator cannot model the implementation-dependent attributes of floating and fixed-point types; see section 13.1.3.

13.1.3 Attribute limitations

13.1.3.1 Unimplemented attributes

The following attributes are officially supported by SPARK according to the language definition, but are not yet implemented by the Examiner. The Examiner will generate error number 30 (“Attribute XXX is not yet implemented in the Examiner”) if you try to use them.

- Adjacent
- Compose
- Copy_Sign
- Leading_Part
- Remainder
- Scaling
- Exponent
- Fraction
- Machine
- Model
- Rounding
- Truncation
- Unbiased_Rounding

Note that these are all function-like attributes concerning floating- and fixed-point types.

13.1.3.2 Unevaluable attributes

The Machine_* and Model_* attributes are accepted by the Examiner, but it does not know how to statically evaluate them since they are inherently implementation dependent. For example, the package:

```
package F is
  type T is digits 6 range -10.0 .. 10.0;
```



```
    C : constant := T'Machine_Emax;  
end F;
```

is legal SPARK, but the Examiner does not know the actual numeric value of C.

13.2 Known error summary

This section lists known errors in the Examiner that are awaiting investigation and correction.

- 1 The SPARK rule that array actual parameters must have the same bounds as the formal parameter is not checked for function parameters where the actual parameter is a subtype of an unconstrained array type. Since subtype bounds are static in SPARK errors of this kind should be detected by an Ada compiler. If not an unconditional run-time error will occur. (SEPR 1060)
- 2 The Examiner permits the body of a subprogram to be entirely made up of proof statements thus breaching the Ada rule that at least one Ada statement must be present. (SEPR 278)
- 3 Where a package declares two or more private types the Examiner permits mutual recursion between their definitions in the private part of the package. (SEPR 848)
- 4 The Examiner does not take due account of a range constraint when determining the subtype of a loop variable; this affects completeness checking of case statements within the loop. For example **for I in Integer range 1..4 loop** would require only values 1, 2, 3 and 4 to be covered by the case statement. (SEPR 693)
- 5 When summarising the counts of pragmas found during an analysis the totals may depend on whether units are selected via the command line (or metafile) or using the index mechanism. The difference affects only pragmas placed *between* program units and arises because placing a file name on the command line causes the entire *file* to be analysed whereas selecting it using indexes causes only the required *unit* to be read from the file. (SEPR 483)



14 Change Summary from Release 2.0

A release note detailing changes from the previous version accompanies each Examiner Release; this section simply summarises the various changes that have been made.

14.1 Release 2.0 - November 1995

Release 2.0 added:

- static expression evaluation;
- variable initialization at declaration;
- full-range scalar subtypes; and
- operator renaming in package specifications.

14.2 Release 2.1 - July 1996

Release 2.1 added:

- facilities for proof of absence of run-time errors

14.3 Release 2.5 - March 1997

Release 2.5 was distributed with “High Integrity Ada - the SPARK Approach” and provided initial facilities for SPARK 95

14.4 Release 3.0 - September 1997

Windows NT was supported for the first time with this release. Release 3.0 also added:

- additional SPARK 95 support;
- flow analysis of record fields;
- command line meta files;
- named numbers;
- unqualified string literals;
- moded global annotations; and



- optional information flow analysis.

14.5 Release 4.0 - December 1998

With Release 4.0 we upgraded all users to a single product standard supporting SPARK 83, SPARK 95 and analysis options up to an including proof facilities. New features were:

- full implementation of public and private child packages;
- default switch file; and
- provision of the INFORMED design document.

14.6 Release 5.0 - June 2000

- Enhanced proof support:
 - I. facilities for proof of programs containing “abstract state”;
 - II. addition of quantified expressions;
 - III. proof rule generation for enumeration types;
 - IV. identification of the kind and source of each VC;
 - V. suppression of trivially true VCs;
 - VI. Proof Obligation Summariser tool (POGS)
- Optional HTML output files with hyperlinks that can be “browsed” interactively
- Better support for common Ada file naming conventions
- User-selectable annotation character
- Improved suppression of analysis were results might otherwise be misleading
- Static expression evaluation in proof contexts
- Singleton enumeration types
- Revised SPARK_IO package
- Error numbering



14.7 Release 6.0 - November 2001

- Introduction of “external variables” to simplify modelling of the interactions between a SPARK program and its external environment.
- Addition of the “null derives” annotation to describe information flows which affect only the external environment.
- Introduction of modular types
- Use of loop labels in exit statements
- Use of global modes on function subprograms
- Extended support for predefined types such as Long_Integer
- Simplified run-time check generation for own variables
- Relaxation of need for mandatory type announcement of own variables
- Plain output option to simplify comparisons of Examiner output files
- Platform-independent switch files and metafiles
- Support for intentionally infinite loops
- Detection of own variables that can never be initialized
- Detection of unusable private types
- Extra refinement checks on global variables when performing data flow analysis
- Detection of unnecessary others clause in case statements
- Extensions to the POGS tool
- Improved error messages to distinguish different cases of variables which are “not declared or visible”
- Improved SPADE Simplifier Release 2.0
- New “SPARKSimp” Tool

14.8 Release 6.1 - June 2002

- Introduction of tagged types



- Introduction of type assertion annotations
- Introduction of modular subtypes
- Introduction of the configuration file
- Introduction of the `help` command line switch
- Demo Examiner now runs on Linux.
- VCG generation for inherited operations of tagged types
- Improved handling of null derives
- Attributes 'Floor and 'Ceiling implemented
- Detection of duplicate record fields
- Improved overflow checks on universal integer expressions
- Corrected handling of loop invariants in while loops
- Strengthened behaviour of `/noecho` option
- Trapping non-positive accuracy in real type declaration
- Recursion in meta-files and index-files
- Improved handling of Address clauses
- Improved handling of Import and Interface pragmas
- VCG Modelling of Boolean membership operators
- Simplification of common Integer inequalities
- Simplification of common enumerated inequalities
- Simplification of VCs involving quantified expressions
- Simplifier performance
- Checker has new built-in rule families: MODULAR, BITWISE and ENUMERATION
- Proved by review option in POGS



14.9 Release 6.3 – December 2002

Release 6.3 of the toolset was constructed to accompany the textbook “High Integrity Software: The SPARK Approach to Safety and Security” by John Barnes, but was not delivered to other users. Its main new features were:

- Slight revision to the rules regarding the placement of tagged type declarations.
- Correction to the modelling of Boolean type membership operators in the verification conditions.
- Support for generating VCs that allow the verification of the Liskov Substitution Principal (LSP) for tagged types and their operations.
- Dramatically improved performance of the Simplifier, particularly in the simplification of quantified expressions.

14.10 Release 7.0 – July 2003

Release 7.0 of the toolset comprised:

Examiner version 7.0

Simplifier version 2.12

Release 7.0 added:

- Ravenscar profile extensions to the language.
- Support for Ada.Interrupts and Ada.Real_Time in the configuration file.
- The new /noduration command line switch.
- VC generation for unconstrained formal parameters.
- Suppression of VC generation for illegal function bodies.
- New “SPARKFormat” tool.

14.11 Release 7.2 – December 2004

Release 7.2 of the toolset comprised:

Examiner version 7.2

Simplifier version 2.17



Release 7.2 added:

- Unconstrained string constants to the language.
- Instantiation of `Unchecked_Conversion` to the language.
- (Full) record subtypes to the language.
- Declaration of subprograms in the private part of packages.
- Refined proof annotations for private types.
- % suffix for referring to value of variable on entry to loop in proof contexts.
- Extra hypotheses for local variables.
- Suppression of VC generation for illegal function bodies.
- Replacement rules for composite constants.
- Concurrent simplification with `SPARKSimp`.
- Improved simplification of VCs with large structured objects.
- Improved simplification of arithmetic and logical expressions.
- New “`SPARKMake`” tool.



15 Operating system compatibility

15.1 VAX/VMS

Examiner 7.3 is available for VAX/VMS releases 5.5-2 and above. The other tools are not available for VAX/VMS at this time.

15.2 SPARC/Solaris

The toolset is compatible with Solaris 5.6 through to 10 including those with a 64-bit kernel.

15.3 Windows NT, 2000 and XP

The toolset is compatible with Windows NT 4.0, Windows 2000, and Windows XP. The executables are also known to work on Windows 95 and 98; however, use of the toolset on these operating systems is unsupported. The FlexLM licence manager software only runs on Windows NT, Windows 2000, or Windows XP.

15.4 Intel/Linux

All the toolset, with the exception of the Checker, is compatible with Intel-based Linux operating systems. Only the “FreeDemo” version of the toolset is currently available for Linux to support buyers of John Barnes’ “SPARK Book.” If you require a full professional SPARK toolset for Linux, then please contact us.



Document Control and References

Praxis High Integrity Systems Limited, 20 Manvers Street, Bath BA1 1PX, UK.
Copyright © Praxis High Integrity Systems Limited 2006. All rights reserved.

Changes history

Issue 0.1 (3rd February 2005): First draft

Issue 0.2 (6th May 2005): Updated following 7.2d03

Issue 0.3 (20th May 2005): Updated following 7.2d04

Issue 0.4 (21st June 2005): Updated following 7.2d05

Issue 0.5 (18th July 2005): Updated following 7.2d06

Issue 0.6 (26th July 2005): Updated following changes to SPARKMake

Issue 0.7 (26th September 2005): Updated following 7.2d07

Issue 0.8 (3rd October 2005): Updated following 7.2d08

Issue 0.9 (17th October 2005): Updated following 7.2d09

Issue 1.0 (14th November 2005): Updated for release 7.2d10

Issue 1.1 (18th November 2005): Updated for Checker 2.05

Issue 1.2 (23rd November 2005): Updated for release 7.2d11 and updated manuals

Issue 1.3 (24th November 2005): Updated for Simplifier release 2.20.

Issue 1.4 (1st December 2005): Updated for Examiner release 7.2d12.

Issue 1.5 (15th December 2005): Updated for Examiner release 7.2d13.

Issue 1.6 (19th December 2005): Updated for Simplifier release 2.21.

Issue 1.7 (3rd January 2006): Updated for Examiner release 7.2d14.

Issue 1.8 (5th January 2006): Updated following changes to SPARKFormat and SPARKmake.

Issue 1.9 (5th January 2006): Updated for Examiner release 7.2d15.

Issue 2.0 (10th January 2006): Updated following review S.P0468.79.90.



Changes forecast

Updates following review.

Document references

File under

CVSROOT/userdocs/Examiner_RN_7p3.doc