Tool Development and Support

SPARK Toolset Release Note 9

RN
Issue: 3.19
Status: Definitive
1st March 2010

Originator

SPARK Team

Approver

SPARK Team Line Manager

Contents

1 Introduction

Continued development of the SPARK Toolset has resulted in the development of a number of useful features that have been incorporated into Toolset Release 9.0.0.

This document describes changes in the behaviour of all variants of the SPARK Toolset Release 9.0.0 compared to Release 8.1.1.

2 SPARK language changes

This section describes SPARK language changes supported by Toolset Release 9.0.0 since release 8.1.1.

2.1 SPARK 2005

2.1.1 SPARK2005 Reserved Words (SEPR 2778)

The language profile for SPARK2005 (Examiner switch -language=2005) now supports the use of the Ada2005 reserved words `interface`, `overriding` and `synchronized`. In SPARK2005 mode, these are treated as reserved words. In SPARK95 mode, these are treated as identifiers, but a warning is issued as before. In SPARK83 mode, these are treated as identifiers and no warning is issued.

2.1.2 Functional Attribute S'Mod (SEPR 2786)

The language profile for SPARK2005 implements the functional attribute `S'Mod` that was introduced in Ada2005. See the Ada2005 LRM section 3.5.4 (16) for more details.

The expression `S'Mod (X)` is expanded by the VC Generator into the equivalent FDL `X mod S'Modulus.`

2.1.3 Functional Attribute S'Machine_Rounding (SEPR 2789)

In SPARK 2005 mode, the Examiner now accepts the attribute `Machine_Rounding` for floating point types. However, similar to the attribute `Rounding`, the Examiner will raise semantic error 30 – "attribute Machine_Rounding is not yet implemented in the Examiner".

2.1.4 Optional Overriding_Indicator (IB04-013)

In SPARK 2005 mode, a subprogram specification of a type extension (of a tagged type) is subject to the following rules:

- The subprogram must be prefixed with the reserved word **overriding** if it is an inherited operation.
- The subprogram specification is optionally prefixed with the reserved words **not overriding** if it is not an inherited operation – that is, not overriding is semantically equivalent to the absence of an overriding indicator. Note that the semantics for SPARK 2005 differ from Ada 2005. In Ada 2005, the absence of an overriding indicator means that the subprogram either "overrides" or "not overrides" an operation.

2.2 Checking safety and security policies using flow analysis (IC15-003)

Release 9 of the Examiner adds support for the checking of specific information-flow policies, using the Integrity property of an own variable to “label” the integrity of an input, output, or state variable. These labels and policies, for example, allow the Examiner to enforce the rule that a “Top Secret” state or input should not affect an “Unclassified” state or output.

Two policies are predefined called “security” and “safety” – these are decribed in section 8.3 of the Examiner User Manual.

2.3 Allow labels on statements in SPARK (SEPR 2703)

SPARK now permits statements to be labelled, as in Ada, for purposes such as instructing a debugger when to break. For example:

```
           procedure Increment (X : in out Integer)
           --# derives X from X;
           is
              T : Integer; -- not a statement so label not allowed
           begin
<<L1>>        T := 1;
<<L2>>        if X < Integer'Last then
<<L3>>           X := X + T;
              else         -- label not permitted here
<<L4>>           null;
              end if;      -- label not permitted here
           end Increment;
```

Note that labels are also permitted to appear before pragmas, but declarations and SPARK annotations may not be labelled.

For more information on labels please refer to the Ada Language Reference Manual section 5.1 and the SPARK Language Definition (docs\PDF\SPARK95.pdf under your SPARK Toolset installation directory).

2.4 Allow use of ‘others’ to initialize unconstrained array (SEPR 2723)

SPARK now permits the use of an unqualified aggregate assignment to initialize an unconstrained array parameter, provided that:

- the aggregate only specifies a value for **others**;
- the array is one-dimensional (ie multi-dimensional unconstrained arrays cannot be initialized in this way at present).

For example:

```
type A is array (Integer range <>) of Integer;

procedure Init_Array (X : out A)
--# derives X from ;
is
begin
   X := (others => 0);
end InitArray;
```

Note that the use of an unconstrained array type name to qualify an aggregate is not permitted in Ada, but it may now be used in SPARK annotations so that a suitable postcondition can be written for the example above, thus:

```
procedure Init_Array (X : out A)
--# derives X from ;
--# post X = A'(others => 0);
```

3 TN IA02-023 – Important Information for Users

3.1 Description

TN IA02-023 concerns a case where the Examiner can generate an incorrect verification condition for a subprogram that employs refined pre- and postconditions for a private type.

The incorrect rules mean that the Simplifier and POGS may report all VCs proved for that subprogram and calling subprograms, even though a potential run-time error does exist.

The exact circumstances for this problem to occur are as follows:

- The subprogram has a parameter which is a private type in the visible part of the enclosing package body.
- The subprogram has both abstract and concrete pre- and post-conditions.
- The enclosing package does *not* employ own-variable state refinement, so there is only one set of global and derives annotations for it.

In this case, the refinement VC for the postcondition is essentially that "concrete postcondition -> abstract postcondition." This is expanded out by the Examiner into a VC of the form:

```
Abstract Precondition and
Concrete Precondition and
Concrete Postcondition
        ->
Abstract Postcondition .
```

The problem lies with the Concrete Precondition section. Here, variables should be decorated with the "~" suffix to indicate the initial value of that variable. Examiners 7.2 through 8.1.4 omit this suffix in the conditions listed above, leading to an incorrect VC. Section 3.3 below shows how this error could lead to a potentially incorrect analysis.

3.2 SPARK Examiner versions affected

This problem affects Examiner versions 7.2 (December 2004) through 8.1.4 (October 2009). In versions of the Examiner earlier than 7.2, refined pre- and post-conditions for packages with private types were not permitted.

3.3 Test Case

The following code illustrates this problem.

```
package Test
is
   type T  is private;
   type T2 is private;
   CT  : constant T;
   CT2 : constant T2;

   subtype S is Integer range 0 .. 10;
   subtype S2 is S range 0 .. 5;

   --# function In_Range (X : T) return Boolean;
   --# function Updated (A : T2; X : T) return Boolean;
   procedure P2 (A : in out T2; X : in T; Y : out S);
   --# derives A from A, X & Y from ;
   --# pre In_Range (X);
   --# post Updated (A, X) and
   --#      Y in S2;

private
   type T is range 0 .. 101;
   subtype Index is T range 1 ..10;
   type T2 is array (Index) of T;

   CT  : constant T := 1;
   CT2 : constant T2 := T2'(others => 1);
end Test;
```

Note the abstract postcondition on P2 includes “Y in S2” meaning that the exported value of Y must lie in the range 0 through 5. The body of the package is as follows:

```
package body Test is
   procedure P2 (A : in out T2; X : in T; Y : out S)
   --# pre X in Index and
   --#     X /= A (X);
   --# post A = A~[X => X];
   is
   begin
      A (X) := X;
      Y := 6;
   end P2;
end Test;
```

Note that the final value of Y is 6, meaning that this code does satisfy the concrete postcondition, but does *not* satisfy the abstract postcondition stated in the package's specification.

The refinement VC generated for the postcondition with Examiner 8.1.4 is as follows:

```
procedure_p2_5.
H1:    in_range(x) .
H2:    x >= index__first .
H3:    x <= index__last .
H4:    x <> element(a, [x]) .
H5:    for_all(i___1: t, ((i___1 >= index__first) and (
           i___1 <= index__last)) -> ((element(a, [i___1]) >=
           t__first) and (element(a, [i___1]) <= t__last))) .
H6:    x >= t__first .
H7:    x <= t__last .
H8:    a = update(a~, [x], x) .
        ->
C1:    updated(a, x) .
C2:    y >= s2__first .
C3:    y <= s2__last .
```

Note that C2 and C3 are unprovable as expected. Unfortunately, H4 is incorrectly generated – the "a" should be "a~". Furthermore, H4 and H8 then form a contradiction since they reduce to x <> x. This allows this VC to be incorrectly proved-by-contradiction by the Simplifier.

3.4 Corrected Behaviour

Examiner 9.0.0 generates the following postcondition refinement VC for the same code:

```
H1:    in_range(x) .
H2:    x >= index__first .
H3:    x <= index__last .
H4:    x <> element(a~, [x]) .
H5:    for_all(i___1: t, ((i___1 >= index__first) and (
           i___1 <= index__last)) -> ((element(a~, [i___1]) >=
           t__first) and (element(a~, [i___1]) <= t__last))) .
H6:    x >= t__first .
H7:    x <= t__last .
H8:    for_all(i___1: t, ((i___1 >= index__first) and (
           i___1 <= index__last)) -> ((element(a, [i___1]) >=
           t__first) and (element(a, [i___1]) <= t__last))) .
H9:    y >= s__first .
```

```
H10:   y <= s__last .
H11:   a = update(a~, [x], x) .
        ->
C1:    updated(a, x) .
C2:    y >= s2__first .
C3:    y <= s2__last .
```

Note that H4 is now correct, and the correct range of Y is reflected in H9 and H10. The Simplifier reduces this to an unprovable VC – the significant fraction of which is:

```
H7:    y >= 0 .
H8:    y <= 10 .
       ->
C1:    updated(update(a~, [x], x), x) .
C2:    y <= 5 .
```

clearly showing how the final value of Y does not imply the abstract postcondition.

3.5 Recommended Action

For projects using Examiner 7.2 through 8.1.4, we recommend a complete "back-to-back" regression proof to check for any differences arising from this TN. We also recommend that projects should check for any use of refined pre- and post-conditions in packages that declare private types. If this language feature is not used, then this problem cannot occur.

Please contact Praxis for further advice regarding this problem.

4 Examiner changes

This section documents other significant changes to the SPARK Examiner made for release 9.0.0. Where appropriate, GNAT Tracker Ticket Numbers (TNs) are given.

4.1 Allow labels on statements [I521-014]

As described in section 2.1 the SPARK language now permits labels, so the SPARK Examiner has been updated accordingly to recognise and accept labelled statements. It should be noted that the Examiner checks the syntax of labels but otherwise ignores them as they have no effect on the semantics of the SPARK program. As such, it does not perform any semantic checks – for example it will not check whether a label's identifier clashes with other identifiers in scope. These checks will be performed by an Ada compiler.

4.2 Allow proof refinement where applicable [I309-006]

Previously, it was not possible to perform proof refinement of functions where the sole justification for the refinement was the function returning a private type. Further, where using subunits, it was never possible to perform proof refinement of functions. These two cases are resolved, such that proof refinement is now possible if it is applicable.

4.3 Output Directory in report file [I610-021]

The Examiner's report file now includes both the requested and actual output directory. When the –plain switch is active, the requested directory is repeated in both cases to avoid spurious differences in regression analysis. For example, the table below shows the output in the report file for running the Examiner with the given options, assuming that current working directory is d:\sparkdev\examiner

Options	Report file output

	`output_directory (requested)=.`
	`output_directory (actual)=D:\sparkdev\examiner\`

`-plain`	`output_directory (requested)=.`
	`output_directory (actual)=.`

`-output=..`	`output_directory (requested)=..`
	`output_directory (actual)=D:\sparkdev\`

`-plain -output=..`	`output_directory (requested)=..`
	`output_directory (actual)=..`

4.4 Allow use of ‘others’ to initialize unconstrained array [I409-012]

The Examiner has been updated in line with the SPARK language change described in section 2.4.

4.5 Control of path-names in “brief” output mode [I604-011]

The “-brief” switch has been extended to allow control over the path-names generated in error and warning messages.

The brief switch may now take an optional value which must be “nopath” or “fullpath”.

“nopath” corresponds with the behaviour of “-brief” in all existing Examiner versions prior to 8.1.4 and remains the default. In this mode, the Examiner reports the base-name of the offending file without any path or directory information.

When “fullpath” is selected, the Examiner includes the full path-name of the offending file in any error or warning messages. This is useful when a project has multiple source directories and not all source files are therefore in the current working directory.

“-brief” alone is equivalent to “-brief=nopath”.

4.6 Index file cache [I220-023]

An index file cache has been introduced to speed up the generation of cross-references within annotations for future integration into IDEs. The operation of the cache means that the behaviour of the Examiner when reading index files may be different. The new behaviour is described in the Examiner User Manual.

In particular it should be noted that the behaviour has changed in the following ways:

1 The order of auxindex entries within a file is now immaterial.

2 If a unit cannot be found within the auxiliary index files associated with an auxindex entry with a matching prefix the search will continue and if necessary proceed with searching the super index file if one is specified.

The different behaviour means that the following entries in an index file are now permissible:

```
new_behaviour.idx
superindex is in super.idx
P auxindex is in p_aux.idx         -- Should not contain P.A.B specification
P.A auxindex is in p-a_aux.idx     -- Should not contain P.A.B specification
P.A.B specification is in p-a-b.ads
```

If the required unit is P.A.B specification then this will be found in index file new_behaviour.idx. If the required unit is P.A.C and it is defined in P.A auxindex but not in P auxindex then the unit will be found in p-a_aux.idx. If the required unit is P.A.D and this is not defined in either P auxindex or P.A auxindex then the search will continue in the super index file super.idx.

The other difference in behaviour is in error handling and the issuing of warnings. The Examiner will terminate immediately with an error message if an error is encountered in an index file. It will issue a new warning (suppressible using the warning control file keyword **index_manager_duplicates**) if two identical entries exist within the index file hierarchy.

There should be no noticeable change in behaviour using index files which satisfy the previous rules.

4.7 New switch to select language profile [I717-014]

A new switch allows selection of the sequential language profile. It takes the form:

```
-language=[83|95|2005]
```

This switch replaces the existing "-ada83" switch, which is now considered obsolete. We recommend that all users upgrade their default switch files, project files, and scripts to use the new switch.

In combination with the "-profile" switch, this yields five legal language profiles: sequential SPARK83, sequential SPARK95, sequential SPARK2005, RavenSPARK95, and RavenSPARK2005. Attempting to combine SPARK83 and RavenSPARK is illegal and is rejected by the Examiner.

Note: currently SPARK2005 mode is identical to SPARK95 mode. Future enhancements to bring SPARK2005-only language features will follow in due course.

4.8 Stronger default "for" loop invariant [I715-017]

The VC Generator has been strengthened to generate an appropriate constraint on the initial value of a "for" loop parameter, where the iteration scheme has an explicit range constraint. For example, consider the following package:

```
package Loops is
   subtype R is Integer range 0 .. 5;
   type RArray (R) of Integer;

   procedure P2 (A     : in out RArray;
                 LowB  : in     R;
                 HighB : in     R);
   --# derives A from A, LowB, HighB;
end Loops;
```

```
package body Loops is
   procedure P2 (A     : in out RArray;
                 LowB  : in     R;
                 HighB : in     R)
   is
   begin
      for I in Integer range LowB .. HighB loop
         A (I) := 4;
      end loop;
   end P2;
end Loops;
```

Examiner 8.1.1 generated the following default loop invariant:

```
--# assert (for all J in R => A~ (J) in Integer) and
--#        LowB in R and
--#        HighB in R and
--#        I in Integer and
--#        I <= HighB;
```

This loop invariant is not sufficiently strong to prove the run-time check VC arising from the assignment statement inside the loop body.

Examiner versions 8.1.4 and above correct this problem, adding a single conjunct to the loop invariant that constrains I relative to its initial value LowB. This generates:

```
--# assert (for all J in R => A~ (J) in Integer) and
--#        LowB in R and
--#        HighB in R and
--#        I in Integer and
--#        I >= LowB and
--#        I <= HighB;
```

and all VCs are automatically discharged by the Simplifier. The same improvement applies to “for” loops with an “in reverse” iteration scheme where the right hand side of the range constraint is used as the initial value of the loop parameter.

4.9 Proof rules for record type equality [I803-018]

The VC Generator has been improved to generate an inference rule for equality of record types. This appears in the RLS file, and can be exploited by Simplifier 8.1.4 and above to prove VCs involving equality between record objects, updates of those objects, and aggregate expressions.

Consider the following record type and subprogram declaration:

```
type R is record
   F1 : Integer;
   F2 : Boolean;
end record;

procedure Init (X : out R);
--# derives X from ;
--# post X = R'(F1 => 0, F2 => False);
```

The body of Init is implemented thus:

```
procedure Init (X : out R) is
   T : R;
begin
   T.F1 := 0;
   T.F2 := False;
   X := T;
end Init;
```

which results in the following VC conclusion:

```
C1:  upf_f2(upf_f1(t, 0), false) =
     mk__r(f1 := 0, f2 := false) .
```

Simplifier 8.1.3 was not capable of proving this conclusion automatically.

Examiner 8.1.4 and above now generate an additional proof rule in the RLS file:

```
A = B may_be_deduced_from
     [goal(checktype(A,r)),
      goal(checktype(B,r)),
      fld_f1(A) = fld_f1(B),
      fld_f2(A) = fld_f2(B)].
```

This rule is effectively an instantiation of the more general Proof Checker rule family RECORD_EQUALITY (see Proof Checker Rules Manual section 4.13) for type R.

Simplifier 8.1.4 and above are able to take advantage of such rules to prove conclusions like the one shown above automatically.

4.10 Improved error messages for undeclared operators [I930-014]

Consider the following code:

```
type T is mod 2**32;

procedure Op (X, Y : in     T;
              Z    :    out T);
--# derives Z from X, Y;
--# post Z = X or Y;
```

The postcondition parses as (Z = X) or Y owing to the lower precedence of “or”, so this expression is illegal since there is no “or “ operator that takes Boolean on the left and type T on the right. Examiner 8.1.4 reported:

```
--# post Z = X or Y;
              ^
*** Semantic Error: 35: Operator is not declared for these types.
```

In Examiner 9.0.0, this error message has been improved to indicate the types found, thus:

```
--# post Z = X or Y;
              ^
*** Semantic Error: 35: Binary operator is not declared for types
BOOLEAN and T.
```

Secondly, a new semantic error 119 has been added that covers the same case for unary operators.

4.11 Removal of Path Functions [IB17-027]

The ability to generate “Path Functions” from SPARK code is now considered obsolete and has been removed from the Examiner. The “Generation of Path Function” manual is no longer distributed and all mention of path functions has been removed from other user manuals.

4.12 Removal of obsolete VC Generation switches [IC01-020]

The –rtc, -exp and –realrtcs switches have been removed from the Examiner in favour of the new –vcg switch, which is equivalent to the old combination "-exp –realrtcs". Existing scripts and project files must now be updated to use –vcg.

4.13 Casing option [I422-010]

A style check switch (-casing) has been introduced which enables case-sensitive checking of identifiers in SPARK code and annotations. When the switch is active the Examiner will issue a warning if the use of an identifier does not match the case used in its declaration. The switch takes options to enable casing checks only for identifiers declared in package Standard, identifiers declared only in the program under analysis, or both.

A related change is that the correct case is now always used when identifiers are reported in Examiner output files. Users who perform a regression analysis and compare their results against those generated by an earlier version of the SPARK Toolset are likely to see a large number of differences due to the correct casing now being used in output files.

For more information on the use of this switch see the Examiner User Manual.

4.14 Return annotations in function calls [I915-022]

Function return annotations are now translated and incorporated in the hypotheses of paths that traverse the function call. It is no longer necessary to provide a user-defined rule to specify the function within the proof tools. User-defined rules may still be required to specify properties resulting from the definition given in the return annotation.

For VC generation a function call is modelled by:

1 a check statement to show that all actual parameters are in-in type (global variables of a function are assumed to be in-type as they are checked prior to the function call) – existing behaviour;

2 a check statement to show that the pre-condition of the called function is satisfied by the actual parameters used and the current values of the called function's global variables – existing behaviour;

3 an assumption (which appears in the hypotheses of VCs for paths which traverse the function call) that the function's return annotation holds for the given actual parameters including global variables – new behaviour; and

4 an assumption that the returned value is in-type –existing behaviour.

For example the following function declaration, F1, with an explicit return annotation gives the VCs shown when it is called by function F2.

```
   function F1 (X : Integer) return Integer;
   --# pre X * X <= Integer'Last and
   --#     X * X >= 0 ;
   --# return X * X;

 6   function F2 ( Y : Integer) return Integer;
 7   --# pre Y * Y <= Integer'Last and
 8   --#     Y * Y >= 0;
 9   --# return Y * Y
10   begin
11      return F1 (Y);
12   end F2;
```

```
For path(s) from start to precondition check associated with
statement of line 11:

function_f2_1.
H1:    y * y <= integer__last .
H2:    y * y >= 0 .
H3:    y >= integer__first .
H4:    y <= integer__last .
        ->
C1:    y * y <= integer__last .
C2:    y * y >= 0 .
C3:    y >= integer__first .
C4:    y <= integer__last .


For path(s) from start to finish:

function_f2_2.
H1:    y * y <= integer__last .
H2:    y * y >= 0 .
H3:    y >= integer__first .
H4:    y <= integer__last .
H5:    y * y <= integer__last .
H6:    y * y >= 0 .
H7:    y >= integer__first .
H8:    y <= integer__last .
H9:    f1(y) >= integer__first .
```

```
H10:   f1(y) <= integer__last .
H11:   f1(y) = y * y .
        ->
C1:    f1(y) = y * y .
C2:    f1(y) >= integer__first .
C3:    f1(y) <= integer__last .
```

In VC f2_1 the conclusions C1 and C2 are derived from the check of the pre conditions of F1 and C3 from the check that the actual parameter, Y, is in-type. This VC has to be proved to show that the preconditions of F1 are satisfied by this call.

The hypotheses in VC f2_2; H3 and H4 follow from the in-type check of the actual parameter Y, H5 and H6 follow from the precondition check for the call to F1, H9 and H10 are the assumptions that the returned value from F1 is in-type and H11 is the assumption that the return annotation of F1 is satisfied.

4.15 SLI file generation [J111-010]

The SPARK Examiner now generates SLI files by default for cross navigation in an IDE (e.g. GPS or Emacs) or using the GNATFind tool. For each SPARK source file that is analysed the Examiner now generates a corresponding SLI file with the extension ".sli". A new switch '-nosli' has been added in order to suppress generation of SLI files if they are not required. The new warning "Warning :430 : SLI generation abandoned owing to syntax or semantic errors or multiple units in a single source file" will be raised if there is a syntax or a semantic error in the SPARK source code or if a file contains multiple compilation units. Once this warning has been produced no more SLI files will be generated.

4.16 Default install directory for Windows InstallShield [IB03-040]

By default the tools are installed to C:\SPARK\<version> when using the Windows InstallShield executable. (The previous default was C:\Spark\<version>.)

5 Simplifier changes

Simplifier 9.0.0 includes the following changes and improvements:

5.1 Enhanced log messages [I521-010]

More informative details of substitutions that have been carried out by the Simplifier are now included in the generated SLG file. SIV files are not affected by this change.

5.2 Increased proof coverage for partially updated composites [I424-030]

Previously, especially in the context of arrays, the Simplifier offered limited proof automation where composite structures are partially updated by a procedure. For example, given an array A, an index variable `I` and a procedure P, passing `A(I)`into an out parameter of P would partially update the composite structure A. The Simplifier now offers improved proof automation in these cases.

5.3 Simplifier uses inference rules from Examiner [I724-003]

Previously, the Simplifier ignored inference rules (e.g. may_be_deduced_from rules) generated by the Examiner in the RLS file if there was no corresponding RLU file. The Simplifier now always tries to use inference rules from the RLS file to discharge VCs.

5.4 Extended Simplifier's reasoning of simple universal quantifiers with array updates [I724-016]

Previously, the Simplifier could not discharge VCs containing simple universal quantifiers generated by the Examiner from "for" loops that update an array (e.g. initialisation of an array). The Simplifier is now able to discharge VCs containing such conclusions. For example, consider an array A, whose index and element type are both Character. If we initialise A with a "for" loop to set each element to a value that is equal to its index position:

```
for I in Character loop
   A (I) := I;
   --# assert for all J in Character range Character'First .. I =>
   --#          (A (J) = J);
end loop;
--# check for all J in Character => (A (J) = J);
```

All VCs arising from this code are now proven automatically.

5.5 Better reasoning for bounds of modular expressions [I805-018]

The Simplifier is now able to infer the lower- and upper-bound for modular expressions where the modulus is up to and including 2**64. In particular, this improves reasoning about programs that involve operations on 64-bit modular types.

5.6 Extended Simplifier's reasoning of simple universal quantifiers [I903-018]

The Simplifier is now able to reason about universal quantifiers where the range is the empty set (lower bound of the range is greater than the upper bound) or the range is a singleton element (lower and upper bounds are equal). This improves the reasoning of programs with loops updating an array.

5.7 Enhanced management of Hypothesis Identifiers [I629-007]

Refactored code that manages hypothesis identifiers. SLG and SIV are affected by the change but not log files.

Added support for *norenum* flag. The hypothesis and conclusion identifiers are not renumbered in siv files and provides better tracing of hypothesis and conclusions from a siv file to the vcg file.

This change can change the hypothesis identifier assigned to a derived hypothesis and can affect the Simplifier's search for proofs when it relies on user rules that contain free variables in side conditions. The use of free variables in side conditions is generally not recommended unless it is essential. When used, it is recommended that a more constrained expression containing the free variable precede less constrained expressions containing the free variable.

5.8 Improved readability of Simplifier verbose mode output [I727-020]

Improved output message in verbose mode.

5.9 Improved support for disjunctions [I611-014]

The Simplifier now provides improved support for reasoning of disjunctions in hypothesis and conclusions. Previously, the Simplifier was unable to discharge VCs of the form shown below.

```
H1:    x or y .
        ->
C1:    y or x .
```

5.10 Improved reasoning of empty ranges in universal quantified VCs [IA19-002]

The Simplifier now discharges all conclusions of the form

```
   for_all(I:T, LWB <= I and I <= UPB -> X)
```

where the range is the empty range – that is, UPB < LWB.

6 SPARKSimp changes

6.1 New command switches and invocation of ZombieScope [IC15-017]

SPARKSimp now finds all *.vcg* and *.dpc* files in a directory tree that require analysis and applies the Simplifier to .vcg files and ZombieScope to .dpc files. The switches –nz can be used to ignore .dpc files and –ns can be used to ignore *.vcg* files. Release 9.0.0 of SPARKSimp has the following usage (**changes introduced in release 9.0.0 are highlighted in bold**):

```
sparksimp [-a] [-v] [-V] [-n] [-ns] [-nz] [-t] [-r] [-l] [-e] [-p=N]
          [-x=Sexec] [-z=Zexec]
          [-sargs {simplifier_options}]
          [-zargs {zombiescope_options}]

-a  all - ignore time stamps, so process all eligible files
-v  report version and terminate
-V  verbose output
-n  dry run - print commands but don't run Simplifier or ZombieScope
-ns ignore VCG files when searching
-nz ignore DPC files when searching
-t  sort VCG files, largest first
-r  reverse simplification order
-l  log output for XXX.vcg to XXX.log and for YYY.dpc to YYY.zsl
-e  echo Simplifier output to screen
-p=N use N concurrent processes
-x=Sexec, specifies an alternative Simplifier executable
-z=Zexec, specifies an alternative ZombieScope executable
-sargs pass following options in this section to Simplifier
-zargs pass following options in this section to ZombieScope
```

7 Proof Checker changes

Proof Checker 9.0.0 includes the following changes and improvements:

7.1 New “replace_more” flag [I820-002]

Added flag “replace_more”. The flag determines if the user is prompted for more replacements after a *replace* command. The default value is *off.* For backward compatibility, set the flag to *on*. (This can be done via the `set` command as described in the Proof Checker User Manual section 5.2.6.)

7.2 Improved reasoning of “false” hypothesis [I820-003]

Added flag “auto_infer_from_false”. When the flag is set to *on* then the *done* command will discharge a VC if one of the hypothesis is *false*. The default value is *on.* For backward compatibility, set the flag to *off.* (This can be done via the `set` command as described in the Proof Checker User Manual section 5.2.6.)

8 POGS changes

POGS 9.0.0 includes the following changes and improvements:

8.1 Preserve casing when reporting files used [I605-005]

This change affects POGS on Windows platforms only. Previously, on Windows, POGS would normalise the paths and filenames of all files to lower case in its output. On other platforms it preserved the original casing. This normalisation on Windows has now been removed so filename casing is preserved consistently across all platforms.

8.2 Extended POGS to process ZombieScope files and enhanced POGS output [IB05-012]

POGS has been extended to process ZombieScope files (.dpc and .sdp) and report dead paths analysis. The header and the per-subprogram analysis sections of POGS have been updated to improve the readability and searchability of POGS Summary files. For full details, refer to the POGS user manual.

9 SPARKFormat changes

SPARKFormat version 9.0.0 includes the following changes and improvements:

9.1 Remove “-order=declaration” option [J121-025]

The option "-order=declaration" has been removed so annotations are now always output in alphabetic order after reformatting. The option "-order=alphabetic" is still supported but is no longer required as it is the default behaviour.

10 SPARKMake changes

SPARKMake 9.0.0 includes the following changes and improvements:

10.1 Language selection switch [IA16-037]

A new switch allows selection of the sequential language profile. It takes the form:

```
-language=[83|95|2005]
```

The default is SPARK95. The language switch is required if your program is written in SPARK83 or SPARK2005 and uses features that would cause it to be rejected by SPARKMake when running in SPARK95 mode. For more information see the SPARKMake User Manual.

11 ZombieScope Changes

ZombieScope 9.0.0 includes the following changes and improvements:

11.1 Distribution of ZombieScope

Inclusion of the ZombieScope tool in SPARK Pro 9.0.0. ZombieScope detects dead paths in SPARK source code – see ZombieScope user manual for further information.

12 Backward incompatibilities

The following backward incompatibilities were introduced between Examiner Releases 8.1.1 and 9.0.0.

12.1 Checker "Replace_more" prompt [I820-002]

The change can break existing Checker command scripts and they will need to be updated.

The prompt "`Replace more`" after a `replace` command has been removed. For backward compatibility, set the "`replace_more`" flag to *on.* (This can be done via the `set` command as described in the Proof Checker User Manual section 5.2.6.)

12.2 Improved reasoning of "false" hypothesis [I820-003]

This change can break existing Checker command scripts and they will need to be updated. The easiest solution is to set the "auto_infer_from_false" flag to *off.* (This can be done via the `set` command as described in the Proof Checker User Manual section 5.2.6.)

12.3 Removal of –rtc, -exp, and –realrtcs switches [IC01-020]

The Examiner switches –rtc, -exp and –realrtcs have been removed and replaced by the single –vcg switch. Existing default switch files, scripts and project files will have to be updated.

12.4 Remove "-order=declaration" option in SPARKFormat [J121-025]

The SPARKFormat switch "-order=declaration" has been removed. By default, SPARKFormat uses alphabetical order to reformat annotations.

12.5 Enhanced management of Hypothesis Identifiers [I629-007]

This change can change the hypothesis identifier assigned to a derived hypothesis and can affect the Simplifier's search for proofs when it relies on user rules that contain free variables in side conditions. The use of free variables in side conditions is generally not recommended unless it is essential. When used, it is recommended that a more constrained expression containing the free variable precede less constrained expressions containing the free variable.

13 User manual change summary

The following user manuals have been changed between Examiner Releases 8.1.1 and 9.0.0.

13.1 New Global Documentation Index

A new Global Index is supplied with the toolset documentation. This is supplied in both PDF and HTML formats, with filenames Global_Index.pdf and Global_Index.htm respectively.

This forms a one-page index for major topics covered in *all* the other manuals and documents. The PDF and HTML versions of the index contain hyperlinks that lead to the referenced manuals.

13.2 SPARK Language Definitions

The SPARK Language Definitions have been renamed and unified in this release to form two documents:

- SPARK83_LRM – this is the language reference manual for SPARK83 only.
- SPARK_LRM – this is the language reference manual for SPARK95 and SPARK2005, including both the “sequential” and “ravenscar” concurrency profiles.

13.3 Examiner User Manual

- Added warning control keyword index_manager_duplicates for suppressing warnings regarding index file duplicate entries.
- Added description of new casing switch.
- Added ‘-nosli’ switch.

13.4 Checker User Manual

The “Replace More” prompt is now controlled by a new flag **replace_more** which is *off* (disabled) by default.

A new feature has been introduced to automatically infer a conclusion from a false hypothesis. This feature is controlled by the flag **auto_infer_from_false** that is *on* (enabling the feature) by default.

13.5 Checker Rules Manual

Update copyright and licence information in listings of proof rule files.

13.6 SPARKMake User Manual

Documents the use of the new language_profile switch.

13.7 SPARKFormat User Manual

Remove “-order=declaration” option.

13.8 Simplifier

Updated section for SPARKSimp to include switches for ZombieScope, removed Path Functions and updates due to changes to the Simplifier.

13.9 ZombieScope User Manual

Added user manual for ZombieScope.

13.10 POGS User Manual

Update document for changes to POGS including the processing of ZombieScope files.

13.11 SPARK Proof Manual

This document replaces the manuals “Generation of VCs for SPARK Programs” and “Generation of RTCs for SPARK Programs”. This should reduce the potential for confusion over which document to consult on a particular topic relating to SPARK proof.

13.12 SPARK Quick Reference Guide 5

New quick reference guide of Proof Checker Commands.

13.13 SPARK Quick Reference Guide 6

New quick reference guide for the Proof Checker Rules.

13.14 INFORMED Design Method for SPARK

Corrected a cross-reference.

13.15 SPARK IO – Input/Output for SPARK Programs

Now states that SPARK_IO may be used with SPARK95 and SPARK2005.

14 Limitations and known errors

14.1 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) or GNAT Tracker Ticket (TN) number is given.

14.1.1 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 user-defined 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)

8 The use of a tagged view conversion as the target of an assignment is not implemented. (SEPR 2739)

14.1.2 Verification Condition Generation and Run-time Checks

1 Ada string inequality and ordering operators are not modelled. Their use can result in a the Simplifier reporting an FDL typecheck error. The workaround is to wrap the operator concerned in a user-defined function. (TN [IC21-017])

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 14.1.3.

14.1.3 Attribute limitations

14.1.3.1 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
- Machine_Rounding
- Model
- Rounding
- Truncation
- Unbiased_Rounding

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

14.1.3.2 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.

14.2 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)

15 Change Summary from Release 2.0

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

15.1 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.

15.2 Release 2.1 - July 1996

Release 2.1 added:

- facilities for proof of absence of run-time errors

15.3 Release 2.5 - March 1997

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

15.4 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.

15.5 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.

15.6 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

15.7 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

15.8 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

15.9 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.

15.10 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.

15.11 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.

15.12 Release 7.3 – February 2006

Release 7.3 of the toolset comprised:

Examiner version 7.3

Simplifier 2.22

Checker 2.06

Significant features of this release included:

- Improved VC Generation.
- New “error explanations” switch for Examiner.
- Generation of proof rules for ‘Size.
- Improved diagnosis and reporting of common syntax errors.

- Error and warning count summary in Examiner output.
- Use of pragma Import to complete an external own variable.
- Correction to FDL declaration order for private and announced types.
- New Simplifier tactics for dealing with rational inequalities and Examiner-generated proof rules.
- User-defined proof rules for the Simplifier.
- New Checker rules for MODULAR expressions.
- Significant performance improvement for Simplifier and Checker.

15.13 Release 7.31 – April 2006

Release 7.31 of the toolset comprised:

Examiner version 7.31

Simplifier 2.24

Checker 2.07

Significant features of this release included:

- Correction to flow analysis of array element "out" parameters.
- Port of "Demo" toolset to Mac OS X.
- New /typecheck option in Simplifier.
- Improved validity checking of user-defined proof rules in the Simplifier.
- Correction to ARITH proof rules in the Checker.

15.14 Release 7.4 – January 2007

Release 7.4 of the toolset comprised:

Examiner version 7.4

Simplifier 2.30

Checker 2.08

Significant features of this release included:

- The ‘accept’ annotation.
- Conditional flow errors are now reported as errors.
- The Simplifier supports user defined proof rules.
- SPARKFormat supports a wider range of annotations.
- SPARKMake has the ability to generate relative pathnames.

15.15 Release 7.5 – May 2007

Release 7.5 of the toolset comprised:

Examiner version 7.5

Simplifier 2.32

Checker 2.08

Significant features of this release included:

- Correct VC generation for record fields and array elements used as “in” or “in out” parameters.

15.16 Release 7.6 – June 2008

Release 7.6 of the toolset comprised:

Examiner version 7.6

Simplifier 2.39

Checker 2.11

Significant features of this release included:

- Allows single-field record type to be Atomic in RavenSPARK mode.
- Simpler modelling of Character’Pos and ‘Val in VCs.
- Better modelling of S’Valid where S is a subtype.
- New “Output_Directory” option for the Examiner.

- New “Order” option for SPARKFormat.

15.17 Release 7.6.2 – February 2009

From release 7.6.2 of the toolset the version numbers of all the SPARK tools were made consistent, ie all tools had the version number 7.6.2. This was an interim release, the significant feature being the correction to SEPR 2517. All users were informed of the availability of this release and it was made available on demand. As the majority of users did not take this release, the changes detailed in the release note for 7.6.2 were also included in the release note for 8.1.0.

15.18 Release 8.1.0 – March 2009

Significant features of this release included:

- New –vcg and –noswitch Examiner switches;
- Documentation of the Examiner’s –debug switch;
- Significantly improved Simplifier performance.
- New platforms supported in SPARK Pro: 32-bit Linux, 64-bit Linux, Apple OS X.

15.19 Release 8.1.1 – May 2009

Significant features of this release included:

- Examiner now warns if –vcg specified but no VCs generated;
- Additional proof rules for task discriminants;
- Correction to modelling of dynamic “for” loop nested inside case statement;
- FDL declarations and rules for numeric constants used in proof rules;
- New –debug=V switch for visualization of VCG graphs.

16 Operating system compatibility

16.1 VAX/VMS

The toolset is no longer available on VAX/VMS.

16.2 SPARC/Solaris

The toolset is compatible with Solaris 8 through to 10 including those with a 64-bit kernel.

n.b. Solaris 6 and 7 are no longer supported.

16.3 Windows

The toolset is compatible with Windows XP, Windows 7, Server 2003, 32-bit Vista, and 64-bit Vista.

16.4 Intel/Linux

The toolset is compatible with Intel x86-based Linux operating systems, in both 32-bit and 64-bit versions. The toolset is built, tested and packaged on RedHat Enterprise Linux 4.7. Other distributions may work, but are not formally supported.

16.5 Apple OS X

The toolset is compatible with Apple's Intel-based OS X/Darwin operating systems. The toolset is built and tested on OS X version 10.5.8.

Document Control and References

Changes history

Issue 0.1 (29th May 2009): First draft for release 8.1.2.

Issue 0.2 (2nd June 2009): Updates following review S.P0468.7.168.

Issue 0.3 (11th June 2009): Draft for changes post 8.1.2 release.

Issue 0.4 (18th June 2009): Update for SEPR 2713.

Issue 0.5 (1st July 2009): Update for SEPR 2723.

Issue 0.6 (3rd July 2009): Update for new –brief switch behaviour (SEPR 2718).

Issue 0.7 (3rd July 2009): Update for new –index file search behaviour (SEPR 2594).

Issue 0.8 (8th July 2009): Updates following review S.P0468.7.190.

Issue 0.9 (17th July 2009): Updates following review S.P0468.7.61.

Issue 1.0 (20th July 2009): Update for new –language switch. (SEPR 2738)

Issue 1.1 (28th July 2009): Updates for release 8.1.3.

Issue 1.2 (28th July 2009): Updates following review 7.200.

Issue 1.3 (29th July 2009): Add section 4.3, describing SEPR 2744.

Issue 1.4 (4th August 2009): Add section 3.8, describing SEPR 2747.

Issue 1.5 (6th August 2009): Add section 3.9, describing SEPR 2748.

Issue 1.6 (24th August 2009): Added section 5.1, describing TN820-002; and sections 5.2 and 9.2 describing TN820-003.

Issue 1.7 (28th August 2009): Added section 4.5, describing TN805-018.

Issue 1.8 (11th September 2009): Added sections 5.1, 5.2 and 9 describing TN820-003.

Issue 1.9 (14th September 2009): Finalised release number as 8.1.4 for release candidate 1.

Issue 1.10 (15th September 2009): Added section 4.6 describing TN 903-018.

Issue 1.11 (18th September 2009): Added warning control file key index_manager_duplicates. User manual changes section.

Issue 1.12 (25th September 2009): Definitive issue for 8.1.4. following review S.P0468.7.221.

Issue 2.0 (1st October 2009): Added CFR 2203, SEPR 2743, TN [I727-020] and SFR 2187, SEPR 2728, TN [I629-007]

Issue 2.1 (2nd October 2009): Added CFR 2233, SEPR 2772, TN [I930-014].

Issue 2.2 (6th October 2009): Added CFR 2238, SEPR 2777, TN [IA05-014].

Issue 2.3 (7th October 2009): Added CFR 2173, SEPR 2537, TN [I611-014].

Issue 2.4 (13th October 2009): Added CFR 2239, SEPR 2778, TN[IA06-016].

Issue 2.5 (19th October 2009): Added CFR 2246, SEPR 2785, TN[IA19-002].

Issue 2.6 (30th October 2009): Added CFR 2241, SEPR 2780, TN[IA02-023].

Issue 2.7 (2nd November 2009): Added CFR 2247, SEPR 2786, TN[IA19-015].

Issue 2.8 (3rd November 2009): Added CFR 2250, SEPR 2789, TN[IB03-014].

Issue 2.9 (17th November 2009): Added section 3 regarding SEPR 2780.

Issue 3.0 (20th November 2009): Remove Path Functions option, TN[IB17-027].

Issue 3.1 (1st December 2009): Remove Examiner –rtc, -exp and –realrtcs switches. TN[IC01-020].

Issue 3.2 (18th December 2009): Added TN[IB04-013].

Issue 3.3 (22nd December 2009): Changed SEPR 712 to TN [IC21-017].

Issue 3.4 (12th January 2010): Added section describing new SPARK Proof Manual [I929-007].

Issue 3.5 (17th January 2010): Added the SPARKMake language switch [IA06-037].

Issue 3.6 (18th January 2010): Casing of the SPARK Examiner output [I422-010].

Issue 3.7 (21st January 2010): Added return annotations in function calls [I915-022].

Issue 3.8 (28th January 2010): Add Global Index [J113-005].

Issue 3.9 (28th January 2010): Remove "-order=declaration" option [J121-025] and add '-nosli' option [J111-010].

Issue 3.10 (3rd February 2010): Default installation directory [IB03-040]. Rebrand to Altran Praxis Limited

Issue 3.11 (8th February 2010): Update title of SPARK IO document [J204-021].

Issue 3.12 (11th February 2010): Minor correction to [I915-022].

Issue 3.13 (12th February 2010): Added section 2.2 on information flow policy checking [IC15-003].

Issue 3.14 (18th February 2010): Update Release Note [J218-022] to document backward incompatibility introduced by [I629-007] – extended section 5.7 and added section 10.5.

Issue 3.15 (19th February 2010): Minor corrections to sections 5.7 and 10.5 following review.

Issue 3.16 (22nd February 2010): Added section for SPARKSimp.

Issue 3.17 (25th February 2010): Added section for ZombieScope and updates for release 9.0.0.

Issue 3.18 (1st March 2010): Replace SEPR references with TN references where possible.

Issue 3.19 (1st March 2010): Definitive issue for release 9.0.0 following review.

Changes forecast

None.

Document references

None.

File under

SVN:trunk/userdocs/Release_Note_9.doc