AdaControl Specific Rules User Guide

Last edited: 21 January 2019

AdaControl is Copyright © 2005-2019 Eurocontrol/Adalog. AdaControl is free software; you can redistribute it and/or modify it under terms of the GNU General Public License as published by the Free Software Foundation; either version 2, or (at your option) any later version. This unit is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details. You should have received a copy of the GNU General Public License distributed with this program; see file COPYING. If not, write to the Free Software Foundation, 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

As a special exception, if other files instantiate generics from this program, or if you link units from this program with other files to produce an executable, this does not by itself cause the resulting executable to be covered by the GNU General Public License. This exception does not however invalidate any other reasons why the executable file might be covered by the GNU Public License.

This document is Copyright © 2005-2019 Ansaldo/Adalog. This document may be copied, in whole or in part, in any form or by any means, as is or with alterations, provided that (1) alterations are clearly marked as alterations and (2) this copyright notice is included unmodified in any copy.

Table of Contents

1 Installation of the specific rules 2

2 Rules Usage 3

- 2.1 Cte_Vmc 3
  - 2.1.1 Syntax 3
  - 2.1.2 Action 3
  - 2.1.3 Variables 5
  - 2.1.4 Tips 6
- 2.2 Filtered_Address_Clauses 6
  - 2.2.1 Syntax 6
  - 2.2.2 Action 6
  - 2.2.3 Tip 6
- 2.3 KPR_Incorrect_Size 6
  - 2.3.1 Syntax 6
  - 2.3.2 Action 7
- 2.4 KPR_Packing 7
  - 2.4.1 Syntax 7
  - 2.4.2 Action 7

1 Installation of the specific rules

The distribution of AdaControl provided by Adalog to Ansaldo STS France contains the specific rules already prepared for installation. Just follow the instruction in the AdaControl User Guide for compiling AdaControl. Note that if you install GPS support as described in the user's guide, you will also have access to this guide from the "`Help/AdaControl/Ansaldo User Guide`" menu.

If you want to upgrade to a newer version of AdaControl while retaining the specific rules, just download and unpack AdaControl as usual, then copy into the `src/` directory the file `framework-specific_plugs.adb`, `framework-specific_plugs.ads`, and the files that correspond to the specific rules (files whose name is of the form `rules-<rule name>.ad[sb]`, where <rule name> is the name of one of the rules described in this guide. Then compile AdaControl normally.

2 Rules Usage

The rules described in this chapter are specific to Ansaldo STS France, and are not part of the general distribution of AdaControl. Other rules are described in the AdaControl User Guide

2.1 Cte_Vmc

2.1.1 Syntax

```
<check|search|count> cte_vmc (<subrule> {, <target> [, "<pattern>"]});
<subrule> ::= info | incorrect | possible | gap
<target>  ::= cte_type | vmc_type | cte | vmc
```

2.1.2 Action

This rule controls various aspects of CTE and VMC types and objects, depending on the provided subrule.

The `<target>` parameter defines which elements are controlled. Only "cte_type" and "vmc_type" can be specified for the "gap" subrule, and no `<target>` parameter is allowed for the "possible" subrule, since this subrule applies only to CTE objects. In the absence of `<target>`, all applicable targets are controlled.

If a `<target>` is followed by a pattern, it specifies the naming convention for the corresponding target. By default, the naming convention is that names should start with "T_Cte" for `cte_type`, "T_Ctx" for `vmc_type`, "G_Cte" for `cte`, and "G_Ctx" for `vmc` (as defined in GID21). If a `<target>` is not to follow any naming convention, give an empty string as the pattern. See the AdaControl User Guide for the definition of patterns and pattern matching.

The detailed behaviour of each subrule is as follows:

- "info": provides information about all CTE/VMC types and objects recognized. The message includes:
  - the "class" of the type of the variable, i.e. one of "discrete", "array", "record", "access", or "address";
  - the applicable size clause, with its value (or "none" if it has been omitted);
  - the usage of the element, i.e. "not read" for objects that are never read, "externally initialized" for objects not initialized from Ada code, and "not used" for types not used for declaring objects.

    This subrule is useful, for example, to build dictionnaries of CTE/VMC objects or types.
- "incorrect": reports violations of the programming rules for any recognized CTE/VMC type or object. There is a message for each CTE/VMC that reports all violations; it includes one or more of the following elements:
  - "CTE type", "VMC type", "CTE object", "VMC object": the kind of element that has been recognized by AdaControl;
  - "(in instance)": this appears if the CTE/VMC is created by an instantiation (and is therefore not visible from the program text). The message itself appears at the place of the instantiation;

- the (full) name of the element;
- “incorrect naming”: if the element does not obey the naming convention (name should start with “T_Cte” for a CTE type, “T_Ctx” for a VMC type, “G_Cte” for a CTE object, and “G_Ctx for a VMC object);
- “not a composite type”: if the element is not an array or record;
- “not global object”: if a CTE/VMC object is local to a subprogram;
- “size clause on object”: if a size clause has been given for a CTE/VMC object (should have been given on the type);
- “no size clause”: if no size clause has been given for a CTE/VMC type. In this case, another message indicates the expected size as computed from the representation clause;
- “unable to evaluate size clause”: if a size clause has been given for a CTE/VMC type, but AdaControl is not able to evaluate it. The size clause should be checked manually;
- “size clause too small”: if a size clause has been given for a CTE/VMC type, but it is less than 32 bits. The value of the size clause is given;
- “size clause not multiple of 8” if a size clause has been given for a CTE/VMC type, but its value is not a multiple of 8. The value of the size clause is given;
- “no record repr. clause”: if a CTE/VMC type is a record, but no record representation clause has been given;
- “gaps in representation”: if a representation clause has been given, but not all possible bits in the representation correspond to actual fields of the record. The subrule “gap” can be used to report precisely where the gaps are.
- “gap at end of array”: if a VMC/CTE type is an array, and a size clause is given which is too big. The subrule “gap” can be used to report the correct size in a message at the location of the size clause;
- “type has pragma pack”: if a pragma Pack applies to the type;
- “address clause uses VMC address”: if a variable has been recognized as a CTE, but its address has been specified through an instantiation of `Csd_Lcap_Services.Adresse_VMC`;
- “address clause uses CTE address”: if a variable has been recognized as a VMC, but its address has been specified through an instantiation of `Csd_Lcap_Services.Adresse_CTE` or `Csd_Lcap_Services.Adresse_CTE_Bis`;
- “address clause not CTE or CTE_Bis”: if a variable has been recognized as a CTE, but its address has been specified without using the appropriate functions;
- “address clause not VMC”: if a variable has been recognized as a VMC, but its address has been specified without using the appropriate functions;
- “no address clause”: if a variable has been recognized as a CTE or VMC, but no address clause has been given;
- “other VMC in same unit”: if more than one VMC variable is declared in the same compilation unit;
- “inconsistent”: if a variable obeys some rules for CTE and some rules for VMC at the same time; for example, if its name matches the rule for CTE objects, but the name of its type matches the rule for VMC types.

In addition, a message is issued for each component of a CTE/VMC type definition that references non-predefined entities outside the package where the type is defined.

- “gap”: reports declarations of CTE/VMC types where the representation clauses leave bits that do not belong to any component. This subrule is recursive, i.e. it checks recursively all types used by subcomponents of CTE/VMC types. In the usual case, the subrule will state precisely where the gaps are; there are some cases however where the check cannot be performed automatically and manual inspection is required. These are:
  - “unable to evaluate record repr. clause, check gaps manually”: if a representation clause has been given, but AdaControl is not able to evaluate some of the representation items;
  - “unable to compute expected size of array - check gaps manually”: if a VMC/CTE type is an array, and AdaControl is not able to evaluate the component’s size;
  - “has discriminated fields - check gaps manually”: if some of the (sub)components of the record are of a discriminated type. The rule makes sure that each bit in the representation clause belongs to at least one component, however there can be variants that leave unused bits. This latter case is not (currently) analyzed by the rule.
- “possible”: reports on global variables and constants that are not otherwise recognized as CTE objects, but are of a structured type and are *not* written into from non-elaboration code. Such objects are potential CTE objects with a missing or incorrect address clause.

Ex:

```
Incorrect_VMC : check cte_vmc (incorrect, vmc, vmc_type);
Incorrect_CTE : check cte_vmc (incorrect, cte, cte_type);
DANGER_GAP    : check cte_vmc (gap);
search cte_vmc (possible);
```

2.1.3 Variables

By default, the rules are checked as specified in GID21. The rule provides variables that allow to choose precisely which of the programming rules that deal with CTE/VMC are to be enforced.

Variable	**Values**	**Default**	**Effect**
Composite_Type	on/off	on	if “on”, all CTE and VMC types are required to be of a composite type.
Propagate_Type	on/off	on	if “on”, a CTE/VMC variable must be of a CTE/VMC type, and conversely, a CTE/VMC type can be used only to declare CTE/VMC variables. Note that setting this variable to “off” removes a number of consistency checks.
Single_In_Package	on/off	on	if “on”, a package may not contain more than one declaration of a VMC variable.
Local_Instantiation	on/off	on	if “on”, the instantiation of the generic function that provides the address of a CTE/VMC variable must be in the same package as the variable.

2.1.4 Tips

If a type is defined in a predefined package, it is never considered a CTE/VMC type, even if used (incorrectly) to define a CTE/VMC variable. This avoids cascaded errors if, for example, a variable is defined of type `Integer`, which would in turn mark all `Integer` variables as CTE/VMC variables.

As explained above, there are some cases where AdaControl cannot evaluate the representation clauses, although their values are static. The most common case is when a representation clause uses the `'Size` of a type which is not a predefined type and has no explicit size clause applying to it. However, this is an indication of a problem, since any type which is (directly or indirectly) part of a CTE/VMC type should have its size explicitely specified.

2.2 Filtered_Address_Clauses

2.2.1 Syntax

```
<check|search|count> Filtered_Address_Clauses [(<filter> {, <filter>})];
<filter> ::= cte | cte_bis | vmc | "<pattern>"
```

2.2.2 Action

This rule controls address clauses whose address does not originate from one of the special generics use to generate addresses for CTE/VMC or that apply to objects whose name does not match one of the provided patterns (the comparison is always case-insensitive). It is useful, for example, to forbid address clauses except to CTE/VMC objects, VL variables, etc.

The special generics are:

- “cte”: `CSD_LCAP_SERVICES.ADRESSE_CTE`
- “cte_bis”: `CSD_LCAP_SERVICES.ADRESSE_CTE_BIS`
- “vmc”: `CSD_LCAP_SERVICES.ADRESSE_VMC`

See the AdaControl User Guide for the definition of patterns and pattern matching.

Ex:

```
-- Allow all CTE, VMC, and variables that contain, start,
-- or end with "VL"
check filtered_address_clauses (cte, cte_bis, vmc, "(^|_)VL(_|$)";
```

2.2.3 Tip

For “cte”, “cte_bis”, and “vmc”, it is sufficient that the address be a direct call to an instantiation of one of the generics, or through one or several constants eventually initialized by such a call; this is unlike the rule CTE_VMC, where it is required that the address be a constant directly initialized by a call to the corresponding function.

2.3 KPR_Incorrect_Size

2.3.1 Syntax

```
<check|search|count> kpr_incorrect_size;
```

2.3.2 Action

This rule controls cases where GNAT v6.1.1 and v6.1.2 can overwrite memory due to KPR KP-611-H225-010 (also referenced as KP-612-...). This happens in assignments, when the left hand side is not packed, and the right hand side is a packed record component, and various other conditions are met. Please refer to the KPR for details.

Ex:

```
check kpr_incorrect_size;
```

2.4 KPR_Packing

2.4.1 Syntax

```
<check|search|count> kpr_packing [(<subrule>)];
<subrule> ::= inefficient | dangerous
```

2.4.2 Action

This rule controls cases where GNAT v6.1.1 and v6.1.2 can behave incorrectly. This corresponds to AdaCore's KPR KP-611-H216-004, KP-611-H115-023, and KP-611-G924-016 (also referenced as KP-612-...). These problems occur for packed data where alignment clauses cause extra padding to be added by the compiler. These problems have been fixed in subsequent versions of the compiler.

In the case of the first two KPR, the resulting structure will waste space due to insufficient packing, but the generated code is correct. The subrule "inefficient" controls declarations that may be subject to this problem.

In the case of the third KPR, which happens when the size of the data has been set by a size attribute clause, a component_size attribute clause, or component clause, the generated code is incorrect and can lead to memory overwriting. The subrule "dangerous" controls declarations that may be subject to this problem.

If the rule is specified without any subrule, both subrules are controlled.

Note that the rule is slightly pessimistic, since it is very difficult to assess the exact cases where these KPR occur. The goal is to make sure that there are no false negative. When the rule signals a problem, it does not necessarily mean that the problem will occur; more details about the compiler behaviour may be obtained by inspecting the ".rep" file generated with the -gnatR option of GNAT.

Ex:

```
check kpr_packing (dangerous);
```