## Contents

<table>
<thead>
<tr>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Editorial Policy for <em>Ada User Journal</em></td>
<td>2</td>
</tr>
<tr>
<td>Editorial</td>
<td>3</td>
</tr>
<tr>
<td>Quarterly News Digest</td>
<td>4</td>
</tr>
<tr>
<td>Conference Calendar</td>
<td>25</td>
</tr>
<tr>
<td>Forthcoming Events</td>
<td>31</td>
</tr>
<tr>
<td><strong>Special Contribution</strong></td>
<td></td>
</tr>
<tr>
<td>J. Cousins</td>
<td></td>
</tr>
<tr>
<td>“ARG Work in Progress”</td>
<td>33</td>
</tr>
<tr>
<td><strong>Article</strong></td>
<td></td>
</tr>
<tr>
<td>B. Wang, H. Gao and J. Cheng</td>
<td></td>
</tr>
<tr>
<td>“Definition-Use Net and System Dependence Net Generators for Ada 2012 Programs and their Applications”</td>
<td>37</td>
</tr>
<tr>
<td><strong>SPARK 2014 Rationale: Type Predicates, Variables that are Constant, Support for Ravenscar and Support for Type Invariants</strong></td>
<td></td>
</tr>
<tr>
<td>Y. Moy and C. Dross</td>
<td>57</td>
</tr>
<tr>
<td><strong>Ada-Europe Associate Members (National Ada Organizations)</strong></td>
<td>64</td>
</tr>
<tr>
<td><strong>Ada-Europe Sponsors</strong></td>
<td>Inside Back Cover</td>
</tr>
</tbody>
</table>
Editorial Policy for Ada User Journal

Publication
Ada User Journal — The Journal for the international Ada Community — is published by Ada-Europe. It appears four times a year, on the last days of March, June, September and December. Copy date is the last day of the month of publication.

Aims

Although the title of the Journal refers to the Ada language, related topics, such as reliable software technologies, are welcome. More information on the scope of the Journal is available on its website at www.ada-europe.org/auj.

The Journal publishes the following types of material:

- Refereed original articles on technical matters concerning Ada and related topics.
- Invited papers on Ada and the Ada standardization process.
- Proceedings of workshops and panels on topics relevant to the Journal.
- Reprints of articles published elsewhere that deserve a wider audience.
- News and miscellany of interest to the Ada community.
- Commentaries on matters relating to Ada and software engineering.
- Announcements and reports of conferences and workshops.
- Announcements regarding standards concerning Ada.
- Reviews of publications in the field of software engineering.

Further details on our approach to these are given below. More complete information is available in the website at www.ada-europe.org/auj.

Original Papers
Manuscripts should be submitted in accordance with the submission guidelines (below).

All original technical contributions are submitted to refereeing by at least two people. Names of referees will be kept confidential, but their comments will be relayed to the authors at the discretion of the Editor.

The first named author will receive a complimentary copy of the issue of the Journal in which their paper appears.

By submitting a manuscript, authors grant Ada-Europe an unlimited license to publish (and, if appropriate, republish) it, if and when the article is accepted for publication. We do not require that authors assign copyright to the Journal.

Unless the authors state explicitly otherwise, submission of an article is taken to imply that it represents original, unpublished work, not under consideration for publication elsewhere.

Proceedings and Special Issues
The Ada User Journal is open to consider the publication of proceedings of workshops or panels related to the Journal’s aims and scope, as well as Special Issues on relevant topics.

Interested proponents are invited to contact the Editor-in-Chief.

News and Product Announcements
Ada User Journal is one of the ways in which people find out what is going on in the Ada community. Our readers need not surf the web or news groups to find out what is going on in the Ada world and in the neighbouring and/or competing communities. We will reprint or report on items that may be of interest to them.

Reprinted Articles
While original material is our first priority, we are willing to reprint (with the permission of the copyright holder) material previously submitted elsewhere if it is appropriate to give it a wider audience. This includes papers published in North America that are not easily available in Europe.

We have a reciprocal approach in granting permission for other publications to reprint papers originally published in Ada User Journal.

Commentaries
We publish commentaries on Ada and software engineering topics. These may represent the views either of individuals or of organisations. Such articles can be of any length – inclusion is at the discretion of the Editor.

Opinions expressed within the Ada User Journal do not necessarily represent the views of the Editor, Ada-Europe or its directors.

Announcements and Reports
We are happy to publicise and report on events that may be of interest to our readers.

Reviews
Inclusion of any review in the Journal is at the discretion of the Editor. A reviewer will be selected by the Editor to review any book or other publication sent to us. We are also prepared to print reviews submitted from elsewhere at the discretion of the Editor.

Submission Guidelines
All material for publication should be sent electronically. Authors are invited to contact the Editor-in-Chief by electronic mail to determine the best format for submission. The language of the journal is English.

Our refereeing process aims to be rapid. Currently, accepted papers submitted electronically are typically published 3-6 months after submission. Items of topical interest will normally appear in the next edition. There is no limitation on the length of papers, though a paper longer than 10,000 words would be regarded as exceptional.
Editorial

One year ago, in the editorial for the March issue of 2016, I pointed out to the readers the, at that time, recent update to the Ada 2012 standard. That issue of the journal included a special contribution which briefly described the main changes, clarifications and corrections in that update. Nevertheless, the work on the language never stops, and it is intention of the Ada User Journal to keep readers aware of the standardisation process, what is being considered and pipelined for future evolution.

We are therefore very happy to be able to publish in this issue a paper on the work in progress in the Ada Rapporteur Group (ARG), the WG9 group responsible for the language interpretation and evolution, written by the Chair of the ARG, Jeff Cousins, from BAE Systems Surface Ships Limited, UK. This paper provides information on the Ada Issues which have been already addressed and approved this past year, but also some information on the work in the pipeline. An interesting read for sure.

Continuing with the technical contributions, the issue then provides an extensive technical work from a group of authors of the Saitama University, Japan, updating previous work on definition-use and system dependence net graphs to consider new features of Ada 2012. Finally, a set of contributions from the SPARK 2014 Rationale, by Yannick Moy and Claire Dross, of AdaCore, France.

As usual, the reader will find the valuable information of the News and Calendar sections, contributed by Jacob Sparre Andersen and Dirk Craeynest, their respective editors. I would also like to draw your attention to the Ada-Europe 2017 conference, which, apart from the rich content of tutorials, exhibition and scientific and technical presentations will also provide a very rich networking environment. A particular highlight this year is the panel discussion on “The Future of Safety-Minded Languages” – which will feature interesting and potential controversial discussions. I hope to see you there!

Luís Miguel Pinho
Porto
March 2017
Email: AUJ_Editor@Ada-Europe.org
Ada-related Events

[To give an idea about the many Ada-related events organised by local groups, some information is included here. If you are organising such an event feel free to inform us as soon as possible. If you attended one please consider writing a small report for the Ada User Journal. —sparre]

“Make with Ada” Winners

From: Olivier Henley
<olivier.henley@gmail.com>
Date: Fri, 2 Dec 2016 06:55:23 -0800
Subject: [Spark2014-discuss] [Save the date] 2017/05/30 Frama-C & SPARK Day
Newsgroups: fr.comp.lang.ada, comp.lang.misc

I don't know when results became public [...] but the winners of the Make with Ada initiative is great and I will kick myself to participate next time.

[See also “Make with Ada” Programming Competition", AUJ 37-2, p. 69. —sparre]

Ada-Europe 2017 in Vienna

From: Dirk Craeynest
<d Dirk@c.kuleuven.be>
Date: Thu, 19 Jan 2017 21:47:34 -0000
Subject: FINAL CfP Ada-Europe 2017, Sun 22 Jan submission deadline
Newsgroups: comp.lang.ada, comp.lang.misc

FINAL Call for Papers
22nd International Conference on Reliable Software Technologies - Ada-Europe 2017
12-16 June 2017, Vienna, Austria
http://www.ada-europe.org/conference2017

Organized by TU Vienna on behalf of Ada-Europe, in cooperation with ACM SIGAda, SIGBED, SIGPLAN and the Ada Resource Association (ARA)
The 22nd International Conference on Reliable Software Technologies - Ada-Europe 2017 will take place in Vienna, Austria. Following its traditional style, the conference will span a full week, including a three-day technical program and vendor exhibition from Tuesday to Thursday, along with parallel tutorials and workshops on Monday and Friday. This edition features a focused Special Session on Reliable and Safe Robotics.

*** DEADLINE Sunday 22 JANUARY 2017 ***

Regular & Special Session Papers + Industrial Presentations: submit via https://easychair.org/conferences/?conf=a daeurope2017
Tutorials & Workshops: submit to the Tutorial & Workshop Chair Ben Brosgol <brosgol at adacore.com>
For more information please see the full Call for Contributions at http://www.ada-europe.org/conference2017

Frama-C and SPARK Day in Paris

From: Claude Marché
<Claude.Marche@inria.fr>
Date: Thu, 26 Jan 2017 13:25:03 +0100
Subject: [Spark2014-discuss] [Save the date] 2017/05/30 Frama-C & SPARK Day: Formal Analysis and Proof for Programs in C and Ada
To: framac-discuss@lists.gforge.inria.fr, spark2014-discuss@lists.gforge.open-do.org, Why3 Club <why3-club@lists.gforge.inria.fr>

Date: Tuesday, May 30th, 2017
Location: Paris (Université Paris-Diderot, Amphithéâtre Buffon, 15 rue Hélène Brion)

This one-day workshop aims at gathering both academic and industrial users of the environments Frama-C and SPARK, for sharing experiences and discussing perspectives. It is co-organized by CEA List (http://www-list.cea.fr/en/), AdaCore (http://www.adacore.com/), Inria joint lab 'ProofInUse' (http://www.spark-2014.org/proofinuse), and Université Paris-Diderot.

This workshop will take place in the context of the event ‘Open Source Innovation Spring 2017’ (http://www.open-source-innovation-spring.org/) initiated by thematic group ‘Logiciel libre’ of the cluster Systematic-Paris-Region and IRILL (‘Initiative de Recherche et Innovation sur le Logiciel Libre’).

Claude Marché | tel: +33 1 69 15 66 08

Ada-related Tools

Bare Bones

From: Luke A. Guest
<laguest@archeia.com>
Date: Thu, 17 Nov 2016 11:12:17 -0800
Subject: Screenshot of bare bones
Newsgroups: comp.lang.ada

I've updated my bare bones OS project and it can now dump the location where a program crashed, using last_chance_handler.

https://github.com/Lucretia/bare_bones
https://snag.gy/JludRq.jpg

[See also “Low-level Programming”, AUJ 34-3, p. 155. —sparre]

Simple Components

From: Dmitry A. Kazakov
<mailbox@dmitry-kazakov.de>
Date: Sat, 19 Nov 2016 13:06:10 +0100
Subject: ANN: Simple Components v4.17 released
Newsgroups: comp.lang.ada

The current version provides implementations of smart pointers, directed graphs, sets, maps, B-trees, stacks, tables, string editing, unbounded arrays, expression analyzers, lock-free data structures, synchronization primitives (events, race condition free pulse events, arrays of events, reentrant mutexes, deadlock-free arrays of mutexes), pseudo-random non-repeating numbers, symmetric encoding and decoding, IEEE 754 representations support, multiple connections server/client designing tools.

http://www.dmitry-kazakov.de/ada/components.htm

Changes the previous version:
- Bug fix in GNAT.Sockets.ConnectionState_Machine.ELV_MAX_Cube_Client related to decoding valve position;
Ada-related Tools

- Set_Thermostat_Valve procedures were added to GNAT.Sockets.Connection_State_Machine.ELV_MAX_Cube_Client;
- Set_Thermostat_Parameters and Set_Thermostat_Schedule have the mode parameter added.

From: Dmitry A. Kazakov
<mailbox@dmitry-kazakov.de>
Date: Sun, 5 Feb 2017 12:36:43 +0100
Subject: ANN: GtkAda contributions v3.17
Newsgroups: comp.lang.ada

The library deals with the following issues:
- Tasking support;
- Custom models for tree view widget;
- Custom cell renderers for tree view widget;
- Multi-columned derived model;
- Extension derived model (to add columns to an existing model);
- Abstract caching model for directory-like data;
- Tree view and list view widgets for navigational browsing of abstract caching models;
- File system navigation widgets with wildcard filtering;
- Resource styles;
- Capturing resources of a widget;
- Embeddable images;
- Some missing subprograms and bug fixes;
- Measurement unit selection widget and dialogs;
- Improved hue-luminance-saturation color model;
- Simplified image buttons and buttons customizable by style properties;
- Controlled Ada types for GTK+ strong and weak references;
- Simplified means to create lists of strings;
- Spawning processes synchronously and asynchronously with pipes;
- Capturing asynchronous process standard I/O by Ada tasks and by text buffers;
- Source view widget support.

http://www.dmitry-kazakov.de/ada/gtkada_contributions.htm

This version provides minor bugs fixes for the Gtk.Main.Router package.

VTKAda

From: Leonid Dulman
<leonid.dulman@gmail.com>
Date: Sun, 27 Nov 2016 08:41:38 -0800
Subject: I’m pleased to announce VTKAda version 7.1.0 free edition release
27/11/2016
Newsgroups: comp.lang.ada

VTKAda is Ada-2012 port to VTK (Visualization Toolkit by Kitware, Inc) and Qt5 application and UI framework by Nokia VTK version 7.1.0, Qt version 5.7.0(5.8.0beta) open source and vtkc.dll,vtkc2.dll,qt5c.dll(libtvtk.so,libvtkc e2.so,libqt5c.so) were built with Microsoft Visual Studio 2015 in Windows (WIN32) and gcc in Linux x86-64.

Package was tested with gnat gpl 2012 ada compiler in Windows 10 64bit, Debian 8.3 x86-64.

As a role Ada is used in embedded systems, but with VTKAda(+QtAda) you can build any desktop applications with powerful 2D/3D rendering and imaging, games, animations, emulations) GUI, Database connection, server/client, Internet browsing and many others things.

VTKAda you can be used without QtAda subsystem Qt5Ada and VTKAda for Windows, Linux (Unix) is available from https://drive.google.com/file/d/0B2QuZLoc-yIPbmNQRl83M1dTRVE &usp=sharing (google drive. It can be mounted as virtual drive or directory or viewed with Web Browser)

[See also “VTKAda”, AUJ 35-1, p. 9. —sparre]

YAMI4

From: Maciej Sobczak
<maciej@msobczak.com>
Date: Thu, 8 Dec 2016 14:53:59 -0800
Subject: YAMI4 1.10.2 released
Newsgroups: comp.lang.ada

I am pleased to announce that YAMI4 1.10.2, which is a minor update, was just released:

http://www.inspirel.com/yami4/

The update targets specifically newer versions of GNAT, which due to stricter handling of some language constructs[*] refused to compile the older code. The library should now work correctly with newest GNAT versions.

[*] The actual language problem was occasionally discussed on comp.lang.ada, most recently in this thread: https://groups.google.com/forum/#!topic/comp.lang.ada/HNUxQAz4_FE/discussion
[See also “YAMI4”, AUJ 34-4, p. 198. —sparre]
PragmAda Reusable Components

From: PragmAda Software Engineering
<pragmada@pragmada.x10hosting.com>
Date: Sat, 10 Dec 2016 13:15:42 -0700
Subject: PragmAda Reusable Components
Update
Newsgroups: comp.lang.ada

There's a new version of the PragmARCs for ISO/IEC 8652:2007 available. This has minor changes to eliminate duplicated code between the random-number pkgs. Those who have used functions to obtain values in a range, or to obtain real values from generators that return integer results, may have to make minor changes to their code. Threefry_Random now has a block of comments describing the concept and expected use of encryption-based (counter-based) RNGs, and has a new, pure function that returns a random value to fulfill the expectations of those who have read that counter-based RNGs can be accessed in parallel.

[See also “PragmAda Reusable Components”, AUJ 37-3, p. 128. —sparre]

Image Analysis Library?

From: Luke A. Guest
<laguest@archeia.com>
Date: Wed, 25 Jan 2017 12:43:10 +0000
Subject: Re: Any suggestion for an image analysis package for Ada?
Newsgroups: comp.lang.ada

> Has anybody a suggestion for an image analysis package for Ada?
> Or Computer/Robot vision?
> Or any idea for an easy way to use for example OpenCV directly from Ada?
Create bindings to OpenCV and don't stick a GPL license on them, use the same license as OpenCV, which is BSD.

From: Björn Lundin
<bjlundin@gmail.com>
Date: Wed, 25 Jan 2017 14:55:54 +0100
Subject: Re: Any suggestion for an image analysis package for Ada?
Newsgroups: comp.lang.ada

> [...] the only binding I've heard of is described at
http://mdh.diva-portal.org/smash/get/diva2:425844/FULLTEXT01.pdf, but I'm not sure where the actual code is.
Googling gives some hints like
https://searchcode.com/codesearch/view/13067579/>, but I'm unsure if it is alive and maintained

Qt5Ada

From: Leonid Dulman
<leonid.dulman@gmail.com>
Date: Sat, 28 Jan 2017 22:21:07 -0800
Subject: Announce: Qt5Ada 5.8.0
Newsgroups: comp.lang.ada

Announce : Qt5Ada version 5.8.0 (539 packages) release 01/02/2017 free edition Qt5Ada is Ada-2012 port to Qt5 framework (based on Qt 5.8.0 final).
Qt5Ada version 5.8.0 open source and qt5.c,d,libqt5c.so(x64) built with Microsoft Visual Studio 2015 in Windows, gcc x86-64 in Linux.
Package tested with gnat gpl 2012 ada compiler in Windows 32bit and 64bit, Linux x86, Linux x86-64 Debian 8.5.
It supports GUI, SQL, Multimedia, Web, Network, Touch devices, Sensors, Bluetooth, Navigation and many others things.
Changes for new Qt5Ada release :
- Added packages for modules QWinExtras, QTTextToSpeech, QGamepad, QHelp, QSxsm, QChart modules support
- Added new demos.
- Added easy way to use Qt resource files (.qrc).

My configuration script to build Qt 5.8.0 is: configure -openssl -release -nomake tests -opengl dynamic -qt-zlib -qt-libpng -qt-libjpeg -openssl-linked OPENSSL_LIBS=-lssl -lsleay32 -llibeay32 -plugin-sql-oci -plugin-sql-odbc -plugin-sql-mysql -plugin-sql-odbc -plugin-sql-oci -iuc -prefix "c:/Qt/5.8.0".

As a role ADA is used in embedded systems, but with QTADA (+VTKADA) you can build any desktop applications with powerful 2D/3D rendering and imaging (games, animations, emulations), GUI, Database connection, server/client, Internet browsing, Modbus control and many others things.
Qt5Ada and VTKAda for Windows, Linux (Unix) is available from
https://drive.google.com/folderview?id=0 B2OuZLoe-yIPbmNQRl83M1dTRVE &usp=sharing (google drive. It can be mounted as virtual drive or directory or viewed with Web Browser).

The full list of released classes is in "Qt5 classes to Qt5Ada packages relation table.docx". VTKAda version 7.1.0 is based on VTK 7.1.0 (OpenGL2) is fully compatible with Qt5Ada 5.8.0.
I hope Qt5Ada and VTKAda will be useful for students, engineers, scientists and enthusiasts. With Qt5Ada you can build any applications and solve any problems easy and quickly.
If you have any problems or questions, tell me know.

[See also “Qt5Ada”, AUJ 36-3, p. 122. —sparre]

Deepend

From: Brad Moore
<bmoore.ada@gmail.com>
Date: Sun, 5 Feb 2017 20:25:15 -0800
Subject: ANN: Depend version 3.9 Release
Newsgroups: comp.lang.ada

It's been a while since a new release of Depend has been announced, the last announced version being version 3.4, though there have been updates since then.
Now that 3.9 has been posted, seems like a good time to mention the availability of new features and fixes.
Depend is a set of storage pools for Ada 95, Ada 2005, and Ada 2012 that includes subpool capabilities. Groups of memory allocations from a storage pool can be assigned to specific subpools where groups of objects can be deallocated as a group by deallocating the subpool, rather than by individual deallocations of objects.
Each subpool is "owned" by a specific task in Ada, allowing allocations and deallocations to be efficient, as well as being safer and less error prone.
Since version 3.4, the most notable changes are;
- Ada 2012 subpools were not working in version 3.4, but now work properly with the Ada 2012 subpool syntax.
- It is now possible to set task ownership of the storage pool itself, as well as subpools.
- Portability changes made to integrate with PTC's ObjectAda 64bit compiler for Ada 2005 and Ada 95
- Portability changes made to integrate with RR Software's Janus Ada compiler for Ada 95
- Memory allocations returned by the generic allocators were not initializing memory. This is important for types that have discriminants or tags. Now memory is initialised which is consistent with behaviour when one uses Ada's "new" keyword syntax to provide the allocations.
- The generic allocator routines now support allocating unconstrained types, such as strings.
There are 4 different storage pools to choose from;
- Unbounded storage pool with subpool support
- Bounded storage pool with subpool support
- Unbounded storage pool without subpool support
- Bounded storage pool without subpool support

Deepend source code can be found at:
https://sourceforge.net/projects/deepend/files/
[See also “Deepend”, AUJ 37-4, p. 190. —sparre]

Reference Manual in Info Format

From: Stephen Leake
<stephen_leake@stephe-leake.org>
Date: Thu, 23 Feb 2017 03:42:35 -0800
Subject: Ada Reference Manual 2012 with Technical Corrigendum 1 in info format available
Newsgroups: comp.lang.ada

The Ada Reference Manual 2012 with Technical Corrigendum 1 is now available in info format.
See http://stephe-leake.org/ada/arm.html or update the Emacs package "Ada Reference Manual"

Gnoga: PIXI Support with Sprites

From: Pascal Pignard <p.p11@orange.fr>
Date: Sat, 18 Mar 2017 19:57:45 +0100
Subject: Rep : PIXI support with sprites.
Newsgroups: gnane.comp.lang.ada.gnoga

I've rewritten PIXI Graphics closer to PIXI API rather than Context 2D API. I've also added some PIXI Containers methods.
I've pushed this on Gnoga dev_1.3 branch.

Feel free to send code review:
https://sourceforge.net/p/gnoga/code/ci/dev_1.3/tree/components/pxi/pxi/src/
One of my main concern is to define right Ada types while there are implicit in Javascript.

Beyond that, I wonder about some new API:
- Move_T to: moves sprite to a specified location with a specified speed
- On_Collision: send a Gnoga event when a collision of 2 sprites or near a location
- Acceleration: add acceleration property (positive or negative)

Any other ideas are welcome.
Feel free to point out some API you want to be available.
Take a look to "leaves" demo which brings a very promising beginning of what could be done with PIXI.
Waiting for your feedbacks on PIXI support, I'll try to bring more support on mmenu plugin.

Ada-related Products

Rapita Verification Suite

From: Rapita Systems
Date: Fri Dec 9 2016
Subject: RVS 3.6 released
URL: https://www.rapitasystems.com/news/rvs-36-released

Rapita Systems is proud to announce the latest release of its on-target software verification tool suite RVS, version 3.6. Over the last 6 months, we have been working tirelessly to make this our highest quality release to date.
Testing processes in the development of software for critical real-time embedded systems are incredibly expensive, both in terms of the effort required to run tests and subsequent analysis effort. We believe that our verification tools should improve the efficiency of these testing processes, by being designed to work seamlessly through the development process.

This is why we have been developing new features in RVS 3.6 that help direct the testing process, reducing the effort required to repeat tests and trace results to tests and requirements. These new features, as well as some of the other improvements made since RVS 3.5, are listed below:
- The new Treemaps feature in RapiCover lets you visualize your coverage at a glance
- New options for managing tests and subprograms allow you to filter coverage with fine granularity
- Our new Optimal Dataset Calculator can determine the minimal set of tests you must run again when you change code
- Our improved justification workflow gives you new options, such as including custom fields in templates
- RVS now supports many new Ada 2012 features, and recent GNAT compilers
- Our coverage parsing tool now lets you process multiple tests in a single command-line
- We have fixed over 150 bugs

You can find more information on some of the new features available in RVS 3.6 on our website.

[...] [See also “Rapita Verification Suite”, AUJ 37-2, p. 77. —sparre]

GNAT Pro, CodePeer, QGen and SPARK Pro

From: AdaCore Press Center
Date: Tue Mar 14 2017

Subject: AdaCore Releases New Versions of GNAT Pro, CodePeer, QGen and SPARK Pro
URL: http://www.adacore.com/press/adacore-releases-v17/

Annual major release of flagship products brings new platform support, other enhancements

NUREMBERG, Germany, Embedded World Conference, March 14, 2017. AdaCore today announced the release of the latest version of its four major products:
- GNAT Pro 17.1, a development environment for Ada and C, on native and cross platforms;
- CodePeer 17.1, a deep static analysis tool for Ada that can identify bugs and vulnerabilities both during development and retrospectively on existing code bases;
- QGen 17.1, a model-based development and verification toolset for Simulink® and Stateflow® models, which generates code in MISRA-C or SPARK; and
- SPARK Pro 17.1, a verification environment that brings mathematics-based assurance to high-integrity software.

“Developing and verifying critical systems is a challenging task, especially when certification against software standards such as DO-178C or EN 50128 is required,” said Cyrille Comar, AdaCore President. “The latest version of our products will help organizations meet this challenge, through enhancements such as QGen’s model-level debugger and CodePeer’s detection of dangerous CWE weaknesses. Customers have long relied on AdaCore’s tools and services when producing safety-critical or high-security software, and our V17.1 product line marks a continuation of our commitment.”

GNAT Pro includes a full-featured build toolset for Ada and C, Integrated Development Environments (the GNAT Programming Studio (GPS) and the Eclipse-based GNATBench), a comprehensive suite of tools (a visual debugger, a coding standard checker, etc.), and an extensive set of libraries and bindings. The GNAT Pro 17.1 release, based on GCC 6 and GDB 7.10, includes improved debugger support under GPS, a better algorithm for Ada elaboration order, enhancements to the GPIBuild multi-language build tool, better integration of GNATest and GNATcoverge, an implementation of the extended Ravenscar profile on bare metal targets, and support for SMP on leon3. The supplemental GNATcoverage dynamic analysis tool has been upgraded with incremental coverage analysis, improved object code coverage, and support for ARM bare metal and native Windows (both 32- and 64-bit) platforms.
Ada and Operating Systems

Linux: Interfacing to External Hardware

From: Patrick
<patrick@spellingbeewinnars.org>
Date: Sun, 1 Jan 2017 18:42:04 -0800
Subject: Interfacing Ada With Full Runtime Directly to Electronic Chips
Newsgroups: comp.lang.ada

Ada-AVR is a neat project and I am not knocking it. It doesn't have tasking support yet but this is of course a huge job.

If I want to connect any sort of computer running Linux directly to a circuit using "full Ada" with tasking support what are my options?

By circuit, I mean a variety of electronic components and in this case with no microcontroller, an A/D chip is a simple example.

I am thinking that single board computers like BeagleBone are my best bet, they have plenty of GPIO lines.

Has anyone interfaced directly with chips via SPI or IC2 via a GPIO PCI card or GPIO-USB adapter?

Are there any other options?

From: Brian Drummond
<brian@shapes.demon.co.uk>
Date: Mon, 2 Jan 2017 14:02:36 -0000
Subject: Re: Interfacing Ada With Full Runtime Directly to Electronic Chips
Newsgroups: comp.lang.ada

> [...] There are ARM-Cortex development boards with a bit more resources, and Ada support including (Ravenscar) tasking, either bare-bones or over a small RTOS.

Look for work by Simon Wright in respect of these.

Much simpler than Beaglebone, but not full Linux.

Interfacing to hardware isn't difficult. Generally you abstract the hardware level into a package which knows where the register addresses are, and different (related) targets use different implementations of the packages.

For full Linux, I'd look at the Raspberry Pi. But then you get involved in writing device drivers - there's a project on the first steps on writing Linux device drivers in Ada, but it's more complex, and not as well trodden as Linux device drivers in C.

From: Simon Wright
<simon@pushface.org>
Date: Mon, 02 Jan 2017 16:25:23 +0000
Subject: Re: Interfacing Ada With Full Runtime Directly to Electronic Chips
Newsgroups: comp.lang.ada

> [...] You can talk I2C from a Raspberry Pi; see [1], [2]. It was simple to install GNAT etc from libre.adacore.com, since Raspbian is a Debian offshoot.

Beaglebone also supports Debian, see [3] for a report on this.

Both of the above use OS device support via file read/write/ioctl. If you want to go bare(ish) metal, there is the AdaCore Ada_Drivers_Library at [4]. This uses cross-compilation for ARM Cortex MCUs, compilers from AdaCore (or [5], [6] for Mac), with runtimes for boards mostly from STMicroelectronics (e.g. [7]).

The AdaCore runtimes support Ravenscar tasking[8] and come in two flavours, small footprint (-sf-) and full (-full-).

The -full- version supports exception handling and finalization and includes...
Ada.Numereics. Neither support containers easy enough to copy into the -full-
version, I expect).
The Ada Drivers Library uses a BSD license. The AdaCore runtimes use a full-
GPL license. If this matters to you I have runtimes for Arduino Due and
STM32F4[9] which are based on FreeRTOS and have the GCC Runtime
Library Exception, allowing release of code on proprietary terms.
Just to indicate the flavour of this bare-
metal code, I have some SPI code for an
STM32F427, using interface code
generated by AdaCore’s SV22Ada[10], at
[1] https://sourceforge.net/projects/
raspi-i2c-c ada/ 
[3] https://groups.google.com/forum/
#topic/beagleboard/OSAUX2XLN3J8 
Ada_Drivers_Library 
gnuada/files/GNAT_GPL%20Mac%20 
OS%202016-arm-eabi-darwin-bin/ 
gnuada/files/GNAT_GCC%20Mac%20 
OS%202016/6.1.0/arm-eabi/ 
stmicroelectronics/stm32f407g-
disc1/dev-board-foundation-line-
mcu/df/2506840 
Ravenscar_profile 
cortex-gnat-rtts/ 
[10] https://github.com/AdaCore/svd2ada 
multiplexed-ios/iblok/master/drivers/spi1/
src/spi1-device.adb

From: Philip Munts <philip.munts@gmail.com>
Date: Mon, 2 Jan 2017 23:55:50 -0800
Subject: Re: Interfacing Ada With Full
Runtime Directly to Electronic Chips
Newsgroups: comp.lang.ada

> [...] 
I suggest my own Linux Simple I/O
Library (http://git.munts.com/lsbsimpleio). It 
provides Pascal calling sequence
wrappers around the Linux system calls for
I2C, SPI, UART, and raw HID
devices and includes some shim packages
for GNAT.
I have tested it with both native and cross
compilers on the BeagleBones Black and
Green and Raspberry Pi 2 and 3. I use it
mostly with the Raspberry Pi 2 cross
toolchain (which also works fine for the
BeagleBone family) from AdaCore and
targeted to my own embedded Linux
distribution called MuntsOS
(http://git.munts.com/arm-linux-mcu).
I also have a collection of packages and
example programs for various
Mikroelectronika click boards and
Raspberry Pi hats that are not published
yet. I am in the middle of writing a paper
about this very topic.

FreeBSD/ARM64: GNAT
From: John Marino
<dragonlance.cx@marino.st>
Date: Wed, 8 Feb 2017 07:30:55 -0800
Subject: ANN: GNAT for FreeBSD/64
Newsgroups: comp.lang.ada
If anybody is interested in Ada on the 64-
bit ARMv8 architecture, there are a
couple more options available to you
today.

Last week I created a FreeBSD/DragonFly to FreeBSD64/ARM
cross-compiler and placed it in ports:
http://www.freshports.org/lang/
gnatcross-aarch64/
That compiler was based on previous
GnatDroid work.

Over the weekend, I used this
cross-compiler to fully bootstrap FSF GCC
6.3.1 to FreeBSD/ARM64. Now the entire
FreeBSD Ports Ada framework is available
on this soon-to-be-tier-1
platform. The existing gcc6-aux port
(http://www.freshports.org/lang/gcc6-aux)
was expanded to support aarch64.
It passes the complete testsuite. That
required providing a freebsd-specific
signal frame unwinder which I’m passing
back to GCC.
results: http://www.dragonlance.net/
gnataux/freebsd_arm64/
I also mentioned this news on my mostly
neglected website:
http://www.dragonlance.net/

MacOS X: GDB on Sierra
From: Simon Wright
<simon@pushface.org>
Date: Mon, 13 Feb 2017 22:09:41 +0000
Subject: GDB vs macOS Sierra
Newsgroups: gnuada/files/GNAT_GCC%20Mac%20 
OS%202016/6.1.0/ 

To: GNAT-OSX-
dhAwdhUhaNgMT+7pcfOT8A@public.g
mane.org 

Apple’s security enhancements in Sierra
prevent the GDBs available online (in
GNAT GPL 2016 from
libre.adacore.com, FSF GCC 6.1.0 at 
Sourceforge) from working at all.
Discussion on an updated version and
how to install it at http://forward-in-
code.blogspot.co.uk/2017/02/gdb-vs-
macos-sierra.html.

[...]

Debian: GNAT
From: Nicolas Boulenguez
<nicolas.boulenguez@free.fr>
Date: Sat, 18 Feb 2017 22:06:01 +0100
Subject: some news
Newsgroups: 
gnuada/freebsd_arm64/

The compiler for next release will be
gnat-7, currently only in experimental.
I suggest that we use this release to
change the directory project in policy.
Installing projects to
/usr/share/ada/adainclude creates
unmotivated divergence with upstream’s
/usr/share/gpr.
Users of gpbuild will not notice, it
already searches in both directories.
Libxmlada compiles and the basic
link/run test passes with this change, so
I have good hope for gpbuild. Most other
libraries rely on dh-ada-library and will
be updated automatically during next build.

OpenVMS: GNAT
From: Gérard Calliet <gerard.calliet@pia-
sofer.fr>
Date: Thu, 2 Mar 2017 13:37:12 +0100
Subject: Gnat Ada on OpenVMS is back
Newsgroups: comp.lang.ada

The OpenVMS OS, after an announced
death in 2013, is back with its primary
engineering team, organized as a start-up
(www.vmssoftware.com).
In this context, we did a new build on
OpenVMS / Itanium of the Gnat Ada
compiler. We needed it for an industrial
project (a port from OenVMs Alpha to
OpenVMS Itanium, an urban
transportation control software). The
project is about to be run in production,
with images compiled by our Gnat Ada
compiler.
We want to generalize this renewal of
Ada on OpenVMS. We offer free of
charge our compiler (with no support) to
downloading. The portal (still in work)
www.vmsadaafl.org can be used to ask for
a download. We wish it becomes a place
for exchanges between all people that are
interested by the initiative.
If you are interested, please tell us, we
need evaluations, advice, ideas.

References to Publications

Writing on Air
From: Dirk Craeynest
<dirk@jeles.cs.kuleuven.be>
Date: Mon Apr 03 2017
Subject: Great "Ada Inside" demo -
"Writing on Air"
URL: http://blog.adacore.com/
writing-on-air
DIY Instant Camera

From: Fabien Chouteau
<fabien.chouteau@gmail.com>
Date: Mon Dec 12 2016
Subject: Make with Ada: DIY instant camera
URL: http://blog.adacore.com/make-with-ada-diy-instant-camera

There are moments in life where you find yourself with an Adafruit thermal printer in one hand, and an OpenMV camera in the other. You bought the former years ago, knowing that you would do something cool with it, and you are playing with the latter in the context of a Hackaday Prize project. When that moment comes — and you know it will come — it’s time to make a DIY instant camera. For me it was at the end of a warm Parisian summer day. The idea kept me awake until 5am, putting the pieces together in my head, designing an enclosure that would look like a camera. Here’s the result:

Introductory Ada Programming Book

From: Andrew Shvets
<andrew.shvets@gmail.com>
Date: Sat, 31 Dec 2016 05:18:07 -0800
Subject: Introductory Ada Programming Book

My name is Andrew Shvets. I've been learning how to program in Ada over the past few years. As someone that came from C/C++, Java and Python many of the concepts that were in Ada were not easy to digest at first. After spending a fair amount of time looking for a guide that would help me out (something that would guide me through much of the ideas in Ada in a gentle manner was strongly preferred and be focused on Ada 2012), I couldn't find something like this (there is, on the other hand plenty of material that would explain more advanced concepts.) Having mastered many of the basic concepts in this wonderful language, I figured that having a guide for this would be very helpful and wrote one. Hence the book "Introductory Ada Programming Book: A Book for Beginner Programmers and Beginners to Ada". The goal was to create a roadmap for those new to Ada learn more quickly and gain a certain level of mastery.


I'm open to sending PDFs as review copies, please send your requests to: introductory dot ada at gmail dot com

Thank you for taking the time to read through this! Also, thank you for those -- on this newsgroup -- who have helped me better understand Ada!

Looking forward to hearing from everyone!

Very Simple Scheduler

From: Maciej Sobczak
<maciej@msobczak.com>
Date: Tue, 24 Jan 2017 00:16:34 -0800
Subject: Very Simple Ada Scheduler

This article is an extension chapter for the Ada on ARM Cortex-M tutorial and presents a very simple, but yet surprisingly flexible scheduler for managing multiple tasks (well, finite state machines) in a single embedded system.

The example program from this article deals with 3 independently blinking LEDs, but can be easily extended to more complex designs.

Your comments are welcome,

CAN Newsletter: Ada for Automation

From: Stéphane Los
<new.stephane.los@gmail.com>
Date: Sun, 3 Mar 2017 12:32:51 -0800
Subject: "Ada for Automation" in the CAN Newsletter magazine March 2017: 25th anniversary

I wanted to let you know about the "CAN Newsletter magazine March 2017: 25th anniversary" which features an article about "Ada for Automation".

The magazine:
https://can-newsletter.org/anniversary/170224-can-newsletter-march-2017

Ada for Automation: Ada language for automation
https://can-newsletter.org/uploads/media/raw1/1cb325a3453440e4ae703be8e392763.pdf

References to Publications

Handbook on DO-178C/ED-12C Guidance

From: AdaCore Press Center
Date: Tue Mar 21 2017
Subject: AdaCore Publishes Handbook on DO-178C/ED-12C Guidance

Free booklet shows how AdaCore qualified tools can reduce costs of certifying airborne software

TOULOUSE, France, Certification Together International Conference, March 21, 2017 – AdaCore today announced the publication and immediate availability of a free booklet, AdaCore Technologies for DO-178C / ED-12C, written by Quentin Ochem (AdaCore) and certification expert Frédéric Pothon. The booklet addresses the DO-178C / ED-12C standards suite – the “core” DO-178C / ED-12C standard and its technology supplements – and explains many of their more subtle aspects in the context of several different development scenarios. In so doing, the booklet provides insights into how the Ada and SPARK languages, combined with AdaCore’s products and services, can help customers develop and verify airborne software. Many of AdaCore’s tools have been qualified on safety-critical projects and have qualification material available; using a qualified tool can save considerable effort in demonstrating that various objectives in the DO-178C / ED-12C standards suite have been met.

“DO-178C is one of the most complex software safety standards in the industry,” said Quentin Ochem, lead of Business Development at AdaCore. “This booklet is aimed at software engineers and architects, to help them read between the lines of the standard and better understand the intent, using AdaCore’s technologies to illustrate how to meet the various requirements.”

The booklet approaches its subject matter from several angles. One chapter summarizes the Ada and SPARK languages and describes various AdaCore tools, many of which have been qualified or are qualifiable for safety-critical systems:

- The GNAT Pro Assurance development environment, including support for “sustained branches”, which allows customers to evolve their software on a stable but maintained version of the GNAT Pro environment;
- The CodePeer advanced static analysis tool for Ada, which can find subtle bugs and vulnerabilities both during development and retrospectively on existing codebases;
- Basic static analysis tools, including the GNATcheck code standard enforcer and
the GNATstack tool for computing maximum stack usage;
- Dynamic analysis tools: GNATTest (a test harness generator), GNATEmulator (a target emulator), and GNATCoverage (a code coverage analyzer at both the object and source levels, handling statement coverage, decision coverage, and modified condition/decision coverage);
- Integrated Development Environments: GNAT Programming Studio (GPS), GNATbench, and GNATdashboard; and
- The QGen model-based development and verification toolset, which includes a tunable and qualifiable code generator from a safe subset of Simulink® and Stateflow® models to SPARK or MISRA-C.

A major section of the booklet is a chapter that shows how to exploit AdaCore’s technologies to comply with the guidance in the DO-178C / ED-12C suite, in the context of several development scenarios (use cases):
- Coding with Ada 2012 without using Object-Oriented Technology (OOT). This use case shows how AdaCore’s products and services contribute to the activities in the core DO-178C / ED-12C standard.
- Coding with Ada 2012 and using OOT. This use case takes into account the guidance in DO-332 / ED-217 (Object-Oriented Technologies and Related Techniques), in particular the objective of Local Type Consistency.
- Developing a design model and using a qualified code generator (QGen). This use case takes into account the guidance in DO-331 / ED-218 (Model-Based Development and Verification).
- Using SPARK and formal analysis. This use case takes into account DO-333 / ED-216 (Formal Methods) and explains how to gain credit for formal proofs to reduce or eliminate testing activities.

For each use case, the booklet explains which AdaCore technologies are applicable, and which activities in the DO-178C / ED-12 suite they contribute to.

The booklet also includes a set of reference tables that summarize how the various AdaCore technologies help satisfy the specific objectives in the DO-178C / ED-12 suite.

Availability
The DO-178C booklet is available now, at no cost. To download a PDF version, please visit www.adacore.com/tech-do-178c. For a printed copy please contact info@adacore.com

---

### Ada Inside

#### SparForte

*From: Ken O. Burtch*

*Date: Mon, 14 Nov 2016 06:32:27 -0800*

*Subject: [ANN] SparForte 2.0*

*Newsgroups: comp.lang.ada*

SparForte is a shell, scripting and web template language based on Ada.

Version 2.0 was released on October 10, 2016.

Major new features include:
- in out, out mode parameters on user functions
- GNU readline is used for the command prompt
- Berkeley DB support
- Software architect policy blocks

Also the database configuration issues should be resolved (thanks to volunteers).

The SparForte home page is:

http://www.sparforte.com

A summary of the changes are found here:

http://www.pegasoft.ca/coder/coder_october_2016.html

The full release notes are found here:


[See also “SparForte”, AUJ 37-3, p. 130. —sparre]

*From: Ken O. Burtch*  

*Date: Wed, 28 Dec 2016 11:34:00 -0800*

*Subject: [ANN] SparForte 2.0.1*

*Newsgroups: comp.lang.ada*

SparForte 2.0.1 is bug fix release of SparForte 2.0.

SparForte is an open source shell, scripting and web template language based on Ada. It runs on Linux and FreeBSD. It is maintained by myself and volunteers.

There are 17 updates included in this release. The change log can be found here:


The language can be downloaded from the SparForte home page:

http://www.sparforte.com/index.html

*From: Ken O. Burtch*  

*Date: Thu, 2 Feb 2017 10:10:05 -0800*

*Subject: [ANN] SparForte 2.0.2*

*Newsgroups: comp.lang.ada*

SparForte 2.0.2 contains follow-up bug fixes for SparForte 2.0. There are 19

---

### MAX! Home Automation

*From: Dmitry A. Kazakov*  

*Date: Mon, 21 Nov 2016 18:17:37 +0100*

*Subject: MAX! home automation v1.6 released*

*Newsgroups: comp.lang.ada*

MAX! home automation is a GTK application used to control ELV/eQ3 network of heating devices, like radiator thermostats, shutter contacts etc.

http://www.dmitry-kazakov.de/ada/max_home_automation.htm

Changes to the previous version:
- Monitoring radio band duty cycle was added;
- Commands that control thermostats are serialized and monitored for failures;
- Cube radiator thermostats configuration save and restore added;
- Thermostat schedule and parameters upload diagnostics added;
- Offset temperature can be set negative;
- HTTP server request documentation get-set_temperature changed to get-set-temperature;
- Save file dialogs changed to ask override confirmation;
- Bug fix in HTTP server that prevented querying the thermostat's measured temperature;
- Valve position is correctly reported;
- Thermostat parameters and schedule optimized to minimize RF traffic;
- Documentation extended with instructions how to run the application remotely or in the HEADLESS mode.

[See also “MAX! Home Automation”, AUJ 37-2, p. 79. —sparre]

*From: Dmitry A. Kazakov*  

*Date: Wed, 22 Feb 2017 18:07:19 +0100*

*Subject: MAX! home automation v1.8 released*

*Newsgroups: comp.lang.ada*

...
Muen Separation Kernel

From: Adrian-Ken Rueegsegger  
<ken@codelabs.ch>

Date: Mon, 6 Feb 2017 19:30:05 +0100
Subject: [ANN] Muen development version 0.8 released

Newsgroups: comp.lang.ada

We are proud to announce the availability of Muen version 0.8. The following major features and improvements have been implemented since the last release:

**Subject Lifecycle**

With the implementation of a subject loader (SL) component it is now possible to reset and restart subjects at runtime with minimal support from the kernel. The new loader policy abstraction eases the specification of loader subjects which control the lifecycle of one or multiple subjects. Aside from the state, loaders have access to the entire address space of a managed subject in order to be able to set up the execution environment.

The new SL component written in SPARK 2014 initializes/resets the writable memory regions of its associated subjects. To find the actual memory regions to process, the loader consults each monitored subject info region by querying it using the new subject info (sinfo) client library. Prior to starting execution of a monitored subject, integrity hashes of each memory region are calculated and compared to their reference value provided by the sinfo data.

The demo system has been adapted to make use of this new functionality: the NIC Linux can be restarted by pressing the Right Control+F10 key combination.

**System reboot/shutdown**

The introduction of a new `kernel` event mode and event actions for system reboot and poweroff enables the policy writers to grant subjects the capability to initiate a system reboot or shutdown. If a subject triggers a reboot event, the kernel performs a power-cycle using the Reset Control Register (I/O port 0x0f). The shutdown functionality is implemented by performing an ACPI shutdown using the poweroff port and the PM1A Control Sleep Type capability.

As with the subject restart, the demo system has been extended to showcase the new feature: a system reboot can be triggered by pressing the Right Control+F11 keys while Right Control+F12 initiates a shutdown.

**Policy & Toolchain improvements**

The Muen policy has been extended with a config section which enables the parametrization of a system via the declaration of configuration values. Boolean expressions referencing existing config values can be used to formulate additional properties. The introduction of conditionals that reference config values or expressions selectively enable/disable parts of the policy which allows for flexible customization of a system at integration time.

To increase the expressiveness of the policy and ease native component development, component descriptions have been extended with the library construct including dependency declaration. Additionally it is now possible to declare logical devices as well as memory and channel array resources. The new component spec generation tool reads the description of a given component and generates Ada/SPARK packages containing constants of the declared logical component resources. These generated specifications can be used in the component source code to access the declared resources. This ensures a consistent view of the system according to the policy.

Physical memory regions can now optionally declare an integrity hash. The SHA-256 hash value can be used to verify the initial content of a given region at runtime. A tool has been implemented that calculates hashes for all physical regions with content (file or pattern). The region hashes are exported as part of the subject info data and used by the SL as described above.

**Kernel improvements**

To prevent potential issues related to Intel Hyper-Threading, e.g. side-channels, the kernel now only activates one thread per physical CPU core. CPUID leaf 11 (Extended Topology Enumeration Leaf) is used to determine the SMT ID of each logical CPU and deactivate it if the ID is non-zero. This effectively disables HT even if no such option is present in the BIOS.

Feature-wise, support for event actions has been added. The improved functionality increases the flexibility of the event mechanism and facilitates subject lifecycle management as well as system restart and shutdown.

Many other improvements and stabilizations such as e.g. replacement of assembler code with SPARK and reduction of stack size to 4K have been implemented.

**Further changes**

Further changes and improvements include:

- Support for 32-bit Windows VMs using Genode/Virtualbox
- Support for Intel Broadwell and Skylake microarchitecture
- Support for PCI multi-function device pass-through
- Implement tool to statically calculate worst-case stack usage
- Implement tool to generate scheduling plans
- Implement subject info client library in SPARK 2014
- Update of Linux kernel to version 4.6
- Add support for multiple intramfs per Linux subject

With the advent of subject lifecycle management, system shutdown and reboot support and the numerous toolchain improvements, the Muen platform is getting more and more mature. The continuous enhancement and refinement of system policy abstractions, especially the components construct, further simplifies the system specification process and enables the description of complex systems composed of a large number of subjects.

We are also thrilled that this release includes the previously announced [1] support for execution of hardware-accelerated 32-bit Windows VMs on top of the Muen SK through the use of Genode/VirtualBox as a privileged VMM. A detailed description of the architecture and an account of how this feature came to be can be found in the Genode 16.08 release notes [2].

Last but not least we would like to thank Christiane Kuhn for her contribution of the scheduling plan generation tool [3] which she developed as part of a student project/internship.

Further information about Muen is available on the project website [4] and the git repository can be found at [5].

Please feel free to give the latest development version of Muen a try. As always, feedback is very much appreciated!

[1] https://groups.google.com/forum/#!topic/muen-dev/IcZzFidK8ec

[See also “Muen Separation Kernel”, AUJ 37-1, p. 18. —spar]

From: Adrian-Ken Rueegsegger  
<ken@codelabs.ch>

Date: Mon, 13 Feb 2017 18:07:39 +0100
Subject: Re: [ANN] Muen development version 0.8 released

Newsgroups: comp.lang.ada

[...] 1. I understand that this separation
Kernel is an hypervisor, but how does it compare to other hypervisor like for instance the one from Wind River?
Does Muen allow hard Real Time software? What kind of scheduling does the Muen kernel? I am not sure how to best answer this question since I do not know the Wind River Hypervisor. Looking at the product information online some of the features are quite similar while in other areas there are differences. Since I am not in a position to make a meaningful comparison, let me instead list some of the features of Muen to give you a better picture:

Static resource allocation
One of the main design choices of the Muen Separation Kernel and systems build on top of it is, that all system resources such as memory, devices, CPU time etc are assigned to subjects via the system policy at integration time. As a consequence, systems have a static structure which does not change during runtime. This means that the SK does not perform any dynamic resource allocation at runtime, which greatly simplifies the kernel design and implementation. Note that Muen has support for Intel VT-d (DMA and Interrupt remapping) which facilitates PCI device pass-through.

Multicore support
All cores provided by a hardware platform can be used for subject execution.

Scheduling
Muen implements a fixed, cyclic scheduler. Execution order and time assignment of each subject is specified in the system policy. The kernel enforces the scheduling plan during runtime by executing each subject for the allotted time and then (preemptively) switching to the next one.

Subject execution environment
Since Muen employs Intel Virtualization Extensions (VT-x) incl. Unrestricted Guest Support and Nested Paging/Extended Page Tables (EPT) as a basic mechanism to execute and separate subjects, it can run many different types of subjects:
- Native 64-bit Ada & SPARK 2014 subjects.
- Linux 32/64-bit VMs.
- Genode x86_64 base-hw system.
- Windows 32-bit VMs (by means of Genode/VirtualBox).
A subject execution environment can be further customized in the system policy, e.g. allow access to the Timestamp Counter (TSC).

Small size and low complexity
During the development of Muen we took deliberate design choices to minimize the overall kernel complexity. The simplification drastically reduces code size which is illustrated by the current, tiny kernel size of ~5'500 sloc. This make the kernel suitable for formal verification. On the other hand, a consequence is that the supported feature set is smaller than what you get from a general purpose/dynamic hypervisor. Here are two examples of restrictions imposed by Muen:
- Subjects are not allowed to migrate between CPU cores
- Muen only runs on recent x86_64 hardware with Intel VT-x and EPT support

Aside from these features there are some additional points that we think are important.

Availability of code & documentation
Muen is an open source project and we publish all code and documentation. We think it is paramount for an SK to be independently inspectable since it is always part of the Trusted Computing Base (TCB) of any system built on top of Muen. The small size makes it realistically possible to read and understand the entire Muen kernel code.

Formal Verification
Since Muen is implemented in SPARK 2014, we prove full absence of runtime errors at the source code level. Since we publish the entire source code and because the SPARK GPL toolchain by AdaCore is freely available, everybody can independently reproduce these proofs in their own environment.

> 2. How does Muen compare to solutions like Linux KVM? How does it compare to Xen?

Similar to the first question it is not so easy to compare Muen to a fully fledged hypervisor such as KVM and Xen since it is comparing apples with oranges. Xen and KVM are fully-fledged hypervisors which feature dynamic VM construction and deconstruction, live migration etc.

With Muen systems you have to take more of an embedded system development approach, where you know your target hardware platform and system structure at integration time.

I think in the end it comes down to this: Since Muen is a Separation Kernel it (intentionally) does not address all use cases targeted by general purpose hypervisor.

> 3. Will it be possible to use Muen with the future OpenVMS on x86-64?

I have no prior experience with OpenVMS. Since there are no inherent restrictions imposed by Muen on the kind of subjects that can be executed it should in principle be possible. However, without more technical information and access to OpenVMS x86-64 there is no way for me to estimate the required porting effort.

PS: If you have more questions regarding Muen there is also a project mailing list, see [1].

[1] https://muen.sk/#mailing-list

From: Adrian-Ken Rueegsegger <ken@iodelabs.ch>
Date: Wed, 1 Mar 2017 22:39:24 +0100
Subject: Re: Unikernel / Ada
Newsgroups: comp.lang.ada

> > Some time ago I read some papers on the MirageOS (https://mirage.io/), a library operating system. The model works in short (simplified): The application sources (in case of Mirage in OCaml) are compiled / linked together with all its depending "library os" sources into one fully standalone binary (unikernel). This binary is then deployed directly. E.g. on a Xen Hypervisor. No complex OS involved. Simple, efficient, more secure, fast to boot,

> > https://mirage.io/wiki/overview-of-mirage

> > > I find this model interesting for GNAT. Maybe there are already some ideas around?

Funnly you should mention MirageOS specifically. I have been toying around with it over the past weekend and actually managed to run some example scenarios as subjects on the Muen Separation Kernel [1]. I posted a small teaser here [2].

Regarding Ada: we have published a few SPARK/Ada subjects that run natively on Muen. Currently we only provide a zero-footprint runtime which obviously restricts the language features one can use to write native Ada/SKAP subjects. However, there is no inherent limitation, it is simply a matter of extending the runtime.

Apropos of developing a TCP/IP stack from scratch: this might be of interest [3].

[1] https://muen.sk
[2] https://twitter.com/Kensan42/status/835941733359882240

Running Gnoga in Amazon Cloud

From: Björn Lundin <b.f.lundin@gmail.com>
Date: Mon, 20 Mar 2017 09:22:15 +0100
Subject: Re: Experience with Amazon Web Services?
Newsgroups: gmane.comp.lang.ada.gnoga

> > Did anyone deploy a GNOGA app on Amazon Web Services (more precisely...
Amazon Cloud service, https://aws.amazon.com/ec2/ ?
Yes, I did a year ago or so.
It just worked - after my friend (the one with Apache know-how of us) had set it up.
> Any advice, caveat, feedback ?
http://https is redirected to the gnoga-service for a certain path.
There is also some stuff handling upgrading the connection to web sockets.

> [...] 
From: Björn Lundin <b.lundin@gmail.com>
Date: Mon, 20 Mar 2017 13:59:48 +0100
Subject: Re: Experience with Amazon Web Services ?
Newsgroups: gmane.comp.lang.ada.gnoga
Björn Lundin wrote:
> http://https is redirected to the gnoga-service for a certain path.
> There is also some stuff handling upgrading the connection to web sockets. No there was not. I think now that was before we went through Apache.

> [...] 
The gnoga part listen at http://localhost:9080 and all https stuff is via Apache2
http is redirected to https some browser do not need to login, others do
The machine is known to DNS servers
This is on a uname -a
Linux prod 3.2.0-4-amd64 #1 SMP Debian 3.2.68-1+deb7u2 x86_64
GNU/Linux
cat /etc/apache2/sites-enabled/030-
Debian 3.2.68-1+deb7u2 x86_64
Linux prod 3.2.0-4-amd64 #1 SMP Debian 3.2.68-1+deb7u2 x86_64
GNU/Linux
uname -a
http is redirected to https some browser via Apache2
When it's time to run the S/W on the real system, you eliminate the environment pkg and replace the H/W I/F bodies with ones that actually I/F with the H/W. Note that the only differences between the simulated and actual systems are those bodies.
This approach has a number of benefits:
- Changing a device only affects a S/W I/F body and a H/W I/F pkg.
- Often the simplest and clearest design for the core S/W needs to access information or take action in a way the intended H/W doesn't support. The S/W I/F pkg provides a single place to convert between the 2 views, keeping the core S/W uncoupled. For example, in the ubiquitous cruise-control problem, the best approach for the core S/W might be for it to decide when it obtains the car's speed, but a common design for the speed sensor is something that generates an interrupt every time something rotates a certain amount.
- While there is usually a 1:1 correspondence between S/W and H/W I/F pkgs, there need not be. I’ve seen sensors that returned multiple, unrelated values. The design had multiple S/W I/F pkgs interacting with a single H/W I/F pkg.
- I’ve worked on projects where the whole point was to create a simulation to see if the approach is viable, with no idea what the H/W devices would be like in a real system. By using this approach, when it was decided to go ahead with a real system, only the H/W I/F pkgs and the S/W I/F bodies had to be rewritten.
When I present such a design, coders usually start whining about “efficiency”. In my decades of experience, such a design has never been responsible for a

Ada in Context
Getting Started with Bare-board Development
From: Jeffrey R. Carter <jrcarter@acm.org>
Date: Sat, 12 Nov 2016 21:01:36 -0700
Subject: Re: Getting started with bare-board development
Newsgroups: comp.lang.ada
Adam Jensen wrote:
> How is it done in embedded software engineering? (Links and/or references are very welcome!)
Typically embedded S/W has to interface to various H/W devices (sensors and actuators). Frequently such S/W is designed around the capabilities and features of the intended H/W. This is not a good idea. When the intended H/W changes (as it does frequently on the projects I’ve been involved in) the entire design has to be revised.
What I have done when designing such S/W is to first design the core S/W without regard to the capabilities and features of the intended H/W. I create the simplest and clearest design, and this identifies the kind of information the S/W needs to obtain and the kind of external actions it needs to take.
Next, for each piece of external information the S/W needs to obtain, I write a pkg spec for a S/W-leaning interface. This keeps the S/W and clear by providing just the kind of I/F it needs.
Then, for each intended H/W device, I write a pkg spec for a H/W-leaning interface. This reflects the capabilities and features of the device.
Then I write bodies for each of the S/W I/F pkgs that use the H/W I/F pkgs.
Now comes the fun part. I write an environment pkg that simulates reality, and write simulation bodies for the H/W I/F pkgs that read or modify that simulated reality. The body can take do things to make its behavior realistic; for example, if a sensor is noisy, the body would add noise to the real value.
This lets you play with your S/W and see if it behaves reasonably.
When it's time to run the S/W on the real system, you eliminate the environment pkg and replace the H/W I/F bodies with ones that actually I/F with the H/W. Note that the only differences between the simulated and actual systems are those bodies.
This approach has a number of benefits:
- Changing a device only affects a S/W I/F body and a H/W I/F pkg.
- Often the simplest and clearest design for the core S/W needs to access information or take action in a way the intended H/W doesn't support. The S/W I/F pkg provides a single place to convert between the 2 views, keeping the core S/W uncoupled. For example, in the ubiquitous cruise-control problem, the best approach for the core S/W might be for it to decide when it obtains the car's speed, but a common design for the speed sensor is something that generates an interrupt every time something rotates a certain amount.
- While there is usually a 1:1 correspondence between S/W and H/W I/F pkgs, there need not be. I’ve seen sensors that returned multiple, unrelated values. The design had multiple S/W I/F pkgs interacting with a single H/W I/F pkg.
- I’ve worked on projects where the whole point was to create a simulation to see if the approach is viable, with no idea what the H/W devices would be like in a real system. By using this approach, when it was decided to go ahead with a real system, only the H/W I/F pkgs and the S/W I/F bodies had to be rewritten.
When I present such a design, coders usually start whining about "efficiency". In my decades of experience, such a design has never been responsible for a
system not meeting its timing requirements.

From: Jeffrey R. Carter <jrccarter@acm.org>
Date: Sun, 13 Nov 2016 14:04:59 -0700
Subject: Re: Getting started with bare-board development
Newsgroups: comp.lang.ada
Adam Jensen wrote:
> When writing device drivers, how do you mock the memory map of the target hardware?
>
> In the mocked hardware, how is timing controlled?
>
> When extending and mapping run-time support to the mocked hardware, how does that fit into the run-time system for the native platform (your workstation)?

You seem to be thinking at too low a level. There isn't any "mocked H/W", only mocked behavior. The H/W simulation bodies give the information or have the effect expected of the devices given the state of the reality modeled in the environment pkg, but they need have no similarity to the real bodies, and usually don't. The device may be memory mapped, but there's no reason for the simulation to be. If access the device takes appreciable time, that's usually simulated using a delay. There's usually no reason to limit these parts of the S/W to the constraints of the target run time.

From: Adam Jensen <hanzer@riseup.net>
Date: Sun, 13 Nov 2016 17:00:19 -0500
Subject: Re: Getting started with bare-board development
Newsgroups: comp.lang.ada
> [...] I suppose software developers might be accustomed to ignoring time, the Turing machine/model-of-computation having no explicit representation of time. But you are correct, I very much retain the perspective of an electrical engineer and I most definitely think about the machine as something that exists in time.

Doesn't the Real Time Annex related parts of the run-time support system expect timing information from the hardware? (I am almost entirely guessing about this, I haven't yet finished reading the basic introductory materials on real-time programming).

It would probably help a lot to see a very basic little ("Hello, Real-Time World") example of [your development approach to] real-time software with a mocked hardware interface that can be executed directly on a workstation. I suppose the hardware could be as simple as a clock and maybe a counter or two. Maybe there could be some interrupts and two or three tasks that do something very simple. And maybe all of this could take place under the Ravenscar profile. Would that be a lot of effort to write and post?

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Mon, 14 Nov 2016 10:04:26 +0100
Subject: Re: Getting started with bare-board development
Newsgroups: comp.lang.ada
> [...] I think you are confusing things a bit. If you have computing hardware mocked you are doing simulation and the time is simulation time. If the peripheral hardware is real or partially real it is hardware-in-the-loop simulation (HIL). HIL is usually real-time. What people are saying is that HIL is much more cost efficient developing platform than some embedded board. Furthermore Ada is ideal for HIL because Ada software is portable. So you can develop almost everything on the PC and test almost everything in the loop. Then if some hardware (except the board itself) is too expensive or difficult to use, it can be simulated (mocked) in turn. Which is especially important when you want to test some catastrophic or improbable scenarios.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Tue, 15 Nov 2016 09:38:21 +0100
Subject: Re: Getting started with bare-board development
Newsgroups: comp.lang.ada
> [...] Typical developing process stages, at least in my area, is like this
1. Workstation Simulation time
   Application
   [HAL]
   Mock actuators/sensors
2. Workstation Hardware-in-the-loop, real-time
   Application
   [HAL]
   Real/Mock actuators/sensors
3. Embedded Target platform
   Application
   [HAL]
   Real/Mock actuators/sensors
Most of developing is done in #1 or #2. Most of testing in #2. #3 is limited to final integration tests.

In my domain (subcontractor for embedded SW in spacecraft) we typically use only one of the stages 1 and 2, not both. But otherwise our work is very similar.

> QEMU et al is not used, [...] Or unless you worry about compiler bugs being different in the native and cross compilers, or about platform-dependencies introduced by mistake in the Ada application. Endian-dependency is easily introduced by mistake if the application does a lot of communication with HW. Our targets are usually big-endian SPARC, but workstations are little-endian PCs.

Typically we do the final unit-testing runs both on workstations and on a target emulator, to settle such worries.

> Whatever OS/platform-dependent parts requiring test under an emulator, they are quite minuscule or non-existent if an OS is used.

We rarely (well, never) use an OS in embedded systems. Ravenscar or bare-board (zero runtime) is the norm for us. Though, there is a trend to have application-independent but domain-specific "execution platform" SW components which are like domain-specific OSs and can support various applications in this domain.

From: Adam Jensen <hanzer@riseup.net>
Date: Tue, 15 Nov 2016 12:32:55 -0500
Subject: Re: Getting started with bare-board development
Newsgroups: comp.lang.ada
> [...] "unless you are an OS developer" might be a key issue here. I have been thinking very much about device driver and run time kernel development for custom hardware. Ideally, what I am looking for (or trying to sort out) is a development methodology and tool-chain that fits into and extends the hardware development process.

It still seems to me that the ability to compartmentalize the emulated/simulated/HIL environment from the workstation's environment would be helpful, if not essential, at various stages of development and verification.
Does this make sense or is my view still somewhat askew?

> Whatever OS/platform-dependent parts requiring test under an emulator, [...] I can appreciate how it might be desirable for the workstation and the embedded target to provide the same OS/RTS environmental abstractions (for a software application developer's convenience), but the class of embedded software that I have in mind probably needs to have deep integration with the hardware, and the hardware definitely will have very deep

From: Simon Wright <simon@pushface.org>
Date: Mon, 14 Nov 2016 18:17:52 +0000
Subject: Getting started with bare-board development
Newsgroups: comp.lang.ada

> How does one develop and verify a board Support Package (device drivers, bootloader, etc.)?

The Cortex-M4 boards developed for e.g. PixRacer[1], based on STM32F427, support DFU[2] and JTAG.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Mon, 28 Nov 2016 15:32:34 -0600
Subject: Re: Generic private type declaration
Newsgroups: comp.lang.ada

> Do the various typical embedded platform profiles (e.g., Ravenscar) require any Run-Time System implementation or extension?

Yes, indeed! you can see AdaCore's implementation or extension?

For what it's worth, Janus/Ada is wrong in this case, there is no place where Y would ever be a tagged type, and thus it isn't a problem for this to be legal.

The general principle is that all Ada legality rules are rechecked in the specification of an instance, using the properties of the actual parameters. In most cases (for most rules), this doesn't matter (nothing changes), but there are cases where it matters and those are potentially contract-breaking. That's annoying, but it is an integral part of the Ada model for generics (the alternative would have been to use assume-the-worst rules in generic specifications, as is done in bodies, but that would make generics almost useless for tagged types -- no extensions could be done in generic specs under such a rule -- in particular, a mix-in generic would not be possible. So, yes, Dmitry, the language could strengthen contracts this way -- if one didn't care about usability (or compatibility!)

Adam Jensen wrote:

> What is the best way to use tags in Ada?

I use tags all the time. It's a shame that the Ada language doesn't allow for tags to be generic.

I will not go into all the details of how to use tags, but I will say that if you are using a tag, you should make sure that you are using it in a way that is consistent with the rest of your code.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Mon, 28 Nov 2016 15:32:34 -0600
Subject: Re: Ada 2012 Constraints (WRT an Ada IR)
Newsgroups: comp.lang.ada

So, with Ada 2012 we gained some really nice possibilities with the way to express constraints, the downside is that there's now a fairly wide range of ways to express constraints on types. Obviously these differences must be accounted for, but they are functionally equivalent, for example:

```
package P0 is
  subtype Natural range Natural’Succ(Natural’First)..Natural’Last;
end P0;
```

```
package P1 is
  subtype Integer range 1..Integer’Last;
end P1;
```

```
package P2 is
  with Static_Predicate => P1 in
  Integer_Last or else raise Constraint_Error;
end P2;
```

Now, these should be generally the same way of writing the same thing (i.e. "Positive") -- though I'm not completely certain that this is the case in terms of subtle semantics (am I missing something?) -- it certainly would be convenient if they were as then we could have an IR wherein the general form of a type-constraint is uniformly handled.

Comments? Insights?

---

Tagged/untagged Generic Parameter

From: Alejandro R. Mosteo <alejandro@mostseo.com>
Date: Fri, 25 Nov 2016 18:36:24 +0100
Subject: Generic private type declaration
Newsgroups: comp.lang.ada

I need some eyes on this error because I'm missing something basic. When compiling this code:

```ada
procedure B001_Tagged is
  generic
  type X is private;
  package Untagged is
    type Y is new X;
  end Untagged;
  package Ok is new Untagged (Integer);
  type Void is tagged null record;
  package Err is new Untagged (Void);
begin
  null;
end B001_Tagged;
```

I get in both GNAT 4.9.3 and GNAT GPL 2016 the following error:

```
 b001_tagged.adb:15:04: instantiation error at line 7
 b001_tagged.adb:15:04: type derived from tagged type must have extension
               gnatmake: "b001_tagged.adb" compilation error
```

I would expect that the view inside the generic package is untagged and so the type renaming in line 7 should be correct? Or I'm floundering with the generic parameter declaration?

From: Randy Brukardt <randy@rrsoftware.com>
Date: Mon, 28 Nov 2016 15:32:34 -0600
Subject: Re: Generic private type declaration
Newsgroups: comp.lang.ada

[Janus/Ada accepts the test case. —spare]

For what it's worth, Janus/Ada is wrong here (it probably isn't making the recheck of the instance; all of those have to be manually programmed and we pretty much only implemented the checks that we've seen in ACATS tests or in our own examples). The issue in this case is that type Y is a visible, tagged type outside of the instance. In that case, we can't allow a derivation without an extension (both for consistency reasons and I believe there also are semantic differences between tagged and untagged types).

But this is the one rule that we intentionally do not use the standard boilerplate about the legality rule also applying in the private part. Therefore, your example is legal so long as the derived type is not visible outside of the generic. Specifically, I think I didn't try it that:

```ada
generic
  type X is private;
  package Untagged is
    private
      type Y is new X;
    end Untagged;
```

In this case, there is no place where Y would ever be a tagged type, and thus it isn't a problem for this to be legal.

The general principle is that all Ada legality rules are rechecked in the specification of an instance, using the properties of the actual parameters. In most cases (for most rules), this doesn't matter (nothing changes), but there are cases where it matters and those are potentially contract-breaking. That's annoying, but it is an integral part of the Ada model for generics (the alternative would have been to use assume-the-worst rules in generic specifications, as is done in bodies, but that would make generics almost useless for tagged types -- no extensions could be done in generic specs under such a rule -- in particular, a mix-in generic would not be possible. So, yes, Dmitry, the language could strengthen contracts this way -- if one didn't care about usability (or compatibility!).
Byte Swapping

From: Jeffrey R. Carter
<jrcarter@acm.org>
Date: Fri, 2 Dec 2016 09:23:32 -0700
Subject: Byte Swapping
Newsgroups: comp.lang.ada

Recently on Ada-Comment there was a discussion of a GNAT aspect that changes the byte order of scalars. Brukardt said, "In the past, we have not be willing to require compilers to be able to generate byte swapping code." However, I think the standard has required compilers to generate byte-swapping code since Ada 83.

On a little-endian, twos-complement, byte-addressable machine, such as x86, we could say

```
Byte Size := constant := 8;
Word Size := constant := 2 * Byte Size;
```

```
type Byte is range -(2 ** (Byte Size - 1)) .. 2 ** (Byte Size - 1) - 1;
for Byte/Size use Byte Size;
type Word is range -(2 ** (Word Size - 1)) .. 2 ** (Word Size - 1) - 1;
for Word/Use use Word Size;
```

-- Signed types for Ada-83 compatibility

```
type Unswapped_Bytes is record
  MSB : Byte;
  LSB : Byte;
end record;
```

```
for Unswapped_Bytes use record
  MSB at 1 range 0 .. 7;
  LSB at 0 range 0 .. 7;
end record;
```

```
type Unswapped_Bytes'Size use Word Size;
  -- Default LE byte order: LSB
  -- at offset 0, MSB at offset 1
```

for Swapped_Bytes use record
  MSB at 0 range 0 .. 7;
  LSB at 1 range 0 .. 7;
end record;

for Swapped_Bytes'Size use Word Size;
  -- BE byte order: MSB at offset 0,
  -- LSB at offset 1

IIUC, type conversion between these two record types performs byte swapping. So, with

```
function To_Unswapped is
  new Unchecked_Conversion
  (Source => Word, Target => Unswapped_Bytes);
function To_Word is
  new Unchecked_Conversion
  (Source => Swapped_Bytes, Target => Word);
```

produces a Word with the bytes of W swapped. Barring any errors I've injected, this should be valid Ada 83 and all later version of the language.

From: Randy Brukardt
<randy@rrsoftware.com>
Date: Fri, 2 Dec 2016 13:28:04 -0600
Subject: Re: Byte Swapping
Newsgroups: comp.lang.ada

P0 and P1 are more likely to be optimized by a compiler (just because of the many years of history). Perhaps P3 will catch up, but I wouldn't hold my breath on that.

For Swapped_Bytes use record
  MSB at 0 range 0 .. 7;
  LSB at 1 range 0 .. 7;
end record;

for Swapped_Bytes'Size use Word Size;
  -- BE byte order: MSB at offset 0,
  -- LSB at offset 1

IIUC, type conversion between these two record types performs byte swapping. So, with

```
function To_Unswapped is
  new Unchecked_Conversion
  (Source => Word, Target => Unswapped_Bytes);
function To_Word is
  new Unchecked_Conversion
  (Source => Swapped_Bytes, Target => Word);
```

produces a Word with the bytes of W swapped. Barring any errors I've injected, this should be valid Ada 83 and all later version of the language.

From: Randy Brukardt
<randy@rrsoftware.com>
Date: Fri, 2 Dec 2016 13:28:04 -0600
Subject: Re: Byte Swapping
Newsgroups: comp.lang.ada

Sure, but that's typically implemented by doing a component-by-component assignment into a temporary (and the rep. clause is illegal for any by-reference type). That code is very slow, but that's OK because type conversions like this are rare in the language and occur rarely even when they are used. The suggested aspect would expect the compiler to be able to generate byte swapping code for a component reference. While that wouldn't be commonly used, when it is used, the code would have to be as efficient as possible as it would probably occur a lot. So the degree of effort is quite different.

To put it another way, a value type conversion "feels" expensive (one tries to avoid them), while a component access "feels" cheap (one does not try to avoid them), and the generated code needs to reflect that.

This is the crux of my (mild) objection to this idea: it makes component access for some record types quite expensive, and I doubt that most users would want to pay that cost regularly. The proper way to deal with data in the wrong byte-sex is to swap it as soon as possible and then process it in the native byte-sex. That means that the programmer has to be aware of it; trying to make a truly machine-independent format is somewhat of a mistake, as the code would be very slow on some targets. Don't see how that helps anything.
subprograms or in other tasks; Ada profile Ravenscar is in effect.


There was/is no/limited support for generic units.

From: Daniel King
<damaki.gh@gmail.com>
Date: Mon, 5 Dec 2016 14:19:31 -0800
Subject: Re: Ada features supported by SPARK 2014
Newsgroups: comp.lang.ada

> There was/is no/limited support for generic units.

To clarify, SPARK language versions 83, 95, and 2005 had no support for generic units at all (as far as I know), but the latest version of SPARK has full support for generic units (as long as they don't use any Ada language features that are not allowed in SPARK).

One consequence of this is that you can't actually run the proof tools on a generic unit directly, since it's not known if it's in SPARK or not until the unit is instantiated (for example, what if one of the generic parameters is an access type, which is not allowed in SPARK). So the proof tools are run on each *instantiation* of a generic unit.

I've used generics heavily in one of my SPARK projects - a SHA-3 hashing library: https://github.com/damaki/libkeccak

From: Daniel King
<damaki.gh@gmail.com>
Date: Mon, 5 Dec 2016 14:01:43 -0800
Subject: Re: Ada features supported by SPARK 2014
Newsgroups: comp.lang.ada

> [...] The SPARK User's Guide has a list of excluded Ada features that you should find useful for comparing SPARK and Ada capabilities:


In addition, for tasking features, SPARK is limited to the "Ravenscar profile", which is basically a set of restrictions on Ada's tasking features, to permit static analysis for formal verification.

A couple of links for SPARK that I find useful are the language reference manual (LRM) and user's guide:


From: Simon Wright
<simon@pushface.org>
Date: Tue, 06 Dec 2016 09:17:57 +0000
Subject: Re: Ada features supported by SPARK 2014
Newsgroups: comp.lang.ada

> [...] I found that - as soon as there's anything involving time - I couldn't work out how to specify flow (when I "fixed" one problem, another would pop up somewhere else; if I "fixed" that, the first would pop up again). So I left it up to the tool to infer flow for itself according to whatever arcane rules it wanted to (not really a satisfactory state of affairs for something that's supposed to increase my confidence in the code).

**Putting Contracts in the Standard Library**

From: Randy Brukardt
<randy@rrsoftware.com>
Date: Tue, 6 Dec 2016 17:09:28 -0600
Subject: Re: Ada 2012 Constraints (WRT an Ada IR)
Newsgroups: comp.lang.ada

Dmitry A. Kazakov wrote:

> Not at all. Constraint_Error is defined and *desired* behavior. Exceptions from pre-/post-conditions is undefined behavior.

Why?

Consider (part of) the procedure Delete in the Map container:

```
procedure Delete (Container : in out Map; Position : in out Cursor);
   -- If Position equals No_Element,
   -- then Constraint_Error is propagated.
```

As you say, this is defined and desired behavior.

Now, consider a better (IMHO) definition of this definition:

```
procedure Delete (Container : in out Map; Position : in out Cursor)
   with Pre => (if not Has_Element (Position) then raise Constraint_Error);
```

Here, instead of using an English comment to define this behavior, we've used an Ada precondition. So what's undefined about this? It's exactly the same semantics, and indeed my hope is that we update the RM to do this for all of the container routines in Ada 202x. (That makes the description of the primary function of the routine much easier to find, because it gets rid of all of the special conditions that start most of those descriptions. And it should make static analysis easier as well.)

IMHO, this is the only sensible use of a precondition (that is, it is some function of the parameters of the routine); it surely looks as defined as any if statement (which is what it would have to be written as in pre-Ada 2012).

**When and How to Report (Assertion) Errors**

From: Robert I. Eachus
<rieachus@comcast.net>
Date: Fri, 9 Dec 2016 01:12:15 -0800
Subject: Re: Ada 2012 Constraints (WRT an Ada IR)
Newsgroups: comp.lang.ada

Robert Eachus wrote:

> I would consider it a major bug to have a pragma Assert that could fail at runtime absent a hardware failure or some such. (Even though it would be turned off in production code.)

Simon Wright wrote:

> Yes. Though it's really up to the system engineers to decide on system behavior in the presence of software failure.

I picked this since it quotes me, rather than to pick on Simon. The important thing that is getting missed in this discussion is that there are lots of uses for software. I spent most of my career working on software for radars, planes, and missiles, where production software means the stuff that flies--or gets installed on a mountain with no software or system engineer in easy reach.

Sometimes that means that crashing the software (well a controlled crash that turns off the radar then restarts from the beginning) is the right safety feature. But in an aircraft you leave it to the pilot to shut the engines down. Yes, the engine might be about to tear itself into little pieces in five minutes--but this may be the only working engine that will get you to the airport. I remember one incident where the mechanic didn't put the o-rings on the (new) oil plugs. The pilot shut down the center engine for low oil pressure, and headed back to Palm Beach. Then the other two engines had oil pressure warnings. He ran them as long as possible, glided until just above the waves--and restarted the center engine. Safe landing, barely.

But notice that the cockpit crew should never end up fighting the software warning system. If it doesn't help, cut the warning. Read about what happened to the Quantas A380, when an engine failed and cut some of the wires in the wing. Telling the cockpit crew IN THE AIR that the warning system is NOT working, cut some of the wires in the wing. The cockpit crew IN THE AIR that thus and so is not reporting every few seconds is NOT helpful. On the ground? Fine, if it is a deadline issue. (Hmm. Not clock deadlines, deadlines as in the plane won't fly.)

Why do I remember such incidents? And why did I consider it important for me to know about them. It all comes back to this issue. Who are your diagnostics and exceptions expected to help?
To bring it back here, as far as I am concerned, the Assert feature makes it easier to ensure that debug only code does not end up causing real accidents. Exceptions often need to be handled in production code, but such exceptions should usually be wrapped closely in specific handlers.

Oh, and that handler around one line might as well say “when others.” There may be some code you don’t see that could result in an unexpected exception, say “Device_Error” not “Use_Error” when reading from a file, but the behavior to deal with it is the same.

From: Randy Brukardt
<randy@rrsoftware.com>
Date: Mon, 5 Dec 2016 16:06:29 -0600
Subject: Re: Ada 2012 Constraints (WRG an Ada IR)
Newsgroups: comp.lang.ada

Stefan Lucks wrote:

> On the other hand, there are systems, where a malfunction is worse than no function (e.g., a secure router – better no communication than allowing attackers to pass through the security perimeter). In such cases, it may be wise do perform Assert-checking even in production executables.

My opinion is that the vast majority of systems are in this category, especially if one considers each task (here using “task” in its English meaning as a block of work) separately. Every system that I’ve personally worked on has been in this category (anti-spam filter – a bug causes a message to be quarantined, which provides a path to reproducing the bug; web server – a bug causes nothing useful to be returned, better than allowing a security hole and returning, say, a password file; Ada compiler - a bug usually causes the compiler to crash rather than producing incorrect code that might cause problems; Claw Builder – a bug usually causes buggy generated code, which breaks the contract with the user).

Most such systems need an “others” handler to ensure that one failing task (again, English meaning) doesn’t cause the entire system to fail. Such handlers need some sort of reporting system so they’re not silently covering bugs.

I also tend to disagree about suppressing/ignoring checks and predicates. In my experience, if a check or predicate or assertion is too expensive to run in the production system, it’s also too expensive to run in testing. Simple checks should never be turned off – a visible bug is always better than an invisible bug. (Moreover, compilers are always getting better about eliminating such checks, in which case many checks aren’t made at all.) Expensive checks, if one has to have them at all, need to be managed separately from assertions/constraints/predicates – one would only want to turn them on if all else has failed, and that clearly needs to be separate from the suppress/ignore mechanism. (Most of the systems I’ve worked on have a runtime management setup for tracing/assertions, where they get managed by functional areas as needed for figuring out the problem at hand.)

**Default Initialization**

*From: Simon Wright*

.isTrue: &true

> <simon@pushface.org>

*Date: Sun, 23 Dec 2016 09:23:14 +0000*

*Subject: Default values*

*Newsgroups: comp.lang.ada*

*Given*

with Ada.Real_Time;

.package Sbus.IMU is

subtype Radians_Per_Sekond is Float;

subtype Acceleration is Float;

subtype Milligauss is Float;

.type Update is record

Magnetometer_Data_Present : Boolean := False;

Time_Valid : Ada.Real_Time.Time;

Gx, Gy, Gz : Radians_Per_Sekond;

Ax, Ay, Az : Acceleration;

case Magnetometer_Data_Present is

when True =>

Mx, My, Mz : Milligauss;

when False =>

null;

end case;

end record;

protected Updater is

.entry Get_New_Data (Data : Update);

.entry Get_Latest_Data (Data : out Update);

.private New_Data_Present : Boolean := False;

Latest_Data : Update := (others => <>);

end Updater;

end Sbus.IMU;

is the line

Latest_Data : Update := (others => <>);

legal? If so, what does it mean? (I’ve looked at the ARM for Record Aggregates, 4.3.1, and Record Types, 3.8, and am no wiser). I do realise that I need to put some default initializations in (or else supply a proper initialization for Latest_Data!)

*From: Jeffrey R. Carter*

.isTrue: &true

> <jrcarter@acm.org>

*Date: Sun, 25 Dec 2016 12:53:41 +0100*

*Subject: Re: Default values*

*Newsgroups: comp.lang.ada*

> [...] Yes, it’s legal. It means the same thing as

Latest_Data : Update;

*From: Randy Brukardt*

.isTrue: &true

> <randy@rrsoftware.com>

*Date: Tue, 27 Dec 2016 17:32:57 -0600*

*Subject: Re: Default values*

*Newsgroups: comp.lang.ada*

> [...] Right. Each component is “initialized by default”, which for Float and subtypes thereof means that they’re uninitialized. Probably not what you want, but well-defined.

**Object Renaming: Copy or Reference?**

*From: Georg Bauhaus*

.isTrue: &true

> <bauhaus@futureapps.de>

*Date: Sun, 22 Jun 2017 09:27:44 +0100*

*Subject: Does object renaming allow the view to be a copy?*

*Newsgroups: comp.lang.ada*

A SO answer (41746244) has given rise to the question of whether or not a compiler implementer may make a renamed object a copy of the original. (Layman’s assumptions from LRM 3.1(7), 8.5.1),

So, is the following program, modifying components of and array, ever allowed to raise Renaming_Is_Copying?

.with System;

procedure Renaming is

Renaming_Is_Copying : exception;

type R is record

A, B : Integer;

end record;

.type List is array (Natural range <>) of R;

.Stuff : List := (10 .. 20 => R(A => 2, B => 3));

begin

for K in Stuff'Range loop

.declare

-- -- Does Ada allow a compiler to

-- make X be a copy?

-- X : R renames Stuff (K);

.use type System.Address;

.begin

if X'Address = Stuff (K)'Address then

X.A := X.B;

else

raise Renaming_Is_Copying;

.end if;

.end loop;

.end Renaming;

*From: Christoph Karl Walter Grein*

.isTrue: &true

> <christ-usch.grein@-online.de>

*Date: Sun, 22 Jun 2017 08:26:39 -0800*

*Subject: Does object renaming allow the view to be a copy?*

*Newsgroups: comp.lang.ada*
> [...] 
3.1(7) ...a renaming_declaration is an example of a declaration that does not define a new entity, but instead defines a view of an existing entity (see 8.5)...

So how can you think a compiler may create a copy?

8.5(3) The elaboration of a renaming_declaration evaluates the name that follows the reserved word renames and thereby determines the view and entity denoted by this name (the renamed view and renamed entity).

[A name that denotes the renaming_declaration denotes (a new view) of the renamed entity.]

Same here. You get a new view of the entity.

BTW: A renaming is not a macro. Thus the following fragment does not change X:

\[ X : T \text{renames} Y (I); \]

\[ 1 : = 1 + 1; \]

From: Simon Wright

Date: Sun, 22 Jan 2017 17:37:10 +0000
Subject: Re: Does object renaming allow the view to be a copy?
Newsgroups: comp.lang.ada

> [...] 
I did suggest that it would have to be a crazy implementer who did this.

> [...] So how can you think a compiler may create a copy?

I think that if the object isn't limited and the operations done on it don't alter its contents you'd be hard put to it to tell the difference, that's all.

But like I said, crazy. Under the hood, any sensible person would have a reference to the original object.

From: Christoph Karl Walter Grein

Date: Mon, 23 Jan 2017 02:49:04 -0800
Subject: Re: Does object renaming allow the view to be a copy?
Newsgroups: comp.lang.ada

> Some packed Boolean, not at the storage element margin?

Of course not in cases like that:

```ada
type Set is array (Index) of Boolean with Packed;
My_Set : Set;
```

\[ My_{Element\_Presence} : Boolean \text{renames} My\_Set (I); \]

\[ My\_Set (I) = \text{not} My\_Set (I); \]

\[ My_{Element\_Presence} := \text{not} My_{Element\_Presence}; \]

Under the hood, the same packing and unpacking has to be performed. A simple reference is impossible in this case.

From: Randy Brukardt

Date: Mon, 23 Jan 2017 14:40:08 -0600
Subject: Re: Does object renaming allow the view to be a copy?
Newsgroups: comp.lang.ada

> [...] 
Logically, the item is not a copy. How the compiler implements that, however, is its business.

> [...] So, is the following program, modifying components of and array, ever allowed to raise Renaming_Is_Copying?

Of course. The program has nothing to do with copying that I can see.

> if X'Address = Stuff (K)'Address then
>

The meaning of X'Address is implementation-defined (as someone said, consider what happens if X is allocated in a register). It's best if its use is limited to the sort of low-level purposes for which it was defined (that is, handling memory-mapped hardware). Note in particular 13.3(16): if the objects in question aren't "aliased", the result of 'Address may not be "useful".

If the objects are aliased, then you don't need to use 'Address to get the answer to your question:

```ada
if X'Access = Stuff (K)'Access then
```

would answer your question (but you might need to declare an appropriate access type somewhere). Note that the compiler would strip off any funny business for this latter case.

IMHO, 'Address should only appear in a program that is interfacing to some memory-mapped entity; else use some form of 'Access (or 'Unchecked_Access).

From: Robert I. Eachus

Date: Tue, 24 Jan 2017 08:06:40 -0800
Subject: Re: Does object renaming allow the view to be a copy?
Newsgroups: comp.lang.ada

> for X'Address use at mod 4;

Looks like you are trying to set Alignment which Ada 83 didn't have. Set alignment directly, so your reader knows what you're doing.

> I also seem to recall that a lot of the NUMWG work uses address clauses when pulling floating point numbers apart—and putting them back together.

Ada 95 says that Unchecked_Conversion can be by-reference, so the performance reason for not using UC in this case doesn't exist (assuming a friendly implementer). UC also doesn't force a compiler to abandon many useful optimizations (or do horrible analysis before allowing it).

> Are all of these deprecated now?

All but the last can be done better with other constructs, IMHO. For the last, it's case-by-case what the best approach would be: you might need 'Address.

Implementing a Dynamic Type System

From: Randy Brukardt

Date: Tue, 24 Jan 2017 15:21:41 -0600
Subject: Re: Dynamic type system for Ada
Newsgroups: comp.lang.ada

Victor Porton wrote:

> All I ask is just an Ada type which would be so flexible that could store any kind of a value (just like as a variable in a dynamic language).

I'd suggest that it be defined as an abstract tagged type, with each of the other kinds of things as a derived tagged type (one for holding integers, one for holding floats, etc.). Then one could get a "variable that could hold anything" by instantiating the Unbounded_Holder container (and that would open uses of the other kinds of containers as well). That way, the package wouldn't have to reinvent all of the memory management stuff that's already in the containers. Plus, if one organized the hierarchy similar to the chart in
under "Root_Numeric", for instance), but you are correct that they are needed (and the reverse as well, to give a way to import values, esp. literals).

And I have no idea how to deal with most user-defined types (enumerations, records, tasks, etc.) in such a scheme. (The interface of arrays is simple enough that I can imagine some mechanism to deal with a subset of them.)

> Otherwise you have to explicitly convert (upcast) to specific instance derived from Abstract_Variable which is much worse.

That's how all of my hierarchies work. I doubt that I'd call it "much worse"; it avoids "god-classes" and allows most of the checks to be made statically (you can't use operations of the wrong type). It's often easy to apply those upconverts when parameter passing, so they don't end up that wide-spread.

But I definitely agree that this is a case where there is always going to be a "bump under the carpet" (as Tucker Taft liked to say during Ada 9x) [you can move the bump to different places under the carpet, but you can't get rid of it (at least without total carpet replacement, which is where this analogy breaks down - but I digress)]. There isn't going to be a totally clean solution.

> P.S. I intended to use that on top of a stream exchange, but the interface is so heavy that it adds no advantage to direct reading the target object from the stream.

I suspect this sort of thing would work pretty well for a hierarchy of numeric types, not so well if other kinds of types are included. There's just not enough overlap of operations for it to make much sense in the general case.

Explicitly Distinguishing Between Static and Dynamic Predicates

From: Randy Brukardt
Date: Wed, 8 Feb 2017 17:37:04 -0600
Subject: Re: I am not understanding user defined exceptions
Newsgroups: comp.lang.ada
Simon Wright wrote:

> [...] I have to say that the GNAT approach (just Predicate, the compiler knows which one to choose) seems the right one to me!

The GNAT approach causes a very significant maintenance hazard: [...] I don't buy it (as you may remember, because I said so in an ARG meeting). The reason is that all these horrible things you mention can happen when you change a static predicate to a different static predicate. E.g.:

```
subtype S is Integer with Static_Predicate => S >= 0;
```

If you change it to:

```
subtype S is Integer with Static_Predicate => S >= 1;
```

client code is just as likely to break as if you changed it to a dynamic predicate. So what? Any time you change the visible part of a widely used library unit, you have to be careful about breaking clients. Note that the first S above is exactly the same as:

```
subtype S is Integer range 0 .. Integer_Last;
```

And we don't bother to mark that as a static range. You could change it to "0 .. Dynamic_Value", and break clients.

> ...and they'd have no understanding of why (or any hope of fixing it).

Now that's REALLY overstating the case. Anybody who can read Ada code can understand why (and hope to fix it).

> ... By declaring your intent as static or dynamic, clients can properly use the predicate subtype and you as the maintainer can't break their expectations without at least realizing that there is a potential problem.

The above argument proves that to be wrong -- the maintainer CAN break...
clients DESPITE the fact that the predicate is marked Static .
> This is especially true as many expressions that *seem* simple aren't allowed as static predicates [...] 

In the same sense that a static constant is quite different from a dynamic one. For example you can say "when X =>" in a case statement if X is static. And if somebody changes X to a different static value, or to a dynamic value, the case statement will become illegal.

From: Randy Brukardt
<randy@rrsoftware.com>
Subject: Re: I am not understanding user defined exceptions
Date: Thu, 9 Feb 2017 15:47:49 -0600
Newsgroups: comp.lang.ada
Organization: JSA Research & Innovation

> [...] 
It's surely possible to have client code break, but it is far less likely if the client used the abstraction as intended.

When one goes from a static to a dynamic predicate, all client code using for loops and case statements will fail. Period.

When one changes the details of a static predicate, the only code that might fail (statically) is use in a case statement. And such problems generally point out issues with the use of the abstraction (presuming of course that the abstraction was sensibly defined in the first place).

For instance, if one had used static predicates to partition a type:

```ada
subtype Part1 is Integer with
  -- Static Predicate => Part1 >= 0;
subtype Part2 is Integer with
  -- Static Predicate => Part2 < 0;
```

then a case statement using the partitions would continue to work if what exactly is in each partition is changed.

If, on the other hand, a case statement assumed which partition a particular value belongs, then it might fail if that is changed down the road. But that clearly broke the abstraction, so the failure seems like a good thing in such a case.

Clearly, there are far fewer possibilities of failure when one changes a values in a static predicate than when one changes from a static predicate to a dynamic one. So that argument does not hold much water.

> So what? Any time you change the visible part of a widely used library unit, you have to be careful about breaking clients.

The more help that we can give the maintainer to prevent such problems, the better. This is an area where Ada does not do very well, as things that usually don't matter (parameter subtypes, for instance) come into play in some obscure rules and thus virtually any change to a specification will break some code. This is a serious problem; once a library gets into wide use its specification is effectively encased in amber. You have to start over to make any significant changes.

I don't see any point in making new features be even worse for that than the existing ones. Luckily, the ARG agreed.

> Note that the first S above is exactly the same as:
>
> subtype S is Integer range 0 .. Integer'Last;
>
> And we don't bother to mark that as a static range. You could change it to "0 .. Dynamic_Value", and break clients.

Right. And as I mentioned elsewhere, we should have done that. (Actually, what we should have done is required to mark dynamic subtypes, as they're not very likely. Definitely too late for that, though.)

> [...] 
It's a bit of an overstatement, but it's close: "no understanding why" => there's no indication in the source code (if you use GNAT's evil "predicate") and the rules for when it is static are not intuitive. There's almost no chance that I would think of such a predicate change when I first saw such a problem, I would waste a lot of time looking elsewhere first.

And there's no hope of fixing it because it happened in reusable code that they have no control over. They've unintentionally depended on a property that the library did not intend to make stable. The only fix is to totally replace the failing constructs with different ones (and in the case case :-) losing the completeness checks at the same time.

> [...] 
Only clients that misused the abstraction in case statements. (For loops won't break, at least not statically -- and if the loop depends on the exact values that it iterates over, they've again missed the point of the abstraction.) I'm definitely less concerned about breakage in iffy code than I am about breakage that occurs in perfect code.

Being forced to replace:

```ada
  case Something is
    when Part1 => ...
    when Part2 => ...
  end case;
```

with a less safe if statement just because someone screwed up seems horrible to me.

> [...] In the same sense that a static constant is quite different from a dynamic one. [...] Right, and I view this as a significant flaw in Ada. If I was designing a language from scratch, these would clearly be marked as different things. Most likely:

X : Integer := ...; -- Static constant
X : constant Integer := ...; -- Non-static constant
X : variable Integer := ...; -- Variable.

Since the default should be the safest thing. (Note that an initializer would be required for all of these; <> could be used to explicitly mark it as default-initialized.) The same with subtypes (anything that has a name).

We can't make this change to Ada for obvious reasons, but surely two wrongs do not make a right.

From: Randy Brukardt
<randy@rrsoftware.com>
Subject: Re: I am not understanding user defined exceptions
Date: Fri, 10 Feb 2017 15:09:34 -0600
Newsgroups: comp.lang.ada

Robert A Duff wrote:

> Well, at least we agree on one thing. It's odd that Ada got it right for parameters ("in" is the default), but got it wrong for object declarations (variable is the default).

> I'd be happy with:

> X : Integer := ...; -- constant

> X : var Integer := ...; -- variable

(1) Ada doesn't generally use abbreviations, thus "var" isn't a likely keyword.

(2) It's important that all properties that clients can depend upon are declarable, so clients aren't depending on accidental properties. (That's the whole principle behind private types.) Static is such a property, so it should be declarable (not necessarily have to be declared in all cases). [Indeed, it should be possible to declare static private types - a whole different kettle of fish.] Similarly, it would be nice if there was a way to prevent people from depending upon the subtype profile of a subprogram (so that it can later be changed if necessary). There's probably other such properties (one would like to include formal parameter names in this sort of restriction, but that would prevent named calls which would be evil.)

From: Randy Brukardt
<randy@rrsoftware.com>
Subject: Re: I am not understanding user defined exceptions
Date: Mon, 13 Feb 2017 17:20:44 -0600
Newsgroups: comp.lang.ada

Dmitry A. Kazakov wrote:

> Why is static a property? [...] 

It's a property because the language design depends on it so heavily. Perhaps
one could design a language that enforced Legality Rules without having some property that controls whether that is possible or not, but I can't quite imagine how. (Only a language that enforced no rules until runtime could work that way, IMHO, and that eliminates most of the benefits of strong typing and early error detection.)

If static was a declarable property, I'd also make it possible to declare user-defined static things, and apply that to all types. For instance, it should be possible to have static System.Address values, static Complex values, and the like. It wouldn't be limited to just whatever the language designers could define.

From: Robert A Duff <bobduff@TheWorld.com>
Subject: Re: I am not understanding user defined exceptions
Date: Thu, 09 Feb 2017 14:15:10 -0500
Newsgroups: comp.lang.ada
Organization: The World Public Access UNIX, Brookline, MA

Simon Wright wrote:
> Good, but I have to admit the strength of Randy's point re: maintainability. I don't share Randy's concerns about maintainability.

> ... Would it be possible for GNAT have a diagnostic option to state whether explicit Static_Predicate would be OK? Sure, but I'm not likely to implement any such thing, given my opinion expressed above. I recommend you use "Predicate =>" unless you want to be portable to non-GNAT compilers (or request the other compilers to mimic GNAT).

To convince me otherwise, you'll have to explain why we don't say:

```
X : static constant Integer := 100;
static subtype S is Integer range 1..100;
> ...(you could tell me to just try Static_Predicate first!)
```

Sure, you can do that if you like. To me, "Static_" is just noise (or necessary for portability). This kind of inconsistency is a flaw in the design of Ada.

From: Randy Brukardt <randy@rrsoftware.com>
Subject: Re: I am not understanding user defined exceptions
Date: Thu, 9 Feb 2017 15:06:56 -0600
Newsgroups: comp.lang.ada
Organization: JSA Research & Innovation

>(1) We don't have it because Ichibiah left it out. I think that was a mistake.
>(2) For the constant, you can declare it to be static and I usually do:

```
X : constant := 100;
```

(3) Ada really does need such a capability. Staticness determines many Legality Rules and it can be a critical property when exposed in a reusable library. Unintentionally eliminating it can be disastrous for clients. I'd definitely be in favor of adding the "static" keyword as you have it above. (It would have to be optional, sadly, but of course a restriction could "fix" that.)

But arguably it is much less likely to be changed by accident (although it has happened to me repeatedly) -- almost every operation that you'd expect to be static can be static (the main exception being representation attributes like Size). That's definitely not the case with Static_Predicates.

(4) <rant> GNAT effectively nullifying a carefully considered and heavily debated decision in the ARG because a couple of people didn't like it is about the most evil behavior that an implementer could take. It's the sort of thing that makes me wonder why AdaCore is so invested in the Standards process if it just going to ignore the result when convenient. </rant>

> [...] To me, "Static_" is just noise (or necessary for portability). This kind of inconsistency is a flaw in the design of Ada.

You're just plain wrong, considering that we discussed this extensively in the ARG and the "maintenance is important" position carried the day. The flaw is that you can't declare most things static to avoid future problems.

P.S. Side note: as with "constant", "static" probably should have been the default. It really should be necessary to declare something non-static or variable. That would be possible in a totally brand-new language, but sadly not in Ada or even an improved Ada-like language.
Complete Ada Solutions for Complex Mission-Critical Systems

- Fast, efficient code generation
- Native or embedded systems deployment
- Support for leading real-time operating systems or bare systems
- Full Ada tasking or deterministic real-time execution

Learn more by visiting: ptc.com/developer-tools
Conference Calendar

Dirk Craeynest
KU Leuven. Email: Dirk.Craeynest@cs.kuleuven.be

This is a list of European and large, worldwide events that may be of interest to the Ada community. Further information on items marked ♦ is available in the Forthcoming Events section of the Journal. Items in larger font denote events with specific Ada focus. Items marked with ☑ denote events with close relation to Ada.

The information in this section is extracted from the on-line Conferences and events for the international Ada community at: http://www.cs.kuleuven.be/~dirk/ada-belgium/events/list.html on the Ada-Belgium Web site. These pages contain full announcements, calls for papers, calls for participation, programs, URLs, etc. and are updated regularly.

2017

☑ April 03-06 The Art, Science, and Engineering of Programming Conference (Programming’2017), Brussels, Belgium. A new conference, with an associated gold open access journal, created with the goal of placing the art of programming in the map of scholarly works. Topics include: The Art (knowledge and technical skills acquired through practice and personal experiences; examples include libraries, frameworks, languages, APIs, programming models and styles, programming pearls, and essays about programming); Science - empirical (knowledge and technical skills acquired through experiments and systematic observations; examples include user studies and programming-related data mining); Science - theoretical (knowledge and technical skills acquired through mathematical formalisms; examples include formal programming models and proofs); Engineering (knowledge and technical skills acquired through designing and building large systems and through calculated application of principles in building those systems; examples include measurements of artifacts' properties, development processes and tools, and quality assurance methods). Areas include: general-purpose programming, distributed systems programming, parallel and multi-core programming, security programming, interpreters, virtual machines and compilers, modeling and modularity, testing and debugging, program verification, programming education, programming environments, etc.

April 03 1st International Workshop on Programming Technology for the Future Web (ProWeb’2017). Topics include: programming technology (i.e., frameworks, libraries, programming languages, program analyses and development tools) for implementing web applications and for maintaining their quality over time, as well as experience reports about the use of state-of-the-art programming technology.

April 03-07 32nd ACM Symposium on Applied Computing (SAC’2017), Marrakech, Morocco.

☑ April 03-07 Track on Object-Oriented Programming Languages and Systems (OOPS’2017). Topics include: aspects and components; code generation, and optimization; distribution and concurrency; formal verification; integration with other paradigms; interoperability, versioning and software evolution and adaptation; language design and implementation; modular and generic programming; runtime verification; secure and dependable software; static analysis; testing and debugging; type systems; virtual machines; etc.

☑ April 03-07 Track on Programming Languages (PL’2017). Topics include: compiling techniques, domain-specific languages, garbage collection, language design and implementation, languages for modeling, model-driven development, new programming language ideas and concepts, practical experiences with programming languages, program analysis and verification, programming languages from all paradigms, etc.

☑ April 03-07 Track on Software Verification and Testing (SVT’2017). Topics include: new results in formal verification and testing, technologies to improve the usability of formal methods in software engineering, applications of mechanical verification to large scale software, model checking, correct by construction development, static and run-time analysis, analysis methods for dependable systems, software certification and proof carrying code, real world applications and case studies applying software verification, etc.
April 03-07  12th Track on Dependable and Adaptive Distributed Systems (DADS'2017). Topics include: Dependable, Adaptive, and trustworthy Distributed Systems (DADS); middleware for DADS; modeling, design, and engineering of DADS; foundations and formal methods for DADS; etc.

April 05-07  1st IEEE International Conference on Software Architecture (ICSA'2017), Gothenburg, Sweden. Topics include: model driven engineering for continuous architecting; component based software engineering and architecture design; re-factoring and evolving architecture design decisions and solutions; architecture frameworks and architecture description languages; preserving architecture quality throughout the system lifetime; software architecture for legacy systems and systems integration; architecting families of products; software architects roles and responsibilities; training, education, and certification of software architects; industrial experiments and case studies; etc.

April 18-21  23rd IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'2017), Pittsburgh, PA, USA. In conjunction with CPSWeek'2017. Topics include: applications, tools, and run-time software for real-time systems; basic methodologies, algorithms, and analyses that are applied to real systems to solve specific problems; hardware/software co-design, integration methodologies, design-time tools and architectures for modern embedded systems for real-time applications; etc.

April 18-21  8th ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS'2017), Pittsburgh, PA, USA. In conjunction with CPSWeek'2017. Topics include: security of cyber-physical systems (CPS), mechanism design for CPS, model-based design and verification of CPS, etc.

April 22-26  8th ACM/SPEC International Conference on Performance Engineering (ICPE'2017), L'Aquila, Italy.

April 22-29  20th European Joint Conferences on Theory and Practice of Software (ETAPS'2017), Uppsala, Sweden. Events include: ESOP (European Symposium on Programming), FASE (Fundamental Approaches to Software Engineering), FOSSACS (Foundations of Software Science and Computation Structures), POST (Principles of Security and Trust), TACAS (Tools and Algorithms for the Construction and Analysis of Systems), SV-COMP (Competition on Software Verification).

April 22  14th International Workshop on Formal Engineering approaches to Software Components and Architectures (FESCA'2017). Topics include: (semi-)formal techniques and their application that aid analysis, design and implementation of software applications; formal modelling of component-based, timed and hybrid systems; temporal properties and their formal verification; interface compliance and contractual use of components; static and dynamic analysis; industrial case studies and experience reports; etc.

© April 29  10th Workshop on Programming Language Approaches to Concurrency and communication-centric Software (PLACES'2017). Topics include: the general area of programming language approaches to concurrency, communication and distribution, such as design and implementation of programming languages with first class support for concurrency and communication; concurrent data types, objects and actors; verification and program analysis methods for concurrent and distributed software; high-level programming abstractions addressing security concerns in concurrent and distributed programming; multi- and many-core programming models, including methods for harnessing GPUs and other accelerators; integration of sequential and concurrent programming techniques; programming language approaches to web services; etc.

April 26-28  7th International Conference on Fundamentals of Software Engineering (FSEN'2017), Tehran, Iran. Topics include: all aspects of formal methods, especially those related to advancing the application of formal methods in the software industry and promoting their integration with practical engineering techniques; software specification, validation, and verification; software architectures and their description languages; integration of formal and informal methods; component-based software systems; model checking; software verification; CASE tools and tool integration; industrial applications; etc.

April 28-29  12th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE'2017), Porto, Portugal. Topics include: application integration technologies, architectural design and frameworks, component-based software engineering, formal methods, model-driven engineering, reverse software engineering, software and system complexity, software and systems development methodologies, software and system quality management, software patterns and refactoring, software product line engineering, software process improvement, etc.
May 16-18 20th IEEE International Symposium On Real-Time Computing (ISORC'2017), Toronto, Canada. Topics include: object/component/service-oriented real-time distributed computing (ORC) technology, programming and system engineering (real-time programming challenges, ORC paradigms, languages, ...), trusted and dependable systems, system software (real-time kernels, middleware support for ORC, extensibility, synchronization, scheduling, fault tolerance, security, ...), applications (medical devices, intelligent transportation systems, industrial automation systems, Internet of Things, embedded systems, ...), system evaluation (performance analysis, monitoring & timing, dependability, ...), cyber-physical, etc.

May 16-18 9th NASA Formal Methods Symposium (NFM'2017), Moffett Field, California, USA. Topics include: identify challenges and provide solutions for achieving assurance for critical systems; model checking; static analysis; model-based development; software and system testing; safety assurance; fault tolerance; compositional verification; design for verification and correct-by-design techniques; applications of formal methods in the development of autonomous systems, cyber-physical, embedded, and hybrid systems, ...; use of formal methods in assurance cases, automated testing and verification, ...; etc.

May 20-28 39th International Conference on Software Engineering (ICSE'2017), Buenos Aires, Argentina.

May 20-28 5th FME Workshop on Formal Methods in Software Engineering (FormaliSE'2017). Topics include: integration of FMs in the software development life cycle, ability of FMs to handle real-world problems, scalability of FM applications, FMs in a certification context, "lightweight" or usable FMs, rigorous software engineering approaches and their tool support, case studies, formal approaches in the development of cyber-physical systems, etc.

May 22-23 20th Ibero-American Conference on Software Engineering (CIbSE'2017), Buenos Aires, Argentina. Event includes Software Engineering Track (SET).

May 22-23 12th IEEE International Conference on Global Software Engineering (ICGSE'2017), Buenos Aires, Argentina. Topics include: strategic issues in distributed development, tools and infrastructure support, software architecture and design, security and privacy, lean and agile development, etc.

May 22-26 18th International Conference on Agile Software Development (XP'2017), Cologne, Germany. Theme: "Uncovering better ways of developing software". Topics include: tools and techniques for agile development, empirical studies and evaluations, adopting and adapting agile and lean in large projects and organizations, etc. Event includes workshops on Agile Development of Safety-Critical Software (ASCS'2017), on Managing Technical Debt (MTD'2017), etc.

May 29-31 16th International Conference on Software Reuse (ICSR'2017), Salvador, Brazil.

May 29 - Jun 02 31st IEEE International Parallel and Distributed Processing Symposium (IPDPS'2017), Orlando, Florida, USA.


June 12-16 29th International Conference on Advanced Information Systems Engineering (CAiSE'2017), Essen, Germany. Theme: "Digital Connected World - Informed, Disruptive Business Transformation". Topics include: methods, models, techniques, architectures and platforms for supporting the engineering and evolution of information systems and organizations in the digital connected world.


June 18-23 31st European Conference on Object-Oriented Programming (ECOOP'2017), Barcelona, Spain. Topics include: theory, design, implementation, optimization, and analysis of programs and programming languages; innovative and creative solutions to real problems, and evaluations of existing solutions in ways that shed new insights; etc. Deadline for submissions: April 15, 2017 (student volunteers), April 20, 2017 (workshop papers).

June 19-23 11th ACM International Conference on Distributed Event-Based Systems (DEBS'2017), Barcelona, Spain. Co-located with PLDI and ECOOP'2017. Topics include: real-time analytics, security, reliability and resilience, embedded systems, enterprise application integration, distributed programming.
availability, scalability, etc. Deadline for submissions: April 7, 2017 (Grand Challenge solutions), April 29, 2017 (posters, demos, Doctoral Workshop papers).

June 27-30

29th Euromicro Conference on Real-Time Systems (ECRTS'2017), Dubrovnik, Croatia. Topics include: scheduling design and analysis, real-time operating systems, hypervisors and middlewares, virtualization and timing isolation, contention-aware scheduling of multi-core systems, heterogeneous real-time systems, mixed-criticality design & assurance, WCET analysis, real-time networks and predictable communication protocols, realistic power/energy/thermal models and algorithms, network/system-on-chips and massively parallel devices, modelling and/or formal methods, industrial use-cases and RT applications, tools, compilers and benchmarks for embedded systems. Deadline for submissions: April 28, 2017 (Work-in-Progress papers).

July 04-08


July 05-07

International Conference on Software and Systems Process (ICSSP'2017), Paris, France. Topics include: mining software/business process repositories (including code, bug trackers, etc.) to improve processes; empirical evidence of the effectiveness of agile/lean practices and approaches in software systems development and evolution; process issues in developing evolving software systems; processes for cutting-edge software technologies, including (but not limited to) multi-core technologies; empirical studies and experience reports, encompassing complete or parts of software and systems development lifecycle; etc.

July 17-21


July 19-20

11th International Conference on Tests And Proofs (TAP'2017). Topics include: many aspects of verification technology, including foundational work, tool development, and empirical research; the connection between proofs (and other static techniques) and testing (and other dynamic techniques); verification and analysis techniques combining proofs and tests; program proving with the aid of testing techniques; deductive techniques to support testing: generating testing inputs and oracles, supporting coverage criteria, and so on; program analysis techniques combining static and dynamic analysis; testing and runtime analysis of formal specifications; model-based testing and verification; using model checking to generate test cases; testing of verification tools and environments; applications of testing and proving to new domains, such as security, configuration management, and language-based techniques; case studies, tool and framework descriptions, and experience reports about combining tests and proofs; etc.

July 22-28

29th International Conference on Computer-Aided Verification (CAV'2017), Heidelberg, Germany. Topics include: theory and practice of computer-aided formal analysis and synthesis methods for hardware and software systems, algorithms and tools for verifying models and implementations, specifications and correctness criteria for programs and systems, deductive verification using proof assistants, program analysis and software verification, verification methods for parallel and concurrent systems, testing and run-time analysis based on verification technology, applications and case studies in verification and synthesis, verification in industrial practice, formal models and methods for security, etc.

July 26-28

IEEE International Conference on Software Quality, Reliability and Security (QRS'2017), Prague, Czech Republic. Since 2015, merger of SERE (International Conference on Software Security and Reliability) and QSIC (International Conference on Quality Software). Topics include: reliability, security, availability, and safety of software systems; software testing, verification and validation; program debugging and comprehension; fault tolerance for software reliability improvement; modeling, prediction, simulation, and evaluation; metrics, measurements, and analysis; software vulnerabilities; formal methods; benchmark, tools, and empirical studies; etc. Deadline for submissions: April 1, 2017 (workshop papers, Student Doctoral Program, fast abstract track).
August 16-18  23th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'2017), Hsinchu, Taiwan. Topics include: multi-core embedded systems; operating systems and scheduling; embedded software and compilers; fault tolerance and security; embedded systems and design methods for cyber-physical systems; real-time operating systems; real-time scheduling; timing analysis; programming languages and run-time systems; middleware systems; design and analysis tools; applications and case studies of IoT and CPS; cyber-physical co-design; etc. Deadline for submissions: April 14, 2017 (papers).

Aug 30 - Sep 01  43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA'2017), Vienna, Austria. Topics include: information technology for software-intensive systems; main tracks on Embedded Software Engineering (ESE), Model-based Development, Components and Services (MOCs), Software Process and Product Improvement (SPPI), Software Product Lines and Software Ecosystems (SPLSeco), etc.; special sessions on Teaching, Education and Training for Dependable Embedded and Cyber-Physical Systems (TET-DEC), Cyber-Physical Systems (CPS), Software Engineering and Technical Debt (SEaTeD), etc.


September 04-08  11th Joint European Meeting of the Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE'2017), Paderborn, Germany. Topics include: API usage and design; debugging, fault localization, and repair; dependability, safety, and reliability; development environments and tools; empirical studies; formal methods and verification; model-driven software engineering; parallel, distributed, and concurrent systems; performance and scalability; program analysis; refactoring, reengineering, and migration; security and privacy; software architecture; software economics; software evolution and maintenance; software processes and project organization; software testing; variability management and software product lines; etc. Deadline for submissions: May 12, 2017 (industry papers, Doctoral Symposium), June 1, 2017 (student volunteers), June 9, 2017 (Student Research Competition, tool demonstrations, artifact evaluation). Deadline for early registration: July 19, 2017.

September 06-10  15th International Conference on Software Engineering and Formal Methods (SEFM'2017), Trento, Italy. Deadline for submissions: June 1, 2017 (workshop papers).

September 12-15  International Conference on Parallel Computing 2017 (ParCo'2017), Bologna, Italy. Topics include: all aspects of parallel computing, including applications, hardware and software technologies as well as languages and development environments; new concepts for parallel computing architectures for all levels of parallelism (multicore and manycore systems, accelerators, including GPUs, FPGAs, ...); software engineering methodologies, methods and tools for developing and maintaining parallel software; parallel programming languages, compilers, libraries and environments; testing and debugging techniques and tools; best practices of parallel computing on multicore, manycore and stream processors; the application of parallel computing to solve all types of business, industrial, scientific and engineering problems using high-performance computing technologies; etc. Deadline for submissions: July 31, 2017 (full papers).

September 13-15  11th International Symposium on Theoretical Aspects of Software Engineering (TASE'2017), Nice, France. Topics include: theoretical aspects of software engineering, such as abstract interpretation, component-based systems, cyber-physical systems, distributed and concurrent systems, embedded and real-time systems, formal verification and program semantics, integration of formal methods, language design, model checking and theorem proving, object-oriented systems, run-time verification and monitoring, software architecture, software testing and quality assurance, software security and reliability, static analysis of programs, type systems and behavioural typing, tools exploiting theoretical results, etc.

September 13-16  17th International Conference on Runtime Verification (RV'2017), Seattle, Washington, USA. Topics include: monitoring and analysis of the runtime behaviour of software and hardware systems. Application areas include cyber-physical systems, safety/mission-critical systems, enterprise and systems software, autonomous and reactive control systems, health management and diagnosis systems,
and system security and privacy. Deadline for submissions: April 24, 2017 (paper and tutorial abstracts), May 1, 2017 (papers, tutorials).

September 20-22
13th International Conference on integrated Formal Methods (iFM’2017), Turin, Italy. Topics include: hybrid approaches to formal modeling and analysis; i.e., the combination of (formal and semi-formal) methods for system development, regarding both modeling and analysis, and covering all aspects from language design through verification and analysis techniques to tools and their integration into software engineering practice. Deadline for submissions: April 4, 2017 (papers).

October 15-20
ACM SIGBED International Conference on Embedded Software (EMSOFT’2017), Seoul, South Korea. Part of ESWeek, EMSOFT brings together researchers and developers from academia, industry, and government to advance the science, engineering, and technology of embedded software development. EMSOFT is a venue for cutting-edge research in the design and analysis of software that interacts with physical processes, with a long-standing tradition for results on cyber-physical systems, which compose computation, networking, and physical dynamics. Deadline for submissions: April 7, 2017 (full papers).

October 15-20
International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES’2017), Seoul, South Korea. Part of ESWeek, CASES is a forum where researchers, developers and practitioners exchange information on the latest advances in compiler and architectures for high-performance, low-power embedded systems. The conference has a long tradition of showcasing leading edge research in embedded processor, memory, interconnect, storage architectures and related compiler techniques targeting performance, power, predictability, security, reliability issues for both traditional and emerging application domains. In addition, we invite innovative papers that address design, synthesis, and optimization challenges in heterogeneous and accelerator-rich architectures. Deadline for submissions: April 7, 2017 (full papers), June 2, 2017 (Work-in-Progress papers).

October 22-27

October 23-27
14th International Colloquium on Theoretical Aspects of Computing (ICTAC’2017), Hanoi, Vietnam.

November 07-09
30th IEEE Conference on Software Engineering Education and Training (CSEET’2017), Savannah, USA.

December 05-08
38th IEEE Real-Time Systems Symposium (RTSS’2017), Paris, France. Topics include: all aspects of real-time systems theory, design, analysis, implementation, evaluation, and experiences. Deadline for submissions: May 1, 2017 (papers), June 9, 2017 (workshops), September 15, 2017 (workshop papers).

December 10 Birthday of Lady Ada Lovelace, born in 1815. Happy Programmers' Day!
Advance Information

The 22nd International Conference on Reliable Software Technologies (Ada-Europe 2017) will take place in Vienna, Austria. This conference is the latest in a series of annual international conferences started in the early 80’s, under the auspices of, and organization by, Ada-Europe, the international organization that promotes the knowledge and use of Ada and Reliable Software in general into academia, research and industry.

Ada-Europe 2017 provides a unique opportunity for dialogue and collaboration between academics and industrial practitioners interested in reliable software.

The conference will span a full week, including tutorials and a central three-day technical program with the latest advances in reliable software technologies and Ada. The core program features 3 keynote talks, 16 refereed scientific papers, and 9 industrial presentations. Co-located with the conference is the DeCPS workshop on “Focus on Transportation of the Future”. Half-day and full-day tutorials will be provided on Monday and Friday.

Week Overview

<table>
<thead>
<tr>
<th>Monday</th>
<th>Tuesday</th>
<th>Wednesday</th>
<th>Thursday</th>
<th>Friday</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regular session Runtimes</td>
<td>Regular session Safety &amp; Security</td>
<td>Regular session Timing Verification</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Industrial session Exploratory Uses of Ada</td>
<td>Industrial session Verification</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Regular session Programming Models</td>
<td>Panel The Future of Safety-Minded Languages</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ada-Europe General Assembly Welcome cocktail</td>
<td>Conference “Heuriger” Best paper award</td>
<td>Best presentation award Closing session</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Keynote talks

Each day of the core program will be opened a keynote talk delivered by one the following eminent speakers:

- **Giovanni Battista Gallus**, Array, Italy, “The laws of robotics and autonomous vehicles may be much more than three, but don't panic... yet.”
- **Thomas Henzinger**, IST, Austria, “Behavioral Software Metrics”
- **Kay Römer**, TU Graz, Austria, “Dependable Internet of Things”

Tutorials

Bracketing the conference on Monday and Tuesday, the program includes eight tutorials:

- Introduction to SPARK 2014, Peter Chapin, Monday full day.
- Ada on ARM Cortex-M, a zero-run-time approach, Maciej Sobczak, Monday morning.
- Real-Time Parallel Programming with the UpScale SDK, L. M. Pinho and E. Quinones, Monday afternoon.
- Frama-C, a Collaborative Framework for C Code Verification, Julien Signoles, Friday morning.
- Modular Open System Architecture for Critical Systems, William Bail, Friday afternoon

Co-Located Workshop

The conference week features the fourth International Workshop on Challenges and new Approaches for Dependable and Cyber-Physical Systems Engineering (De-CPS 2017), following the success of the inaugural workshop in 2014, its second edition in Madrid in 2015, and its third edition in Pisa in 2016. The workshop will take place Friday, June 16th, from 09:30 to 17:30.

About the Venue

The conference will take place at Vienna, Austria. The conference venue, the Palais Eschenbach was built in Palladian Style. It was inaugurated in 1872 by the Austrian Emperor Franz Joseph I. The so-called “golden ballroom” with its impressive coffered ceiling, arcade arches and a number of marble pilasters mirrors the great era of Vienna. The palais is located near the center of Vienna and can easily be accessed by metro lines U1, U2, and U4.

Sponsors
ARG Work in Progress

Jeff Cousins CEng FIET
Chair of the Ada Rapporteur Group
BAE Systems Surface Ships Limited KT3 4LH; Tel: +44 3300 466346; email: jeff.cousins@baesystems.com,
jeffrey.cousins@btinternet.com

Abstract

Where is Ada going next? After the exciting additions of Ada 2012, in particular contracts and aspects, Ada should be entering a quieter period. Small but interesting new features include array aggregate initialisation and a short-hand for the left-hand side of an assignment statement.

1 Introduction

2016 saw the publication of Technical Corrigendum 1 to the Ada 2012 standard, at the end of January. With that out of the way, the Ada Rapporteur Group (ARG) was able to start considering the Ada Issues (AIs) proposing various amendments to the language, not just those dealing with corrections to and clarifications of the existing language. We are unlikely to publish a new edition until we have some major (and exciting!) features ready, and these are still in the pipeline, but a number of smaller features have already been approved. This paper gives an overview of the more interesting of these changes, and a glimpse of what is still to come.

Readers will recall that Ada Issues are first worked on and approved by the ARG. They are then passed to WG 9 (the ISO/IEC Working Group responsible for Ada) for consideration and approval before eventually being consolidated and sent to ISO for formal processing to create a revised international standard.

2 WG 9 approved

This section describes some of the more important changes to the language that have been approved by WG 9.

2.1 Index parameters in array aggregates (AI12-0061)

Consider:

```ada
subtype Index is Positive range 1 .. 10;
type Array_Type is array (Index) of Positive;
Squares_Array : Array_Type := (for I in Index => I * I);
```

This provides a means of creating an aggregate when the element type is limited, provides a better means of initialising an array with a type invariant, and should be useful for everyday programming.

2.2 Object Size attribute (AI12-0059)

Users have been after this since 1983! S'Object_Size denotes the size of an object of subtype S. It can be specified, but must be specified to a value that the compiler is able to allocate (usually an entire storage unit for most implementations).

S'Object_Size is an improvement on S'Size (which cannot be redefined without breaking existing code). Reading 'Size is not terribly useful as it just gives the theoretical minimum number of bits required for a value of a given range, not the number of bits that the compiler is actually going to allocate to an object of the type. Specifying S'Size just gives a minimum, the compiler may allocate more.

2.3 Aggregates and variant parts (AI12-0086)

A discriminant that controls a variant can now be non-static if the subtype of the discriminant is static and all values belonging to that subtype select the same variant. For example:

```ada
type Enum is (Aa, Bb, Cc, ..., Zz);
subtype S is Enum range Dd .. Hh;
type Rec (D : Enum) is record
case D is
  when S => Foo, Bar : Integer;
  when others => null;
end case;
end record;
function Make (D : S) return Rec is
begin
  return (D => Dd, Foo => 123, Bar => 456);
end;
```

2.4 Use subtype_indication in generalized iterators (AI12-0156)

Ada 2012 added the ability to simplify

```ada
Vec : Int_Vectors.Vector;
...
for I in Vec.Iterate loop
  Vec(I) := Vec(I) + 1;
end loop;
```

where the optional "T" acts as a comment to the reader that the subtype of element E is T (and the compiler verifies this comment). An optional subtype indication --
though of the cursor not the element – can now also be
given for the original “in” form of the loop, i.e.:

```ada
for I : Index in Vec.Iterate loop
  Vec(I) := Vec(I) + 1;
end loop;
```

where Index is the subtype of the loop variable.

### 2.5 Preelaborable packages with address clauses (AI12-0175)

Packages with aspect Preelaborate can now contain
certain simple functions known to the compiler, i.e. an
instance of Unchecked_Conversion, a function declared in
System.Storage_Element, or the functions To.Pointer and
To.Address declared in an instance of System.Address_to_Access_Conversions. This allows the
declaring of objects with an address clause within a
preelaborable package, which can be very useful for small
embedded systems.

### 2.6 Access to parts of composite atomic objects (AI12-0128)

Memory accesses to subcomponents of an atomic
composite object must read or write the entire object. For example:

```ada
type Status is record
  Ready : Boolean;
  Length : Integer range 0 .. 15;
end record;
for Status use record
  Ready at 0 range 0 .. 0;
  Length at 0 range 1 .. 5;
end record;
Status_Register : Status
  with Address => ..., Size => 32, Atomic => True;
if Status_Register.Ready then -- Reads entire register
  null;
end if;
Status_Register.Length := 10; -- Prereads entire
  register, then writes
  -- entire register.
```

This is useful for controlling accesses to memory mapped
device registers, which often require reads or writes to be
to the entire register.

### 2.7 Aggregates of Unchecked_Unions using
named notation (AI12-0174)

Given that it is generally regarded as good practice to use
named notation rather than positional notation, it was
somewhat bizarre that Unchecked_Unions only allowed the latter. Both are now allowed. For example:

```ada
type Data_Kind is (C_int, C_char);
type C_Variant (Format : Data_Kind := C_int) is record
  case Format is
    when C_int =>
      int_Val : C.int;
    when C_char =>
      char_Val : C.char;
end case;
end record with Unchecked_Union, Convention => C;
Int1 : C_Variant := (C_int, 12); -- Always OK
Int2 : C_Variant := (Format => C_int, int_Val => 12);
-- Was illegal, now OK
```

### 3 In the pipeline

These have been approved by the ARG but have yet to be
approved by WG 9.

#### 3.1 Add @ as an abbreviation for the LHS of an
assignment (AI12-0125-3)

This proposal, which has proven rather controversial with
those who are used to Ada being verbose, uses a single
character placeholder for the left hand side of an
assignment.

```ada
My_Package.My_Array(I).Field :=
  My_Package.My_Array(I).Field + 1;
```

could be shortened to:

```ada
My_Package.My_Array(I).Field := @ + 1;
```

The above is similar in function to the += of the C family
of languages. The proposal for Ada is more powerful
though, being able to handle expressions such as series
expansions. Here are a couple of examples:

```ada
My_Package.My_Array(I).Field :=
  My_Package.My_Array(I).Field ** 3 +
  My_Package.My_Array(I).Field ** 2 +
  My_Package.My_Array(I).Field;
```

could be shortened to:

```ada
My_Package.My_Array(I).Field := @ ** 3 + @ ** 2 + @;
```

and:

```ada
My_Package.My_Array(I).Field :=
  Natural'Min (My_Package.My_Array(I).Field, 1000);
```

could be shortened to:

```ada
My_Package.My_Array(I).Field := Natural'Min
  (@, 1000);
```

#### 3.2 Update to the Fortran Annex (AI12-0058)

The Fortran Annex will be updated to support Fortran
2008, in particular better support for double precision
complex arithmetic. Permissions corresponding to non-
standard extensions, or implementation advice that is now
considered to be bad practice, will be removed.

### 4 The Future

#### 4.1 Support for Static Analysis

Global aspects (AI12-0079) extend the contract features
provided by Ada 2012, and are used to specify which
global objects a subprogram may access, and in which
mode. For example (for a board game):

```ada
procedure Include_Piece_In_Board
  with Global => (Input => Cur_Piece,
      In_Out => Cur_Board);
```
For backward compatibility reasons, if the global aspect is not specified, then the subprogram is presumed to read and write an unspecified set of global variables (or none if the subprogram is in a pure package).

The new pragma Loop_Invariant allows some property to be checked each time around a loop. For example:

```ada
for ... loop
    Some_Complex_Calculation;
    pragma Loop_Invariant (Calculation_Is_Converging);
end loop;
```

where Calculation_Is_Converging is a Boolean expression.

The above are already provided by GNAT to support SPARK 2014. This does not mean that it will be trivial to add them to the main language though, as these features have to be generalised to cover the whole language, not just the SPARK subset.

The following features are likely to prove useful to analysis tools such as SPARK.

The new aspect Nonblocking (AI12-0064) turns the bounded error of 9.5.1 (or the runtime check of pragma Detect_Blocking) into a compile-time check. When used, it means that one cannot call a potentially blocking operation by mistake and cause a problem down the road. Although we cannot make it the default for compatibility reasons, one hopes that most new code will make using it the norm.

The new aspect Stable_Properties (AI12-0187) simplifies the description of properties of an abstract data type (ADT), by making it easy to specify properties that are usually unchanged by most of the operations of the ADT. The classic example is the Mode of a file, which is unchanged by all of the operations other than Create/Open/Close/Reset (and Set_Mode for streams). The stable properties are automatically included in the postconditions of all of the primitive operations of the ADT, decreasing clutter and increasing the information that provers can use.

### 4.2 Parallelism (AI12-0119)

As Ada provided tasking from the outset, migrating to multi-core processors with a small number of cores was relatively easy. But as the number of cores increases, finer grained control of parallelism will be required. OpenMP already provides facilities for C, C++ and Fortran.

A sub-group, led by Canada, is considering various ideas for supporting parallel blocks and loops. The compiler would determine how much parallelism (how many “tasklets”, in the terminology of one of the proposals) is used to implement these constructs (possibly based on the number of cores on the target machine).

### 4.3 Others

Investigations continue into providing more variants of the Containers, e.g. with faster access or including mutual exclusion, and improving iteration over Containers and other structures, though if every proposal were to be accepted there would be a combinatorial explosion in the number of permutations of the Containers.

AdaCore presentations have suggested the following extensions:

- Generators/co-routines (AI12-0197);
- Lambda functions (AI12-0190);
- Function decorators;
- Declare variable in expression constructs.

So far only the first two have been raised as AIs, and even then only recently.

### 5 Conclusions

The balancing act continues, between keeping the language stable and backwardly compatible and adding new features to move with the times. When the parallel processing proposals mature or SPARK is ready to be standardised, no doubt Ada will rise to the occasion.
Tools to get you there. Safely.

Ada and The GNAT Pro High-Integrity Family

www.adacore.com
Definition-Use Net and System Dependence Net Generators for Ada 2012 Programs and their Applications

Bo Wang, Hongbiao Gao, Jingde Cheng
Department of Information and Computer Sciences, Saitama University, 255 Shimo-Okubo, Sakura-ku, Saitama, 338-8570, Japan; Tel: +81 48 858 3785; email: {wangbo,gaohongbiao,chengf}@aise.tcs.saitama-u.ac.jp

Abstract

Both Definition-Use Net (DUN) and System Dependence Net (SDN) are formal graph-theoretical representation models for concurrent programs. Both models are very helpful in software development activities such as program slicing, testing, debugging, and complexity measuring. Ada 2012, which is the new generation of the world's premier programming language for engineering safe, secure and reliable software, has many changes and extensions from Ada 2005. Until now, however, there has been no investigation on the DUN of Ada 2012 programs and its automated generation method, and although some new program dependences and one new interprocedural relation have been found in Ada 2012 programs, there is no report about definitions of the new program dependences and interprocedural relations of an SDN, and even their automated generation methods. For capturing Definition-Use Nets and System Dependence Nets of Ada 2012 programs automatically, we developed an ASIS-based DUN generator, and an SDN generator. This paper introduces DUNs of Ada 2012 programs, presents definitions of new types of program dependences and SDNs in Ada 2012 programs, shows example SDNs including new types of program dependences in Ada 2012 programs, presents the methods of constructing DUNs and SDNs of Ada 2012 programs, and shows various applications of DUNs and SDNs.

Keywords: Ada 2012, concurrent programs, definition-use nets, program dependences, system dependence nets, interprocedural relations.

1 Introduction

Concurrent systems are more and more important in the contemporary society. We use concurrent systems to manipulate large-scale devices applied in various areas, such as aerospace, aircraft, high-speed railway, avionics system, military, traffic and air control, vehicle control system, securities exchange system, banking system, and so on. These concurrent systems are not allowed any malfunction. Otherwise personal safety and assets might suffer catastrophic consequences. Thus, software engineering must make sure these software systems safe and secure, so that software programs do not even have a fault.

It is quite difficult to design, understand, test, debug, and maintain concurrent programs. The reason is that multiple control flows and data flows can exist simultaneously in concurrent systems, and their behaviors are non-deterministic. Therefore, their development needs effective supporting tools.

A statement in a program cannot exist independently, but is dependent on others. Program dependences [1, 2] are dependences holding between statements in a program and they are determined by control flows and data flows in the program. A subprogram in a program also depends on other subprograms. Moreover, a process in a concurrent program is also dependent on other processes.

Program dependences analysis has been proven the cornerstone for various of software engineering activities [3]. Program Dependence Graph (PDG) [1, 2] is very useful for software engineering applications, such as testing [4], debugging [5, 6], maintenance [7], identifying similar code [8], refactoring [9], probabilistic noninterference analysis [10], and exploring and enforcing security guarantees [11].

Cheng has proposed Nondeterministic Parallel Control Flow Net (CFN), and Nondeterministic Parallel Definition-Use Net (DUN) [12]; which are both arc-classified digraphs, in order to represent concurrent and/or distributed programs. The CFN represents multiple control flows in a concurrent program as well as the single control flow in a sequential program. The DUN is an extension of CFN, such that it represents multiple control flows, definitions and uses of variables, inter-process synchronization and communication in a concurrent program. Moreover, based on the CFN and DUN of a concurrent program, Cheng has further proposed Process Dependence Net (PDN) [13, 14], which is an arc-classified digraph to represent various program dependences in concurrent programs explicitly. Task Dependence Net (TDN) [13, 15, 16] is a version for the PDN of Ada programs. System Dependence Net [17] is a formal model which can present the program dependences and interprocedural relations in a concurrent program with multiple procedures, such that it is extended from PDN.

There are many useful applications of DUNs and SDNs. DUNs can provide a clear and precise basis for definitions of notions, descriptions of methods, and developments of
tools in software engineering. We can use them as program representation and understanding tools in the software design and specifications. DUNs also provide a basis for defining the software test coverage criteria and generating the software test data [12]. Meanwhile, SDNs are important to software engineering activities such as program slicing, testing, debugging, and complexity measuring [18, 19].

Furthermore, the programming language Ada, the new generation of the world’s premier programming language for engineering safe, secure, and reliable software, has evolved to Ada 2012, i.e., the latest version of the Ada language standard [20]. It is designed for large and long-lived systems, and widely used in the worlds of high-integrity, high-security, which are closely related to commercial, military airborne avionics, air traffic control, railroad systems, and medical equipment [21, 22, 23]. Ada 2012 has many changes and extensions from the previous version Ada 2005, including a seismic shift for supporting contract-based programming explicitly, which effectively improves the reliability of a program [20, 24, 25].

Until now, for previous versions of Ada 2012, program dependences have been proposed [12, 15, 16, 17, 26], such as control dependence, data dependence, selection dependence, synchronization dependence, communication dependence. Meanwhile some interprocedural relations have already been proposed, such as call-relations, parameter-in-relation, returned-value-relation, parameter-out relation. Moreover, some new types of program dependences have been found in Ada 2012 programs, such as pre-condition dependence, post-condition dependence, predicate dependence, expression dependence, and task-barriers dependence [27].

Although Ada 2012 has many changes and extensions from Ada 2005, there is no report about DUNs of Ada 2012 programs. And despite the fact that we have found these new types of program dependences in Ada 2012 programs mentioned above, there is no report of representing definitions of new types of program dependences and approach of generating them. Furthermore, the SDNs of Ada 2012 programs have not been defined. In this paper, we introduce DUNs of Ada 2012 programs, present definitions of new types of program dependences in Ada 2012 programs, SDNs of Ada 2012 programs, show example SDNs including new types of program dependences in Ada 2012 programs, present the method of constructing DUNs and SDNs of Ada 2012 programs, elaborate our approach to generate DUNs and SDNs of Ada 2012 programs automatically, show various applications of DUNs and SDNs.

The rest of this paper is organized as follows: Section 2 constructs definition-use nets for Ada 2012 programs; Section 3 defines new types of program dependences in an Ada 2012 program based on its DUNs; Section 4 introduces and illustrates the method of constructing system dependence nets of Ada 2012 programs; Section 5 presents the approach to generate DUNs of Ada 2012 programs automatically; Section 6 presents the approach to generate SDNs of Ada 2012 programs automatically; Section 7 shows some applications of DUNs and SDNs; Some concluding remarks are given in Section 8.

2 Definition-Use Nets of Ada 2012 Programs

2.1 An Overview

With the purpose of constructing SDNs for Ada 2012 programs, we have to recognize, judge, define, and analyze various program dependences and interprocedural relations, which are determined by multiple control flow and data flow implicitly.

Definition-Use Net is a graph-theoretical concurrent program representation. It is an arc-classified digraph, which has vertices representing program statements with information about definitions and/or uses of the values of variables, formal parameters of the subprograms, returned values from functions, actual parameters of subprogram call, and inter-process synchronization and communication, and arcs representing deterministic or non-deterministic control flow between vertices. By exploiting it, we can acquire an explicit representation of control flow and data flow.

Therefore, it is indispensable to model DUNs of Ada 2012 programs at first, to emerge from multiple control flow and data flow in an Ada 2012 program, we must convert mathematical symbols [12] to actual graphics for Ada 2012 programs.

Here, we illustrate how to represent Ada 2012 programs by Definition-Use Nets as definitions [12], such that convert mathematical symbols to actual graphics. DUNs of Ada 2012 programs consist of vertices, types of vertices, labels of vertices, and arcs. We map all kinds of statements concerned with DUNs to vertices in Table 1, and then illustrate every representation of the vertex corresponding to the term Num of examples in Table 2.

2.2 Vertices of DUNs of Ada 2012 Programs

A vertex of definition-use nets of an Ada 2012 program may represent either a declaration, or a definition, or a control predicate corresponding to an expression, or a statement [20, 24, 25, 28], respectively. Every representation has some subordinates as the following Table 1. Every vertex takes one type and one label on account of Table 1.

2.3 Types of Vertices of DUNs of Ada 2012 Programs

As above representations, there are five types of special vertices as nondeterministic selection vertex $N_s$, parallel execution fork vertex $P_F$, parallel execution join vertex $P_J$, start vertex Start, termination vertex Termination, and an ordinary type of others, as the term Type in Table 1.

In Ada 2012, the nondeterministic selection vertex just appears as a select accept statement, or a conditional entry call statement, or a timed entry call statement, or a queue statement with abort as the Num A-19, M-1, N-1, P-1 in Table 2, respectively.

The parallel execution fork vertex represents a parallel execution branch starting vertex. For Ada 2012 programs, subprogram and task units may be arranged in hierarchies of parent and child units giving fine control over visibility of the
Table 1: Vertices of DUNs of Ada 2012 Programs

<table>
<thead>
<tr>
<th>Representation</th>
<th>Subordinate</th>
<th>Type</th>
<th>Label</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Declaration</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>variable declaration</td>
<td>Ordinary</td>
<td>D(v)</td>
<td>A-2</td>
<td></td>
</tr>
<tr>
<td>procedure body declaration</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ordinary</td>
<td>P_f</td>
<td>n/a</td>
<td>A-29</td>
</tr>
<tr>
<td></td>
<td>Ordinary</td>
<td>P_j</td>
<td>n/a</td>
<td>A-31</td>
</tr>
<tr>
<td></td>
<td>S</td>
<td>n/a</td>
<td>A-1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>T</td>
<td>n/a</td>
<td>A-31</td>
</tr>
<tr>
<td>task body declaration</td>
<td>Ordinary</td>
<td>S(v)</td>
<td>A-9,14,18</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ordinary</td>
<td>R(v)</td>
<td>A-30</td>
<td></td>
</tr>
<tr>
<td>aspect specification</td>
<td>Ordinary</td>
<td>U(v)</td>
<td>B-2</td>
<td></td>
</tr>
<tr>
<td>Definition</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>function call</td>
<td>Ordinary</td>
<td>U(v)</td>
<td>C-1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ordinary</td>
<td>U(v)</td>
<td>D-1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ordinary</td>
<td>U(v)</td>
<td>E-1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ordinary</td>
<td>U(v)</td>
<td>F-1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ordinary</td>
<td>U(v)</td>
<td>G-1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ordinary</td>
<td>U(v)</td>
<td>G-4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ordinary</td>
<td>U(v)</td>
<td>H-1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ordinary</td>
<td>U(v)</td>
<td>I-1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ordinary</td>
<td>U(v)</td>
<td>A-21</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ordinary</td>
<td>U(v)</td>
<td>A-21</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ordinary</td>
<td>U(v)</td>
<td>J-1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ordinary</td>
<td>D(v)</td>
<td>K-1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ordinary</td>
<td>U(v)</td>
<td>L-1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ordinary</td>
<td>D(v)</td>
<td>A-15</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ordinary</td>
<td>U(v)</td>
<td>A-10</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ordinary</td>
<td>S(v)</td>
<td>A-10,15</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ordinary</td>
<td>R(v)</td>
<td>A-11,15</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>N_s</td>
<td>n/a</td>
<td>M-1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>N_s</td>
<td>n/a</td>
<td>N-1</td>
</tr>
<tr>
<td></td>
<td>Ordinal</td>
<td>D(v)</td>
<td>A-15</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ordinal</td>
<td>U(v)</td>
<td>A-10</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ordinal</td>
<td>S(v)</td>
<td>A-10,15</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ordinal</td>
<td>R(v)</td>
<td>A-11,15</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ordinal</td>
<td>D(v)</td>
<td>O-1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Statement</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ordinal</td>
<td>D(v)</td>
<td>A-19</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ordinal</td>
<td>U(v)</td>
<td>A-20</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ordinal</td>
<td>U(v)</td>
<td>A-26</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ordinal</td>
<td>R(v)</td>
<td>A-20,24</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ordinal</td>
<td>S(v)</td>
<td>A-22,26</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>N_s</td>
<td>n/a</td>
<td>P-1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ordinal</td>
<td>D(v)</td>
<td>Q-1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ordinal</td>
<td>U(v)</td>
<td>R-1</td>
<td></td>
</tr>
</tbody>
</table>
Table 2: Examples of DUNs of Ada 2012 Programs

<table>
<thead>
<tr>
<th>Num</th>
<th>Example</th>
<th>Type/Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>A-1</td>
<td>procedure Parent is</td>
<td></td>
</tr>
<tr>
<td>A-2</td>
<td>X,Y:Integer :=1;</td>
<td></td>
</tr>
<tr>
<td>A-3</td>
<td>task T0 is</td>
<td></td>
</tr>
<tr>
<td>A-4</td>
<td>entry E1(I : in Integer);</td>
<td></td>
</tr>
<tr>
<td>A-5</td>
<td>entry E2(O : out Integer);</td>
<td></td>
</tr>
<tr>
<td>A-6</td>
<td>end T0;</td>
<td></td>
</tr>
<tr>
<td>A-7</td>
<td>task T1; task T2;</td>
<td></td>
</tr>
<tr>
<td>A-8</td>
<td>task body T1 is</td>
<td></td>
</tr>
<tr>
<td>A-9</td>
<td>begin</td>
<td></td>
</tr>
<tr>
<td>A-10</td>
<td>T0.E1(Y);</td>
<td></td>
</tr>
<tr>
<td>A-11</td>
<td>Put_Line(“This is Child T1.”);</td>
<td></td>
</tr>
<tr>
<td>A-12</td>
<td>end T1;</td>
<td></td>
</tr>
<tr>
<td>A-13</td>
<td>task body T2 is</td>
<td></td>
</tr>
<tr>
<td>A-14</td>
<td>begin</td>
<td></td>
</tr>
<tr>
<td>A-15</td>
<td>T0.E2(X);</td>
<td></td>
</tr>
<tr>
<td>A-16</td>
<td>end T2;</td>
<td></td>
</tr>
<tr>
<td>A-17</td>
<td>task body T0 is</td>
<td></td>
</tr>
<tr>
<td>A-18</td>
<td>begin</td>
<td></td>
</tr>
<tr>
<td>A-19</td>
<td>select</td>
<td></td>
</tr>
<tr>
<td>A-20</td>
<td>accept E1(I : in Integer) do</td>
<td></td>
</tr>
<tr>
<td>A-21</td>
<td>Y:=I*I;</td>
<td></td>
</tr>
<tr>
<td>A-22</td>
<td>end E1;</td>
<td></td>
</tr>
<tr>
<td>A-23</td>
<td>or</td>
<td></td>
</tr>
<tr>
<td>A-24</td>
<td>accept E2(O : out Integer) do</td>
<td></td>
</tr>
<tr>
<td>A-25</td>
<td>O:=X;</td>
<td></td>
</tr>
<tr>
<td>A-26</td>
<td>end E2;</td>
<td></td>
</tr>
<tr>
<td>A-27</td>
<td>end select;</td>
<td></td>
</tr>
<tr>
<td>A-28</td>
<td>end T0;</td>
<td></td>
</tr>
<tr>
<td>A-29</td>
<td>begin</td>
<td></td>
</tr>
<tr>
<td>A-30</td>
<td>Put_Line(“This is Parent.”);</td>
<td></td>
</tr>
<tr>
<td>A-31</td>
<td>end Parent;</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Example</th>
<th>Type/Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>D(2)={X,Y}</td>
</tr>
<tr>
<td>S(9)={Parent}</td>
<td></td>
</tr>
<tr>
<td>U(10)={Y}, S(10)={T0.E1}</td>
<td></td>
</tr>
<tr>
<td>R(11)={T0.E1.END!}</td>
<td></td>
</tr>
<tr>
<td>S(14)={Parent}</td>
<td></td>
</tr>
<tr>
<td>D(15)={X}, S(15)={T0.E1}</td>
<td></td>
</tr>
<tr>
<td>R(16)={T0.E2.END!}</td>
<td></td>
</tr>
<tr>
<td>S(18)={Parent}</td>
<td></td>
</tr>
<tr>
<td>Nv</td>
<td></td>
</tr>
<tr>
<td>D(20)={I}, R(20)={T0.E1}</td>
<td></td>
</tr>
<tr>
<td>D(21)={Y}, U(21)={I}</td>
<td></td>
</tr>
<tr>
<td>S(22)={T0.E1.END!}</td>
<td></td>
</tr>
<tr>
<td>R(24)={T0.E2}</td>
<td></td>
</tr>
<tr>
<td>U(26)={O}</td>
<td></td>
</tr>
<tr>
<td>S(26)={T0.E2.END!}</td>
<td></td>
</tr>
<tr>
<td>Pf</td>
<td></td>
</tr>
<tr>
<td>R(30)={Parent}</td>
<td></td>
</tr>
<tr>
<td>Pf, Termination</td>
<td></td>
</tr>
</tbody>
</table>
Table 3: Examples of DUNs of Ada 2012 Programs (Continued)

<table>
<thead>
<tr>
<th>Num</th>
<th>Example</th>
<th>Type/Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>B-1</td>
<td>function Sqrt (I : Integer) return Integer with Precondition =&gt; I = 0;</td>
<td>U(2)=(I)</td>
</tr>
<tr>
<td>B-2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C-1</td>
<td>Function_Name(X);</td>
<td>U(1)={X}</td>
</tr>
<tr>
<td>D-1</td>
<td>if N = 1 and Y &lt; 1 then</td>
<td>U(1)={N,Y}</td>
</tr>
<tr>
<td>E-1</td>
<td>if X in 0.5 .. Z</td>
<td>U(1)={X,Z}</td>
</tr>
<tr>
<td>F-1</td>
<td>if M not in Mon .. Fri then</td>
<td>U(1)={M}</td>
</tr>
<tr>
<td>G-1</td>
<td>Trans_fares :=(case Transport is</td>
<td>U(1)={Transport}</td>
</tr>
<tr>
<td>G-2</td>
<td>when Train</td>
<td>U(1)={Mile}</td>
</tr>
<tr>
<td>G-3</td>
<td>when Bus =&gt; (if Mile &lt;= 100 then</td>
<td>U(1)={Mile}</td>
</tr>
<tr>
<td>G-4</td>
<td>(if Mile &gt; 70 then 220 else 180);</td>
<td>U(1)={Mile}</td>
</tr>
<tr>
<td>H-1</td>
<td>B :=(for all I in M'Range =&gt; M(I)='C');</td>
<td>U(1)={I}</td>
</tr>
<tr>
<td>I-1</td>
<td>B :=(for some I in K'Range =&gt; K(I)=10);</td>
<td>U(1)={I}</td>
</tr>
<tr>
<td>J-1</td>
<td>while X &lt; 10 loop</td>
<td>U(1)={X}</td>
</tr>
<tr>
<td>K-1</td>
<td>for I in Integer range 100 .. 200 loop</td>
<td>D(1)={I}</td>
</tr>
<tr>
<td>L-1</td>
<td>Procedure_Name(X);</td>
<td>U(1)={X}</td>
</tr>
<tr>
<td>M-1</td>
<td>select</td>
<td></td>
</tr>
<tr>
<td>M-2</td>
<td>C.Rendezvous</td>
<td></td>
</tr>
<tr>
<td>M-3</td>
<td>else</td>
<td></td>
</tr>
<tr>
<td>M-4</td>
<td>Put_Line(&quot;Else&quot;);</td>
<td></td>
</tr>
<tr>
<td>M-5</td>
<td>end select;</td>
<td></td>
</tr>
<tr>
<td>N-1</td>
<td>select</td>
<td></td>
</tr>
<tr>
<td>N-2</td>
<td>C.Rendezvous</td>
<td></td>
</tr>
<tr>
<td>N-3</td>
<td>or</td>
<td></td>
</tr>
<tr>
<td>N-4</td>
<td>delay 5.0;</td>
<td></td>
</tr>
<tr>
<td>N-5</td>
<td>end select;</td>
<td></td>
</tr>
<tr>
<td>O-1</td>
<td>return R: Integer do</td>
<td>D(1)={R}</td>
</tr>
<tr>
<td>P-1</td>
<td>requeue Entry1 with abort;</td>
<td>N_a</td>
</tr>
<tr>
<td>Q-1</td>
<td>requeue E(O: out Integer);</td>
<td>D(1)={O}</td>
</tr>
<tr>
<td>R-1</td>
<td>requeue E(I: in Integer);</td>
<td>U(1)={I}</td>
</tr>
</tbody>
</table>
logical properties and their detailed implementation. If there are some tasks declared in a subprogram, the subprogram is called these tasks’ parent. They are parent-child relation. After elaborating the declarative_part of the parent subprogram, the tasks start to be activated. The initial part of the execution of the task body is referred to as the activation of the task, which consists of the elaboration of the declarative_part of the task body [20, 29, 30, 31]. Therefore, a parallel execution branch starting vertex always represents the statement immediately following the declaration part of the parent subprogram, such that it is the reserved word begin of the parent subprogram body. For example, the parallel execution branch starting vertex appears on line A-29 in the Parent example program of Table 2.

The parallel execution joins vertex represents a parallel execution branch confluence vertex. In Ada 2012 programs, after all statements of the task body are completed, the parent subprogram frees up the space of local variables and then exits. Even all the statements of the parent subprogram body are earlier completed, it must wait for the completion of all the statements of the task body. In such a case, the task body is dependent on the parent subprogram [20, 29, 30, 31]. Hence, a parallel execution branch join vertex always represents the statement immediately following the last statement of the parent subprogram body, such that it is the statement with the reserved word end of the parent subprogram body, like the line A-31 of the above example program Parent.

The start vertex of DUNs for Ada 2012 programs represents the program unique start vertex, that is, it is always the start vertex of the outermost parent subprogram. In Ada 2012 programs, it appears as the first statement of a procedure body, or a package body, for example, the line A-1 of the example program Parent.

Similarly, the termination vertex of DUNs for Ada 2012 programs represents the program unique termination vertex, that is, it is always the termination vertex of the outermost parent subprogram. In Ada 2012 programs, it appears as the last statement of a procedure body declaration, or a package body declaration, i.e., it has the reserved word end. Note that sometimes the termination vertex and the parallel execution join vertex is the same vertex, but still unique vertex, for instance, the line A-31 of the example program Parent is also a termination vertex, moreover it is the unique termination vertex of the program.

2.4 Types of Labels of Vertices of DUNs of Ada 2012 Programs

The vertices of the DUNs are labeled with information about definitions and/or uses of the values of variables and inter-process synchronizations and communications as above definitions [15, 20, 26, 29, 30, 31].

$D(v)$ is the set of all variables defined at $v$. In Ada 2012 programs, the set of variables defined representations as follows:

- a variable that appears on the left of an assignment statement, such as A-21 of Table 2
- a variable that is declared by a declaration statement excluding a constant declaration statement, such as A-2
- a variable that is actually referred to a formal parameter with in mode, or in out mode in an accept statement, such as A-20
- a variable that is actually referred to an actual parameter with out mode, or in out mode in an entry call statement without protected, such as A-15
- a variable that is actually referred to a loop parameter in a for loop statement, such as K-1
- a variable that is actually referred to a formal parameter with out mode, or in out mode in a requeue statement, such as Q-1
- a variable that appears in an extended return statement, such as O-1

$U(v)$ is the set of all variables used at $v$. In Ada 2012 programs, the set of variables used representations as follows:

- a variable that appears on the right of an assignment statement, such as A-21 in Table 2
- a variable that appears in a function call expression, such as C-1
- a variable that appears in a procedure call expression without containing any parameters, such as L-1
- a variable appears in the control predicate of conditional branches, such as D-1
- a variable that is actually referred to a formal parameter with out mode, or in out mode, appears on the last statement to represent the end of an accept statement, such as A-26
- a variable that is actually referred to an actual parameter with in mode, or in out mode in an entry call statement without protected, such as A-10
- a variable that is actually referred to a formal parameter with in mode, or in out mode in a requeue statement, such as R-1
- a variable appears on the right of a Boolean expression, such that it appears on the right of the symbol = of an aspect specification, such as Precondition, Postcondition, Type_Invariant, Static_Predicate, Dynamic_Predicate, even Dispatching_Domain, such as B-2
- a variable appears in some expressions, such as a membership test expression, a case expression, an if expression, a for all quantified expression, a for some quantified expression, such as E-1, F-1, G-1, G-4, H-1, and I-1 in Table 2

$S(v)$ is the set of symbols of the sending message functions at $v$.

$R(v)$ is the set of symbols of the receiving message functions at $v$.

In Ada 2012 programs, after the activation of the task is completed, the statements of the parent subprogram begin to
execute. Therefore, the statement immediately following the declarative part of the task body, i.e., a vertex representing the reserved word \textit{begin} of the task body, is labeled with the sending message function \(S(v)\), such as A-9, A-14, A-18 in Table 2, while the first statement of the parent subprogram, i.e., the first statement under the \textit{begin} of the parent subprogram, such as A-30, is labeled with receiving message function \(R(v)\), which is meaning that the child tasks complete the activations. The name of the symbol is the same as the parent subprogram name.

Also, if there is an entry call to another task, the entry call statement is labeled with \(S(v)\) representing the sending message function, such as A-10, A-15 in Table 2, the accept statement corresponding to the entry call statement is labeled with \(R(v)\) representing receiving message function, such as A-20 and A-24. Whereas, the vertex that represents \textit{end} of the accept statement is labeled with \(S(v)\) as the sending message function, such as A-22 and A-26, to the next statement of the entry call statement, which is labeled with receiving message function \(R(v)\), such as A-11 and A-16. The symbol’s name is the name of the entry call, representing that the rendezvous of this entry call is ending. The next statement can execute.

Furthermore, if an Ada 2012 program has some subprogram calls, i.e., function calls or procedure calls [20], we should extend the DUN with labels, such as:

- \(A_{in}\) represents the set of all parameters with the mode \textit{in} \(v\) that is the subprogram call statement.
- \(A_{out}\) represents the set of all parameters with the mode \textit{out} or \textit{in} \(v\) that is the subprogram call statement.
- \(F_{in}\) represents the set of all parameters with the mode \textit{in} \(v\) that is the start vertex of the subprogram, i.e., callee function or procedure.
- \(F_{out}\) represents the set of all parameters with the mode \textit{out} or \textit{in} \(v\) that is the start vertex of the subprogram, i.e., a callee function or procedure.

### 2.5 Arcs of DUNs of Ada 2012 Programs

In DUNs for Ada 2012 programs, the arcs represent several types of possible transfers of control between vertices. The two vertices have one or more types of transfers of control.

### 2.6 Types of Arcs of DUNs of Ada 2012 Programs

As definitions given [12], there are three types of arcs, sequential control arcs, nondeterministic selection arcs, and parallel execution arcs.

The sequential control arc \((A_C)\) represents that the control is transferred sequentially between a sequence of statements in a subprogram, or a task, or a procedure.

The nondeterministic selection arc \((A_{NS})\), always directs from the nondeterministic selection vertex to every select alternatives. There are at least two select alternatives corresponding to the selective statement, such that a nondeterministic selection vertex has at least two nondeterministic selection arcs adjacent to the select alternatives.

The parallel execution arc represents a control thread diverges from a parallel execution fork vertex to the vertices representing other blocks start, i.e., \(A_{P_e}\); Or represents several control threads are confluence to a parallel execution join vertex, i.e., \(A_{P_j}\).

Especially, synchronization channels represent a channel of \(S(v)\) sending message function to \(R(v)\) receiving message function. Though in essence they do not belong to control transfers, we still use arcs to depict them to express synchronization between vertices.

Also, for the scenario that there are some subprogram calls in an Ada 2012 program, the DUN should represent the interprocedural relationships between call sites and callee sites. The call arc \(A_{CA}\) is connected from a function call statement/procedure call statement to the start statement of function body corresponding to its call/the start statement of procedure corresponding to its call.

Here, we will illustrate how to represent Ada 2012 programs by Definition-Use Nets as above definitions. There has been given an example of an Ada 2012 program in the following program Example, and its DUN is depicted in Figure 1. Program Example is a representative Ada 2012 program. The vertex number represents the line number of the actual program corresponding to the statement. The number following a variable expresses the number of the variable in a compilation unit (The text of a program can be submitted to the compiler in one or more compilations. Each compilation is a succession of compilation units. A compilation unit contains either the declaration, the body, or a renaming of a program unit [25]). The same variable uses the same number. In this Example, procedure Example is the parent of task T1 and T2, and it has two subprograms, i.e., function Add and function expression ExpA. Line 67 is fork vertex to the vertices line 45 and line 61 corresponding to the beginning of the activation of task body T1 and T2 respectively. Both are labeling parallel execution arc. After the activation of task body T1 and T2, the vertex line 46 and 63 are sending a message to the vertex line 68 of the execution of the beginning of the parent, labeling synchronization channel, respectively. The vertices line 59 and line 65 are labeled with Join, expressing that control threads are confluence to the termination of the parent. For task T1, task T2, procedure Example, and subprogram function Add, control flow is just sequential transfers, such that they are labeled with sequential control arc in their own internal.

In addition, there are two entry calls in the procedure Example, such as Start and Quit. The entry call statements 74 and 76, both are labeled with \(S(v)\) representing to send messages, and the two accept statements corresponding to the two entry calls statement respectively, are labeled with \(R(v)\) representing to receive messages, such as line 50 and line 53 in the task body T1. Whereas, the vertex that represents the finish of the accept statement is labeled with \(S(v)\) representing the sending messages, such as line 51 and line 54, to the next statement of the entry call statement, which is labeled with the receiving message function \(R(v)\), such as line 75 and line 77. The symbol’s name is the name of the entry call,
representing that the rendezvous of this entry call is ending. The next statement can execute.

Listing 1: Example of an Ada 2012 Program

1 with Ada.Text_IO;
2 use Ada.Text_IO;
3 with Ada.Integer_Text_IO;
4 use Ada.Integer_Text_IO;
5 with Ada.Synchronous_Bariers;
6 use Ada.Synchronous_Bariers;
7
8 Procedure Example is
9
10 NT : constant := 2;
11 SB : Synchronous_Barrier (NT);
12 Notified : Boolean := False;
13
14 subtype Single is Integer range 1..50;
15 subtype Double is Integer
16 with Dynamic_Predicate =>
17 Double mod 3 = 0
18 and then
19 Double / 2 in Single;
20
21 function Add(Arg : Integer) return Integer is
22 with Pre => Arg >= 10,
23 Post => Add_Result <= 100;
24
25 function Add(Arg : in Double) return Integer is
26 begin
27 return Arg + 10;
28 end Add;
29
30 function Exp(A: Integer) return Integer is
31 (if 1 = 0 then 1 else Exp(A-1) + 2);
32
33 X, Y : Integer;
34 Z : Integer := 40;
35 Bool := False;
36
37 task T1 is
38 entry Start;
39 entry Quit;
40 end T1;
41
task T2;
42
task body T1 is
43 begin
44 while not Bool loop
45 Wait_For_Release(SB, Notified);
46 select
47 accept Start;
48 Put_Line("Y+Z=" & Integer’Image(Y+Z));
49 or
50 accept Quit;
51 Bool := True;
52 or
53 terminate;
54 end select;
55 end loop;
56 end T1;
57
task body T2 is
58 K : Integer := 5;
59 begin
60 Z := Exp(A(K));
61 end T2;
62
63 begin
64 loop
65 Wait_For_Release(SB, Notified);
66 Get(X);
67 exit when X = 100;
68 delay 0.5;
69 Y := Add(X);
70 T1.Start;
71 end loop;
72 T1.Quit;
73 end Example;
74

3 Program Dependences and Interprocedural Relations

3.1 An Overview

By means of constructing formal representation of DUNs of Ada 2012 programs, we can analyze, identify, and define various types of primary program dependences and interprocedural relations in the program [14]. In this section, based on concurrent Ada 2012 programs, we define five primary program dependences, such as control dependence, data dependence, selection dependence, communication dependence, and synchronization dependence, present four types of interprocedural relations, and then based on primary program dependences, we give definitions of new types of program dependences in Ada 2012 programs.

3.2 Program Dependences

Definition 1. Let \( (V, N_s, P_F, P_J, A_{in}, A_{out}, F_{in}, F_{out}, A_C, A_N, A_P, A_P, A_C, s, t) \) be the CFN of a concurrent program, and \( u \in V, v \in \{V - (N_s \cup P_F \cup P_J \cup F_{in} \cup F_{out})\} \) be any two vertices of the net. \( u \) is directly strongly control dependent on \( v \) iff there exists a path \( P = (v_1 = v, v_2, (v_2, v_3), \ldots, (v_{n-1}, v_n = u) \) from \( v \) to \( u \) such that \( P \) does not contain the immediate forward dominator of \( v \) and there exists no vertex \( v' \) in \( P \) such that the path from \( v' \) to \( u \) does not contain the immediate forward dominator of \( v' \). \( u \) is directly weakly control dependent on \( v \) iff \( v \) has two successors \( v' \) and \( v'' \) such that there exists a path \( P = (v_1 = v, v_2, (v_2, v_3), \ldots, (v_{n-1}, v_n = u) \) from \( v \) to \( u \) and vertex \( v_i (1 < i \leq n) \) in \( P \) strongly forward dominates \( v' \) but does not strongly forward dominate \( v'' \).

Not that according to the above definition, if \( u \) is directly strongly control dependent on \( v \), then \( u \) is also directly weakly control dependent on \( v \), but the converse is not necessarily true.

Informally, if \( u \) is directly strongly control dependent on \( v \), then \( v \) must have at least two successors \( v' \) and \( v'' \) such that if the branch from \( v \) to \( v' \) is executed then \( u \) must be executed, while if the branch from \( v \) to \( v'' \) is executed then \( u \) may not be executed. If \( u \) is directly weakly control dependent on \( v \), then \( v \) must have two successors \( v' \) and \( v'' \) such that if the branch from \( v \) to \( v' \) is executed then \( u \) is necessarily executed within a fixed number of steps, while if the branch from \( v \) to \( v'' \) is executed then \( u \) may not be executed or the execution of \( u \) may be delayed indefinitely. The difference between strong and weak control dependences is that the latter reflects a dependence between an exit condition of a loop and a statement outside the loop that may be executed after the loop is exited, but the former does not. For example, in Figure 1, vertices 47, 48, 49, and 58 are directly strongly (weakly) control dependent on vertex 47; vertex 59 is directly weakly control dependent on vertex 47 but not directly strongly control dependent on 47; vertices 68, 69, 70, 71, 72, 73, 74, and 75 are directly strongly (weakly) control dependent on vertex 71; vertices 76 and 77 are directly weakly control dependent on vertex 71 but not directly strongly control dependent on 71.
Definition 2. Let \((V, N, P_F, P_J, A_{in}, A_{out}, F_{in}, F_{out}, A_C, A_{N}, A_{PF}, A_{PJ}, A_{CA}, s, t)\) be the CFN of a concurrent program, and \(u \in V, v \in N\) be any two vertices of the net. \(u\) is directly selection dependent on \(v\) iff (1) there exists a path \(P = (v_1 = v, v_2), (v_2, v_3), \ldots, (v_{n−1}, v_n = u)\) from \(v\) to \(u\) such that \(P\) does not contain the immediate forward dominator of \(v\), and (2) there exists no vertex \(v_i (1 < i < n)\) in \(P\) such that the path from \(v_i\) to \(u\) does not contain the immediate forward dominator of \(v_i\).

Informally, if \(u\) is directly selection dependent on \(v\), then \(v\) must have some successors such that if the branch from \(v\) to one of the successors is executed then \(u\) must be executed, while if another branch is executed then \(u\) may not be executed. For example, in Figure 1, vertices 50, 51, 53, and 54 are directly selection dependent on vertex 49.

The difference between the direct (strong or weak) control dependence and the direct selection dependence is that the former defines a kind of program dependence holding between the control predicate of a conditional branch statement and a statement whether it is executed determined by the truth value of the control predicate, but the latter defines a kind of program dependence holding between a nondeterministic selection statement and a statement whether it is executed determined by the nondeterministic selection.

Definition 3. Let \((N_C, \Sigma_V, D, U, \Sigma_C, S, R)\) be the DUN of a concurrent program, where \(N_C\) is the CFN \((V, N, P_F, P_J, A_{in}, A_{out}, F_{in}, F_{out}, A_C, A_{N}, A_{PF}, A_{PJ}, A_{CA}, s, t)\) of the program, and \(u\) and \(v\) be any two vertices of the net. \(u\) is directly synchronization dependent on \(v\) iff any of the following conditions holds:

1. \((v, u) \in A_{PF} \cup A_{PJ}, \text{i.e., } (v, u)\) is a parallel execution arc,
2. \(S(v) = R(u)\), or
(3) there exists a vertex \( v' \) such that \( v' \) directly synchronization dependent on \( v \), \( u \) is the last continuous forward dominator of \( v' \), and \( S(v') = \phi \) and \( R(v') = \phi \) for any vertex \( v'' \) (excluding \( v' \)) in the path from \( v' \) to \( u \).

Informally, if \( u \) is directly synchronization dependent on \( v \), then the start and/or termination of execution of \( v \) directly determines whether or not the execution of \( u \) starts and/or terminates. For example, in Figure 1, vertices 45, 61, 62 are directly synchronization dependent on vertex 67; vertex 68 is directly synchronization dependent on both vertices 46 and 63; vertex 70 is directly synchronization dependent on vertex 48; vertex 49 is directly synchronization dependent on 69; vertex 53 is directly synchronization dependent on 76; vertex 50 is directly synchronization dependent on 74; vertex 75 is directly synchronization dependent on 51; vertex 77 is directly synchronization dependent on vertices 54, 59, and 65.

The difference between the direct (strong or weak) control dependence and the direct synchronization dependence is that the former is irrelevant to the execution timing of a program but the latter is intrinsically relevant to the execution timing.

**Definition 5.** Let \((N_C, \Sigma_V, D, U, \Sigma_G, S, R)\) be the DUN of a concurrent program, where \( N_C \) is the CFN \((V, N_s, P_F, P_t, A_{in}, A_{out}, F_{in}, F_{out}, A_C, A_{in}, A_{F}, A_{P}, A_{P}, A_{C}, s, t)\) of the program, and \( u \) and \( v \) be any two vertices of the net. \( u \) is directly communication dependent on \( v \) iff there exist two vertices \( v' \) and \( v'' \) such that \( u \) is directly data dependent on \( v' \), \( R(v') = S(v'') \), and \( v'' \) is directly data dependent on \( v \).

Informally, if \( u \) is direct communication dependent on \( v \), then the value of a variable computed at \( v \) has direct influence on the value of a variable computed at \( u \) by an inter-process communication. For example, in Figure 1, vertex 51 is direct communication dependent on vertices 64 and 73.

The difference between the direct data dependence and the direct communication dependence is that the direct data dependence is irrelevant to communication channels of a program but the direct communication dependence is intrinsically relevant to the channels.

### 3.3 Interprocedural Relations

To handle interprocedure calling and parameter passing issues, Horwitz et al. modeled System Dependence Graph (SDG) [32]. In Ada programs, there are four types of interprocedural relations [17, 26]:

- if statement \( v \) is calling a subprogram, the start statement \( u \) of the subprogram is said to be call-related with \( v \). For example, in Figure 1, vertex 31 is call-relation with vertices 64 and 31; vertex 26 is call-relation with vertex 73
- the formal parameter \( u \) labeled with \( F_{in} \), is said to be parameter-in-related with the actual parameter \( v \) corresponding to \( u \). For example, in Figure 1, vertex 31 is also parameter-in-related with vertices 64 and 31; vertex 26 is call-relation with vertex 73
- the actual parameter \( u \) is said to be parameter-out-related with the formal parameter \( v \) labeled with \( F_{out} \) corresponding to \( u \)
- the parameter/statement \( u \) is said to be returned-value-related with a return statement \( v \) corresponding to the function call, such that a returned value from \( v \) directly affects the variables assigned at \( u \). For example, in Figure 1, vertex 64 is also returned-value-related with vertex 31; vertex 73 is returned-value-related with vertex 24

### 3.4 New Types of Program Dependences in Ada 2012 Programs

On account of changes and extensions introduced by Ada 2012, some new types of program dependences have been found in [27], such as precondition dependence, postcondition dependence, predicate dependence, expression dependence, task-barriers dependence.

- Precondition dependence, postcondition dependence, and predicate dependence comes from extensions of “contract-based programming” [20, 24, 25]. In Ada 2012, a precondition is an obligation on the caller to ensure that it is true when the subprogram is called and it is a guarantee to the implementer of the body that it can be relied upon on entry to the body [24]. A postcondition is an obligation on the implementer of the body to ensure that it is true on return from the subprogram and it is a guarantee to the caller that it can be relied upon on return [24]. Similarly, Ada 2012 introduced assertions for types and subtypes with predicates.
- Expression dependence originates from expression functions, which can parameterize an expression without the formality of providing a function body. In this way, we can express an if, case, quantification expression, even a function in one statement that is an expression.
- Task-barriers dependence is considered from a package of Ada. Synchronous_barriers to make the tasks given to be waited for by using a discriminant and to be released together [20]

However, there are no definitions about the new program dependences in the paper [27], whereby predicate dependences are relevant to types and belong to dependent types [33], and thus they cannot be defined by control flow or data flow, such that there is no predicate dependence in any DUNs. Here, we will give formal definitions of the other four types.

**Definition 6.** Let \((N_C, \Sigma_V, D, U, \Sigma_G, S, R)\) be the DUN of a concurrent program, where \( N_C \) is the CFN \((V, N_s, P_F, P_t, A_{in}, A_{out}, F_{in}, F_{out}, A_C, A_{in}, A_{F}, A_{P}, A_{P}, A_{C}, s, t)\) of the program, and \( u \) and \( v \) be any two vertices of the net. \( u \) is directly precondition dependent on \( v \) iff (1) there exists a path \( P = (v_1 = F_{in}, v_2), \ldots, (v_{n-1}, v_n = A_{in}) \), (2) \( v \) in \( P \) is the immediate forward dominator of \( v_1 \), and the \( t \) of the net is the immediate forward dominator of \( v \), and (3) vertex \( u \) in \( P \) is directly strongly control dependent on \( v \).

Informally, if \( u \) is directly precondition dependent on \( v \), then \( v \) must have two successors such that if the branch from \( v \) to
one of the successors is executed then \( u \) must be executed, that is meaning that the requirement of precondition meets, which is true on entry, whereas, on the other branch, \( v \) raises a program exception on precondition, the control flow is running to termination, \( u \) is never executed. For example, in Figure 1, vertices 27, 28, 24, and 73 are directly precondition dependent on vertex 23.

The differences between the direct (strong or weak) control dependence and the direct precondition dependence is that the former is irrelevant to runtime checks, and must appear in the control flow but the latter is intrinsically relevant to the runtime checks optionally by means of a switch.

**Definition 7.** Let \((N_C, \Sigma_V, D, U, \Sigma_C, S, R)\) be the DUN of a concurrent program, where \(N_C\) is the CFN \((V, N_s, P_F, P_J, A_{in}, A_{out}, F_{in}, F_{out}, A_C, A_{N_s}, A_{P_F}, A_{P_J}, A_{C_A}, s, t)\) of the program, and \(u\) and \(v\) be any two vertices of the net. \( u \) is directly postcondition dependent on \( v \) iff (1) there exist two different paths \( P_1 \) and \( P_2 \) and three different vertices \( u', w, \text{ and } w' \), (2) \( u \) is the immediate forward dominator of \( u' \) in \( P_1 \) and \( w \) is the immediate forward dominator of \( w' \) in \( P_2 \), (3) \( u' \) and \( w' \) are directly data dependent on \( v \), and (4) \( u \) is directly synchronization dependent on \( w' \) and \( w \) is directly synchronization dependent on \( u' \).

Informally, if \( u \) is direct task-barriers dependent on \( v \), then \( v \) must define a specified count value. After the number of blocked tasks gets to the value, synchronously release a group of tasks. Each call to \texttt{Wait\_For\_Release} blocks the calling task until the number of blocked tasks associated with the Synchronous Barrier Object is equal to Release Threshold, at which time all blocked tasks are released. For example, in Figure 1, vertices 49 and 70 are direct task-barriers dependent on vertex 11, representing that line 49 and line 70 can be executed when the number of synchronous barrier object is equal to Release Threshold defined on line 11.

In Ada 2012 programs, after the activation of the task is completed, the statements of the parent block begin to execute. Therefore, the statement immediately following the declarative part of the task body, i.e., a vertex representing the reserved word \texttt{begin} of the task body, is labeled with send message function \( S(v) \), such as A-9, A-14, A-18 in Table 2, while the first statement of the parent block, i.e., the first statement under the \texttt{begin} of the parent block, such as A-30, is labeled with the receiving message function \( R(v) \), which is meaning that the child tasks complete the activations. The name of the symbol is the same as the parent block name.

### 4 System Dependence Nets of Ada 2012 Programs

As above, based on graph-theoretical, when we present the definitions of various types of program dependences and interprocedural relations, we can construct a formal model (SDN) to depict dependence-based program representation explicitly.

The SDN of a concurrent Ada 2012 program is a kind of arc-classified digraph to represent program dependences and interprocedural relations, such that each type of arc explicitly denotes a type of program dependence or interprocedural relation, as well as each node indicates a statement at both ends of the arc.

In Ada 2012 programs, the SDN is defined as an arc-classified digraph, as \((V_{DUN}, \text{Con}, \text{Dat}, \text{Sel}, \text{Sym}, \text{Com}, \text{Pre}, \text{Pos}, \text{Exp}, \text{Tas}, \text{Cal}, P_{in}, P_{out}, \text{Ret})\), where

- \(V_{DUN} \in (V \cap A_{in} \cap A_{out} \cap F_{in} \cap F_{out})\) to represent the node set of DUN
- \(\text{Con}\) is the set of control dependence arcs such that any 2-tuple \((u, v) \in \text{Con}\) if and only if \( u \) is directly control dependent on \( v \)
- \(\text{Dat}\) is the set of data dependence arcs such that any 2-tuple \((u, v) \in \text{Dat}\) if and only if \( u \) is directly data dependent on \( v \)
Definition-Use Net and System Dependence Net Generators for Ada 2012

5 Method of Generating DUNs for Ada 2012 Programs

5.1 Requirements Analysis of DUNs of Ada 2012 Programs

In order to generate a DUN for an Ada 2012 program automatically, we compute vertices, labels, and arcs as described in Section 2, such that they are regarded as the output of the DUN generator. For vertices, we just need to judge what statement the line number is. A DUN generator should satisfy the following requirements.

- **Sel** is the set of selection dependence arcs such that any 2-tuple \((u, v) \in Sel\) if and only if \(u\) is directly selection dependent on \(v\)
- **Syn** is the set of synchronization dependence arcs such that any 2-tuple \((u, v) \in Syn\) if and only if \(u\) is directly synchronization dependent on \(v\)
- **Com** is the set of communication dependence arcs such that any 2-tuple \((u, v) \in Com\) if and only if \(u\) is directly communication dependent on \(v\)
- **Pre** is the set of precondition dependence arcs such that any 2-tuple \((u, v) \in Pre\) if and only if \(u\) is directly precondition dependent on \(v\)
- **Pos** is the set of postcondition dependence arcs such that any 2-tuple \((u, v) \in Pos\) if and only if \(u\) is directly postcondition dependent on \(v\)
- **Exp** is the set of expression dependence arcs such that any 2-tuple \((u, v) \in Exp\) if and only if \(u\) is directly expression dependent on \(v\)
- **Tas** is the set of task-barriers dependence arcs such that any 2-tuple \((u, v) \in Tas\) if and only if \(u\) is directly task-barriers dependent on \(v\)
- **Cal** is the set of call relation arcs such that any 2-tuple \((u, v) \in Cal\) if and only if \(u\) is call related with \(v\)
- **Pin** is the set of parameter-in relation arcs such that any 2-tuple \((u, v) \in Pin\) if and only if \(u\) is parameter-in related with \(v\)
- **Pout** is the set of parameter-out relation arcs such that any 2-tuple \((u, v) \in Pout\) if and only if \(u\) is parameter-out related with \(v\)
- **Ret** is the set of returned-value relation arcs such that any 2-tuple \((u, v) \in Ret\) if and only if \(u\) is returned-value related with \(v\)

As thus, by using various types of arcs, the SDN of an Ada 2012 program can explicitly represent control dependence, data dependence, selection dependence, synchronization dependence, communication dependence, precondition dependence, postcondition dependence, expression dependence, task-barriers, call-relation, parameter-in-relation, parameter-out-relation, and returned-value-relation, respectively. Figure 2 shows the SDN of the Ada 2012 program Example, corresponding to its DUN in Figure 1.
Algorithm 2 Compute nondeterministic selection arcs

Input: compilation units
Output: nondeterministic selection arcs

for every task/block/procedure T do

if T has select statements then
    for every vertex v as every select statement of T do
        connect a nondeterministic selection arc to from v to the first statement of every select alternative corresponding to v
    end for
end if

if T has requeue with abort then
    for every vertex v as every a requeue with queue of T do
        connect a nondeterministic selection arc from v to the entry call statement corresponding to v [requeue to the corresponding entry call]
        connect a nondeterministic selection arc from v to v itself [abort due to an abort or the expiration of a delay of the entry call]
    end for
end if
end for

compilations. Each compilation is a succession of compilation units. A compilation unit contains either a declaration, a body, or a renaming of a program unit [25]. A program unit is either a package, a task unit, a protected unit, a protected entry, a generic unit, or an explicitly declared subprogram other than an enumeration literal. Certain kinds of program units can be separately compiled. Alternatively, they can appear physically nested within other program units [25]. Here, we should traverse every task/block/procedure to compute arcs.

5.3 Implementation of a DUN Generator for Ada 2012 Programs

The Ada Semantic Interface Specification (ASIS) [28] is an ISO standard that defines an interface between Ada environments. We developed an ASIS-based tool to cope with syntax and semantics of Ada 2012. The tool can get considerable information through the ASIS interface, which is installed as an Ada library. The functions of the DUN generator are to generate DUNs of compilation units in the Ada environment. Figure 3 shows a generation flow of DUNs for Ada 2012 programs.

In order to generate DUNs from Ada 2012 programs, there are six functional components in a definition-use generator. The core component, called Ada2DUN, is invoking other five components, DUN_Handler, Gela_Ids, Stacks, V_Strings, and Id_List.

The component Ada2DUN can traverse a target Ada 2012 program to generate a DUN.

Component DUN_Handler encapsulates 77 functions and procedures to handle DUNs as definitions and requirements. The Gela_Ids [34] encapsulates a set of operations and queries that implement the ASIS Id abstraction. An Id is a way of identifying a particular Element, from a particular Compilation_Unit, from a particular Context. Ids can be written to files. Ids can be read from files and converted into an Element.
value with the use of a suitable open Context. By using Id, the Gela_Ids uses hash of an element concatenated with unit name.

Also, the Stacks encapsulates a generic stack handler package. The V_Strings encapsulates some strings handler functions. Finally, the Id_List supports some functions for handling a list of Ids.

The generator of DUNs contains the following processes, i.e., an ASIS application must use the following steps [35]:

S1 Asis.Implementation.Initialize (...); -- Initialize ASIS.
S2 Asis.Ada_Environments.Associate (...); -- Associate ASIS.
S3 Asis.Ada_Environments.Open (...); -- Open a context.
S4 Get the name of the target unit for the user, and get a compilation unit of the name.
S5 Process elements contained in each compilation unit and generate DUN.
S6 Asis.Environments.Close (...); -- Close the context.
S7 Asis.Environments.Dissociate (...); -- Process the whole DUN.

Figure 3: A generation flow of DUNs for Ada 2012 programs

```
S8 Asis.Implementation.Finalize (...); -- Finalize ASIS.
S9 Output the DUN as the text file.
```

We give an output of this generator for the example program of Figure 1. The elapsed time is Clock 58156.807824882—58158.226458761 on Windows 7 with Intel Core i7-860 Processor (2.8 GHz, 4 cores, 8 threads), and 4 Gbyte memory.

A text output List 2 of this generator for the example program of Figure 1 as follows.

**Listing 2: A text output of the DUN generator for Example**

1. <line> bh–8 <connect> 2
2. <line> bh–11 <connect> 3 <def> Synchronous_Barrier, MAIN.Example5 SB–1
3. <line> bh–12 <connect> 4 <def> Boolean, MAIN.Example5 Notified–2
4. <line> bh–35 <connect> 5 <def> INTEGER, MAIN.Example5 Y–4 INTEGER, MAIN.Example5 X–3
5. <line> bh–36 <connect> 6 <def> INTEGER, MAIN.Example5 Z–5
6. <line> bh–37 <connect> 7 <def> BOOLEAN, MAIN.Example5 Bool–6
7. <line> bh–68 <connect> 8 <fork> 32 21
8. <line> bh–17 <receive> MAIN.Example5
9. <line> bh–23 <pre> connect 11
10. <line> bh–24 <Pre> connect 42
37. <line> bh–69 connect 38
38. <line> bh–70 connect 39
39. <line> bh–71 connect 40
40. <line> bh–72 connect 41 45
41. <line> bh–73 connect 42
42. <line> bh–74 connect 43 <use> INTEGER, MAIN.Example5.Add–7 call 9 <ca–in>
43. <line> bh–75 connect 44 <send> MAIN.Example5.T1.Start
44. <line> bh–76 <connect> 37
Here, we input an open source of Ada 2012 programs. Ada Web Server (AWS) is an Ada-based web server, which can be embedded allowing your application to talk with all modern web browsers [36]. We chose Ada Web Server to evaluate the various types of program dependences. We propose algorithms to compute forward-dominator tree, control dependence, data dependence, synchronization dependence, communication dependence, selection dependence, precondition dependence, postcondition dependence, and expression dependence for Ada 2012 programs.

6 Method of Generating SDNs for Ada 2012 Programs

We can capture various program dependences in Figure 4, representing a data flow for generating program dependences from the DUN in Ada 2012 programs.

By means of inputting DUNs of target programs, we can get various types of program dependences. We propose algorithms to compute forward-dominator tree, control dependence, data dependence, synchronization dependence, communication dependence, selection dependence, precondition dependence, postcondition dependence, and expression dependence for Ada 2012 programs.

Algorithm 4 Compute forward-dominator tree

Input: definition-use net
Output: forward-dominator tree

If vertex \( v \) only has one successor then
the forward-dominator of \( v \) is the successor
else
find all the passes from \( v \) to the end vertex
find the common path of all the passes from \( v \) to the end vertex
the first vertex of the common path is the forward-dominator of \( v \)
end if

Algorithm 5 Compute control dependence

Input: definition-use net
Output: control dependence arcs

Make the forward-dominator tree
for each vertex in the DUN do
if the control flow begin from the vertex \( v \) is at least two then
make the parent list of the vertex \( v \) based on the forward dominator
for each vertex \( u \) in the branch of the vertex do
make the parent list of the vertex \( u \) based on the forward dominator
end for
if the vertex \( v \) is in the parent list of vertex \( u \) or the any parent of vertex \( v \) is in the parent list of vertex \( u \) then
vertex \( u \) is control dependent on (vertex \( u \) and parent list of vertex \( v \) - parent list of vertex \( u \))
end if
end if
end for

Figure 4: A data flow for generating program dependences from the DUN in Ada 2012
Algorithm 6 Compute data dependence
Input definition-use net
Output data dependence arcs
repeat for each vertex v and its successor u do if the variable x is defined in v then push v into chain[u][x] else push chain[v][x] into chain[u][x] end if end for until the array chain is changed during the block above for each vertex u do for each vertex y used in u do draw the arc that shows y to chain[y][w] to show the data dependence end for end for

Algorithm 7 Compute synchronization dependence
Input definition-use net, control dependence arcs
Output synchronization dependence arcs
for each vertex v that contains fork/join arc(s) in DUN do for each vertex u of the successor of the fork/join arcs do use connectsync (u, v) to draw synchronization dependences end for end for
for each vertex v do for each node u that has the same channel in their receiving do if the set of sending of v equals the set of receiving of u then use connectsync (u, v) to draw synchronization dependences end if end for end for
connectsync (u, v)
draw the arc that shows u synchronization depends on v while u has only one successor do u’ = the successor of u if u has the set of sending or receiving then break end if draw the arc that u’ depends on v end while

Algorithm 8 Compute communication dependence
Input definition-use net, control dependence arcs
Output communication dependence arcs
for each vertex v that has set of sending do for each vertex u that the set of receiving equals to the set of sending of v do for each vertex start node that u depends on do for each vertex end node that depends on u do draw the arcs that show end node depends on start node end for end for end for end for

Algorithm 9 Compute selection dependence
Input definition-use net, control dependence arcs
Output selection dependence arcs
for each nondeterministic vertex v do for each vertex u that control depends on v do add the arc that shows u selection depends on v delete the arc that shows u control depends on v end for end for

Algorithm 10 Compute precondition dependence
Input definition-use net, control dependence arcs
Output precondition dependence arcs
for each vertex v in the DUN do if u is the immediate forward dominator of a vertex F, t is the immediate forward dominator of v then if u strong control depends on v then add the arc that shows u precondition depends on v delete the arc that shows u control depends on v end if end if end for

Algorithm 11 Compute postcondition dependence
Input definition-use net, control dependence arcs
Output postcondition dependence arcs
for each vertex labeled with \(A_{\text{in}}(u)\) do if u is the immediate forward dominator of v then if t is the immediate forward dominator of v then add the arc that shows u postcondition depends on v end if end if end for

Algorithm 12 Compute expression dependence
Input definition-use net
Output Expression dependence arcs
for each vertex v do if \(F_{\text{in}}(v) = U(v)\) then if \((v, v) \in \mathcal{A}_{\text{I}}\) then add the arc that shows v expression depends on v end if end if end for

Here, we show a text output List 3 of the SDN generator corresponding to the input file of the DUN of Example program of Figure 1.

Listing 3: A text output of the SDN generator for Example

Also, we could output the result on demand with some switches on the console, such as "--c", "--d", "--syn", "--com", "--se", "--pre", "--post", "--exp", "--pout", "--pin", "--call" corresponding to separately outputting control dependence, data dependence, synchronization dependence, communication dependence, selection dependence, precondition dependence, postcondition dependence, expression dependence, parameter-out-relation, parameter-in-relation, call-relation, if any.

7 Applications of DUNs and SDNs

7.1 Applications of DUNs

The DUN of a concurrent program can provide a clear and precise basis for definitions of notions, descriptions of methods, and developments of tools in software engineering. Some applications dependent on the DUN are as follows.

First, the DUN provides a basis for defining software test coverage criteria and generating software test data.

Second, having the DUN as representations of concurrent programs, many well-known complexity metrics of sequential programs can be redefined for concurrent programs based on the representation [18].

Third, the DUN can be used as program understanding tools in software maintenance and re-engineering.

7.2 Applications of SDNs

7.2.1 Slicing

The most direct and important use of SDNs will be program slicing. The explicit representation of the various program dependences in programs makes SDN ideal for program slicing.

Program slicing was first proposed by Weiser [5, 37]. It is a method for automatically decomposing programs. It provides a reduced program by erasing irrelevant statements in the original program based on certain statements and a set of variables, called the “criterion”. The reduced program is called a “slice” based on the criterion. He claimed that experienced programmers always create a “slice” in their mind to find the suspect which causes the error. And with program slicing technology, creating the “slice” based on certain criterion can be done automatically and helps programmers by narrowing down the possible scope of the cause of error. During about 30 years’ researches, the most popular approach [19] to process program slices are based on SDNs. Program slicing based on SDN will be simplified as a reachability problem on a digraph.

7.2.2 Testing

Testing is the process that executes the program with the intention of finding errors. Since SDNs can represent the data flow properties of the program, the dependence-coverage of programs can be found by using SDN.

7.2.3 Understanding and Maintenance

To understand a program, we always intent to find which variable in which statement might affect a variable of interest. With a certain slice of the program, the set of the statements and the variables which affect the variable of interest can be easily found. As we mentioned above, to create the slice, we have to build the SDN of the program.

In program maintenance, the problem which is called “ripple effect” [38] is well-known as: whether changing a code in a program will affect the behavior of other codes of the program, which may cause new error(s). With the slice of the changing code, we can find all the codes in the program that might be affected. It is obviously useful in program maintenance to build slices. As we mentioned above, to build slices, the SDN will come in handy.

7.2.4 Complexity Measurement

Software metrics have many uses in software engineering, such as program understanding, debugging, testing, analysis, maintenance, and so on [18]. Based on SDN, we can define a set of metrics for measuring the complexity of programs from several viewpoints. For example, the metric defined by the sum of all program dependences between statements can be used to measure the complexity of corresponding program. The metric defined by the sum of communication dependences can be used to measure the complexity of concurrency in corresponding program. And the proportion of the communication dependences in a program can be used to measure the degree of concurrency in corresponding program.

8 Concluding Remarks

This paper presented how to construct and generate DUNs and SDNs of Ada 2012 programs. Based on graph theory, we modeled a complete representations of DUNs and SDNs of Ada 2012 programs including new types of program dependences and proposed methods to generate DUNs and SDNs of Ada 2012 programs. We developed a definition-use net generator and a system dependence net generator for Ada 2012 programs, which could compute control flow and data flow, and all types of program dependences and interprocedural relations of SDNs of Ada 2012 programs, respectively.

There are three limitations of the DUN generator. First, it is about Exception handler. The DUN generator ignores exceptions, if any. Because as a static tool, it cannot model
an exception raised and exception propagation raised at run-
time, and we would like to process it like SPARK (subset of
Ada) [39] programming language without exception handlers.
Second, it cannot process task types, because of dynamic
task creation with allocators, which need dynamic running
history method. Finally, it is about tagged types. A tagged
type provides support for dynamic polymorphism and type
extension. Because of hidden tag, a tagged type identifies the
type at run-time, therefore, it is dependent on type dependence
and none of DUN.

References

dependence graph in a software development environment,
ACM Transaction on Programming Languages and Systems,
vol. 19, no. 5, pp. 177–184.

dependence graph and its use in optimization,
ACM Transaction on Programming Languages and Systems,
vol. 9, no. 3, pp. 319–349.

(2010), Assessing the impact of global variables on
program dependence and dependence clusters,


[8] J. Krinke (2001), Identifying similar code with program

tailoring, and suggesting form template method refactoring opportunities with program dependence graph, in 16th European Conference on Software Maintenance and Reengineering (CSMR), pp. 53–62, IEEE.


Proven Test Solutions for Reliable Embedded Software

"VectorCAST is unique in that it provides us with the ability to increase the reliability and quality of our flight software."
-- Honeywell

Vector Software, Inc.
Golden Cross House | 8 Duncannon Street | London WC2N 4JF UK | +44 203 603 0120 | sales@vectorcast.com
SPARK 2014 Rationale: Type Predicates, Variables that are Constant, Support for Ravenscar and Support for Type Invariants

Yannick Moy, Claire Dross
AdaCore, France

Abstract
This paper continues the publication of the "SPARK 2014 Rationale", which started in the December 2013 issue of the Ada User Journal. In this instalment, we present four new contributions regarding type predicates, variables that are constant, support for Ravenscar and support for type invariants.

1 Type Predicates

Yannick Moy

Preconditions and postconditions define a very strong mechanism for specifying invariant properties over the program's control. What about similar properties for the program's data? It turns out Ada 2012 defined such a construct, type predicates, which was not supported in SPARK until now. And now it is.

With type predicates, one can express all the invariant properties of data that could not be expressed with type ranges, non-nullness constraints, discriminants, etc. For example, are you doing cryptography and you want to define a type for prime numbers? You can with type predicates:

```plaintext
(type
   Prime is new Positive with
     Predicate => (for all Divisor in 2 .. Prime / 2 =>
       Prime mod Divisor /= 0));
```

The predicate above reads: "for all values of divisor between 2 and half the value of a prime number, the divisor does not divide the prime number". Which is indeed a possible definition of prime numbers.

Closer to a problem reported recently by a customer, are you building control software for a chip that does not support subnormal floating-point numbers, and you want to define a type for normal floating-point numbers only? You can with type predicates:

```plaintext
(subtype Normal_Float is Float with
  Predicate =>
    Normal_Float <= -2.0**(-126) or
    Normal_Float = 0.0 or
    Normal_Float >= 2.0**(-126));
```

Whether you're doing cryptography or control software, in both cases you'll get an error at run time if you try to assign a value that does not respect the predicate into a variable of that type. With GNATprove, you can additionally prove that such run-time errors can never occur in your programs, hence that the predicates are always respected by the data.

Interestingly, predicate checks are the first kind of run-time checks for which SPARK rules mandate more verifications than the Ada rules. Indeed, Ada rules are on purpose not bulletproof. It would be hard in Ada, because any Boolean expression is allowed as predicate, so we can write:

```plaintext
(type My_Type is new Integer with
  Predicate => Global_Var /= 0);
```

But then, if I assign value 0 in Global_Var somewhere in the code, I'm suddenly violating the predicates of all values of type My_Type. I could even hide the predicate behind function calls to further complicate the matter. Hence, when they created predicates in Ada, the ARG committee chose to request that only some run-time checks are performed to verify that predicates are respected, without trying to detect all possible violations.

In SPARK, we took the opposite path, restricting expressions that can be used as predicates so that we can perform bulletproof formal verification that no violation can occur. In the SPARK Reference Manual, the restriction is expressed quite succinctly: "A Dynamic_Predicate expression shall not have a variable input." In particular, this means that the predicate of My_Type above is not allowed in SPARK.

Even with this restriction, the Ada rules are not sufficient to make sure a predicate is never violated. In particular, Ada does not require that the predicate is checked after assignment to a record component. For example, assume you have a type for pairs of distinct integer values:

```plaintext
(type
  Distinct_Pair is record
    Val1, Val2 : Integer;
  end record
  with Dynamic_Predicate =>
    Distinct_Pair.Val1 /= Distinct_Pair.Val2;
)
```

Ada rules do not disallow the following code (although it violates the spirit of using type predicates) which temporarily violates the predicate above:

```plaintext
D : Distinct_Pair := (1, 2);
D.Val2 := 1;
<do something which may read D's value>
D.Val2 := 2;
```
In SPARK, we have defined a verification rule that forces the predicate to hold when assigning to component `Val2` above. Thus, GNATprove detects the possible violation in this code. Note that this is only a verification rule, which does not change how GNAT compiles this code.

For simplicity, I used the GNAT-specific aspect `Predicate` in the examples above. In fact, Ada defines two different aspects `Static_Predicate` and `Dynamic_Predicate`. See the references below for an explanation of their differences.

To know more about predicates in Ada, see the Ada Reference Manual [1] and John Barnes’s excellent rationale on the subject (section 5 on Subtype predicates) [2].

To know more about predicates in SPARK, see the SPARK User’s Guide [3].

2 Variables that are Constant

The SPARK tools now support yet another feature that allows users to better specify the intended behavior of their programs. This new feature enables users to declare that specific variables can only be updated during the elaboration of their enclosing package.

Aspect `Constant_After_Elaboration` can be used on a library-level variable to indicate that the variable must retain the value that it has after elaboration of its enclosing package throughout the entire life of the program. This means that no user of the package is allowed to update the variable’s value. The tools will issue warnings if the aspect is not respected. Since users of the package are prohibited from updating such variables, procedures that update them cannot be declared in the visible part of the package, they can only be declared in the package’s body so that they can be used within the package itself. Let’s have a quick look at some code:

```ada
package CAE is
   Var : Integer := 0
   with Constant_After_Elaboration;

   procedure Illegal;
end CAE;

package body CAE is
   procedure Illegal is
   begin
      Var := 10; -- Problem
   end Illegal;

   procedure Legal is
   begin
      Var := Var + 2; -- This is fine
   end Legal;
   begin
      Var := Var + 1;
      Legal;
   end CAE;
end
```

with `CAE; use CAE;

```ada
procedure User is
begin
   Var := 5; -- Problem
end User;
```

On the following code the tools issue two messages:

```
user.adb:3:11: high: constant after elaboration "Var" must not be an output of procedure "User"
cae.ads:5:14: high: constant after elaboration "Var" must not be an output of procedure "Illegal"
```

These messages inform us that the `Constant_After_Elaboration` contract of `Var` has been violated. Notice that procedure `Illegal` cannot be called by a user of `CAE` without resulting in a violation of the `Constant_After_Elaboration` aspect. So having procedure `Illegal` appear in the visible part of `CAE` makes no sense. On the other hand, procedure `Legal`, which also does update `Var`, is perfectly OK since it is declared in the body of `CAE` and therefore can be used during the elaboration of package `CAE` but cannot be called from users of `CAE`. This is the reason why the tools issue a message on procedure `Illegal` but not on procedure `Legal`. This new feature is particularly useful when tasking code is involved. Variables which have aspect `Constant_After_Elaboration` are guaranteed to be free of unsynchronised updates (since they are only ever updated during elaboration).

3 Support for Ravenscar

Yannick Moy

The upcoming release of SPARK Pro will support concurrency features of Ada, with the restrictions defined in the Ravenscar profile of Ada [4]. This profile restricts concurrency so that concurrent programs are deterministic and schedulable. SPARK analysis makes it possible to prove that shared data is protected against data races, that deadlocks cannot occur and that no other run-time errors related to concurrency can be encountered when running the program. Avoiding deadlocks in Ravenscar can be seen as a special form of run-time error to avoid, as the Ceiling Priority protocol in Ravenscar defines precisely which tasks are allowed to access which shared data and makes it a run-time error to perform an access which could create a deadlock. The main restriction to respect to fit in Ravenscar profile is that all concurrent objects (tasks and protected objects) should be defined statically at the top-level (that is, not created dynamically inside subprograms). I’ll reuse the example presented by Pavlos [5] to illustrate the main features of this support for concurrency in SPARK.

The preferred way to communicate between tasks in Ravenscar is through protected objects, as task rendez-vous (using task entries) are forbidden in Ravenscar (because they make schedulability analysis too difficult). A protected object declares public operations: functions which cannot modify the state of the protected object, procedures which can modify the state of the protected object and entries
components: encapsulating a value of this record type plus a few other

Functionally, a protected object maintains an invariant over the data it protects. As a refinement over the code presented by Pavlos, I'm going here to express the invariant that traffic lights maintain as a type predicate (see Section 1):

-- The following type represents the actual lights of the
-- traffic light. There are three lights for vehicles and two
-- for pedestrians. When a component is True the
-- corresponding light is On. When a component is
-- False the corresponding light is Off.

type Lights_State is record
  Vehicles_Green : Boolean := False;
  Vehicles_Yellow : Boolean := False;
  Vehicles_Red   : Boolean := True;
  Pedestrians_Green : Boolean := True;
  Pedestrians_Red  : Boolean := False;
end record;

function Valid_Comination (LS : Lights_State) return Boolean;

subtype Valid_Lights_State is Lights_State
with Predicate => Valid_Comination (Valid_Lights_State);

The protected object Traffic_Light is now mostly encapsulating a value of this record type plus a few other components:

protected Traffic_Light is
  entry Change_Lights;
  procedure Change_Lights;

private
  -- The following holds the time when the last state
  -- change occurred.
  Last_State_Change : Time := Time_First;

— The following is a boolean flag that indicates
— whether or not the time has arrived to change the
— state of the traffic light.
  Change_State : Boolean := False;

— The following variable represents the actual lights
— of the traffic light. There are three lights for
— vehicles and two for pedestrians.
  Lights : Valid_Lights_State;

end Traffic_Light;

Tasks call the protected subprograms to communicate. So task Change_The_Time calls Check_Time to update the inner state of the protected object so that the next activation of task Change_The_Lights updates the lights by calling Change_Lights which follows the inner automaton of Traffic_Lights. Because the two tasks only communicate through protected object Traffic_Light, there is no possible data race here, and no possible deadlock either, as correctly analyzed by GNATprove. The respect of the Ceiling Priority protocol is guaranteed here because the (default) priority of tasks Change_The_Time and Change_The_Lights is indeed lower than the (default) priority of protected object Traffic_Light. But this could be false if priorities were specified here explicitly, and in such a case GNATprove would detect it. For more details, see [3].

Like in the original code, an assumption is still required at the start of entry Change_Lights, for a different reason though. In the original code, the assumption was needed as a replacement for calling protected function Valid_Combination in the precondition. Indeed, as the precondition is logically outside the protected subprogram, getting the assurance that Valid_Combination holds in the precondition is not a protection against it not holding anymore when the protected subprogram is entered. Thus, Ada forbids such calls in preconditions. In our code, the invariant that Valid_Combination expresses is stated on the type of the component Lights of protected object Traffic_Light, so no precondition is needed. The assumption is only needed because GNATprove is not currently smart enough to assume the invariant properties of components of protected objects on entry of protected subprograms.

There is more in the support of concurrency in SPARK, in relation with suspension objects (lightweight form of protected objects, as if the data was a single boolean), task contracts and state abstraction. See the details in [3].

Finally, here is the complete program:

-- For the sake of this example the lights go as follows:
--
--  Vehicles                  Pedestrians
--  Green                    Red
--  Yellow                   Red
--  Red                      Green
--  Red and Yellow           Red
--  and over and over they go..

with Ada.Real_Time; use Ada.Real_Time;

package Traffic_Lights is
  -- The following type represents the actual lights of the
  -- traffic light. There are three lights for vehicles and
  -- two for pedestrians. When a component is True the
  -- corresponding light is On. When a component is
  -- False the corresponding light is Off.

type Lights_State is record
  Vehicles_Green : Boolean := False;
  Vehicles_Yellow : Boolean := False;
  Vehicles_Red   : Boolean := True;
  Pedestrians_Green : Boolean := True;
  Pedestrians_Red  : Boolean := False;
end record;
function Valid_Combination (LS : Lights_State) return Boolean;

subtype Valid_Lights_State is Lights_State with Predicate => Valid_Combination (Valid_Lights_State);

protected Traffic_Light is
  entry Change_Lights;
  procedure Check_Time;

private
  -- The following holds the time when the last state change occurred.
  Last_State_Change : Time := Time_First;

  -- The following is a boolean flag that indicates whether or not the time has arrived to change the state of the traffic light.
  Change_State : Boolean := False;

  -- The following variable represents the actual lights of the traffic light. There are three lights for vehicles and two for pedestrians.
  Lights : Valid_Lights_State;

end Traffic_Light;

task Check_The_Time;
  -- This task determines when it's time to change the traffic light.

task Change_The_Lights;
  -- This task is periodically notified to change the traffic light.

end Traffic_Lights;

package body Traffic_Lights is

function Valid_Combination (LS : Lights_State) return Boolean is
  (if LS.Vehicles_Green then
    not LS.Vehicles_Yellow
    and not LS.Vehicles_Red
    and not LS.Pedestrians_Green
    and LS.Pedestrians_Red
  elsif LS.Pedestrians_Green then
    not LS.Vehicles_Green
    and not LS.Vehicles_Yellow
    and LS.Vehicles_Red
    and not LS.Pedestrians_Red
  else
    not LS.Pedestrians_Green
    and LS.Pedestrians_Red
  end if);

protected body Traffic_Light is

  entry Change_Lights when Change_State is
    LS : Lights_State := Lights;

  begin
    pragma Assume (Valid_Combination (Lights));
    if LS.Vehicles_Green then
      LS.Vehicles_Green := False;
      LS.Vehicles_Yellow := True;
    elsif LS.Vehicles_Yellow and not
      LS.Vehicles_Red then
      LS.Vehicles_Yellow := False;
      LS.Vehicles_Red := True;
      LS.Pedestrians_Green := True;
      LS.Pedestrians_Red := False;
    elsif LS.Vehicles_Green and
      LS.Vehicles_Yellow then
      LS.Vehicles_Green := True;
      LS.Vehicles_Yellow := False;
      LS.Vehicles_Red := False;
    end if;
    Lights := LS;
    Change_State := False;
    Last_State_Change := Clock;
  end Change_Lights;

  procedure Check_Time is
    Wait_Duration : constant Time_Span :=
      (if Lights.Vehicles_Yellow then
        Seconds (2)
      else
        Seconds (15));
    begin
      if Clock - Last_State_Change >= Wait_Duration then
        -- We have waited enough. It is time for a state change...
        Change_State := True;
      end if;
    end Check_Time;

end Traffic_Light;

task body Check_The_Time is
  begin
    loop
      Traffic_Light.Check_Time;
    end loop;
  end Check_The_Time;
4 Support for Type Invariants

Claire Dross

Type invariants are used to model properties that should always hold for users of a data type but can be broken inside the data type implementation. Type invariants are part of Ada 2012 but were not supported in SPARK until SPARK Pro 17.

To demonstrate how they can be used, let us consider an implementation of binary trees as an example. As GNATprove does not support access types, we model them using indexes inside an array.

```ada
package Binary_Trees is
  type Index_Type is range 1 .. Max;
  subtype Extended_Index_Type is Index_Type'Base range 0 .. Max;
  type Position_Type is (Left, Right, Top);

  type Tree is private;
  private
    type Cell is record
      Left, Right, Parent : Extended_Index_Type := 0;
      Position : Position_Type := Top;
    end record;
    type Cell_Array is array (Index_Type) of Cell;
    type Tree is record
      Top : Extended_Index_Type := 0;
      C : Cell_Array;
    end record;
  end Binary_Trees;

  each cell contains the index of its right and left child, as well as the index of its parent. This index is 0 if the cell has no left or right child or no parent. It also contains a position which can be Top for the root of the tree and Left or Right for the other nodes, depending on whether they are left or right children in the tree structure. A tree contains an array of cells as well as the index of the tree root.

  There are properties that are imposed on the record fields by the tree structure. These properties are required for a record to represent a valid binary tree structure. For example, the root must have position Top and no parent:

  (for all i in Index_Type =>
    (if C ([i]).Left /= 0
      then C (C ([i]).Left).Position = Left
    and then C (C ([i]).Left).Parent = I))

  All these properties represent an invariant over the structure. They can be grouped together in an expression function which can then be attached to the full view of Tree using a Type_Invariant aspect:

  type Tree is record
    Top : Extended_Index_Type := 0;
    C : Cell_Array;
  end record
  with Type_Invariant => Tree_Structure (Tree);

  function Tree_Structure (T : Tree) return Boolean is
    (if T.Top /= 0 then T.C (T.Top).Parent = 0
     and then T.C (T.Top).Position = Top)
   and then
    (for all i in Index_Type =>
      (if T.C ([i]).Left /= 0
        then T.C (T.C ([i]).Left).Position = Left
      and then T.C (T.C ([i]).Left).Parent = I))
   end function

  ... In spirit, this means that Tree_Structure must always return True on objects of type Tree visible from outside Binary_Trees. To ensure this property, GNATprove enforces restrictions on subprograms working on trees depending on where they are declared. If the subprogram is private, like Tree_Structure, no invariant checks are required for its parameters, neither on entry nor on exit of the subprogram. In the invariant expression, only private functions should be used so as to avoid any circularity.

  If the subprogram is declared outside of Binary_Trees or if it is declared in the public part of Binary_Trees, then the invariant must hold for its input in entry of the subprogram and for its outputs in exit of the subprogram.

  For example, let us consider the Insert function which inserts a new node into a tree. Let us assume this is a boundary function for Tree, that is, it is declared in the public part of the specification of the package Binary_Trees in which Tree is declared:

  procedure Insert (T : in out Tree;
    I : Index_Type; D : Direction);

  The invariant is required to hold on input T when entering Insert. GNATprove will check Tree's invariant every time Insert is called inside Binary_Trees to make sure this is verified. In the same way, verification conditions are generated by GNATprove to ensure that the invariant holds for T at the end of Insert. In effect, it is like if we had written:

  procedure Insert (T : in out Tree;
    I : Index_Type; D : Direction) with
  Pre => Tree_Structure (T),
  Post => Tree_Structure (T);
```
Unlike type predicates (Section 1), type invariant can be broken temporarily in the body of Insert, as long as it is restored at the end of the subprogram:

```ada
procedure Insert (T : in out Tree;
  I : Index_Type; D : Direction) is
  M : Model_Type := Model (T) with Ghost;
  J : Index_Type;
begin
  -- Find an empty slot in the underlying array
  Find_Empty_Slot (T, J);
  -- Plug it as the D child of I
  T.C (J).Position := D;
  T.C (J).Parent := I;
  -- The invariant of T is broken, J is not the child of I
  if D = Left then
    T.C (I).Left := J;
  else
    T.C (I).Right := J;
  end if;
  -- Tree_Structure (T) holds again
end Insert;
```

Outside of Binary_Tree, the invariant of Tree is never checked. Indeed, the rules of SPARK are enough to ensure no invariant breaking value can leak out of Binary_Tree's implementation. This allows to split considerations between multiple layers. For example, we can then reuse our binary trees to implement search trees. We do not need to prove the tree structure invariant anymore, but can simply rely on it to prove the remaining properties:

```ada
package Search_Trees with SPARK_Mode is
  type Search_Tree is private;
  function Mem (T : Search_Tree; V : Natural) return Boolean;
  procedure Insert (T : in out Search_Tree;
    V : Natural; I : out Extended_Index_Type);
private
  type Value_Array is array (Index_Type) of Natural;
  type Search_Tree is record
    Struct : Binary_Trees.Tree;
    Values : Value_Array;
  end record
  with Type_Invariant => Ordered_Leafs (Search_Tree);
  function Ordered_Leafs (T : Search_Tree)
    return Boolean with Ghost;
end Search_Trees;
```

When calling Binary_Trees.Insert in the implementation of Search_Trees.Insert, GNATprove does not need to check that the invariant of T.Struct hold, as it is enforced at the boundary of Binary_Trees:

```ada
procedure Insert (T : in out Search_Tree;
  V : Natural; I : out Extended_Index_Type) is
begin
  if Top (T.Struct) = 0 then
    Init (T.Struct);
    I := Top (T.Struct);
    T.Values (I) := V;
    return;
  end if;

declare
  Current : Extended_Index_Type := Top (T.Struct);
  Previous : Extended_Index_Type := 0;
  D : Direction := Left;
begin
  while Current /= 0 loop
    Previous := Current;
    if V = T.Values (Previous) then
      I := 0;
      return;
    elsif V < T.Values (Previous) then
      D := Left;
    else
      D := Right;
    end if;
    Current := Peek (T.Struct, Previous, D);
  end loop;
  -- We have found the leaf where we want
  -- to insert V
  Insert (T.Struct, Previous, D);
  -- No invariant check
  -- The tree structure is preserved by Insert
  I := Peek (T.Struct, Previous, D);
  T.Values (I) := V;
  -- Check that the leaf ordering is preserved
end Insert;
```

Note that GNATprove does not support type invariants on tagged types nor on types declared in nested/child units for now. Therefore, we can neither change Search_Tree to derive from Tree nor move Binary_Trees as a nested or child package of Search_Trees.

References


National Ada Organizations

Ada-Belgium
attn. Dirk Craeynest
c/o KU Leuven
Dept. of Computer Science
Celestijnenlaan 200-A
B-3001 Leuven (Heverlee)
Belgium
Email: Dirk.Craeynest@cs.kuleuven.be
URL: www.cs.kuleuven.be/~dirk/ada-belgium

Ada in Denmark
attn. Jørgen Bundgaard
Email: Info@Ada-DK.org
URL: Ada-DK.org

Ada-Deutschland
Dr. Hubert B. Keller
Karlsruher Institut für Technologie (KIT)
Institut für Angewandte Informatik (IAI)
Campus Nord, Gebäude 445, Raum 243
Postfach 3640
76021 Karlsruhe
Germany
Email: Hubert.Keller@kit.edu
URL: ada-deutschland.de

Ada-France
attn: J-P Rosen
115, avenue du Maine
75014 Paris
France
URL: www.ada-france.org

Ada-Spain
attn. Sergio Sáez
DISCA-ETSINF-Edificio 1G
Universitat Politècnica de València
Camino de Vera s/n
E46022 Valencia
Spain
Phone: +34-963-877-007, Ext. 75741
Email: ssaez@disca.upv.es
URL: www.adaspain.org

Ada-Switzerland
c/o Ahlan Marriott
Altweg 5
8450 Andelfingen
Switzerland
Phone: +41 52 624 2939
e-mail: president@ada-switzerland.ch
URL: www.ada-switzerland.ch