The journal for the international Ada community

Ada User Journal

Volume 41
Number 4
December 2020

Produced by Ada-Europe

Information on Subscriptions and Advertisements

Ada User Journal (ISSN 1381-6551) is published in one volume of four issues. The Journal is provided free of charge to members of Ada-Europe. Library subscription details can be obtained direct from the Ada-Europe General Secretary (contact details above). Claims for missing issues will be honoured free of charge, if made within three months of the publication date for the issues. Mail order, subscription information and enquiries to the Ada-Europe General Secretary.

For details of advertisement rates please contact the Ada-Europe General Secretary (contact details above).

ADA USER JOURNAL

Volume 41
Number 4
December 2020

Contents

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

Ada User Journal aims to inform readers of developments in the Ada programming language and its use, general Ada-related software engineering issues and Ada-related activities. The language of the journal is English. 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

This is the last issue of 2020, a year during which we had to work and live in a very different manner, facing several challenges no one would have expected one year ago. The preparation of the Ada User Journal was significantly affected throughout the year and is now getting back to normal. The journal is again being printed and past issues will be steadily arriving to our subscribers during the next few months. Nevertheless, this issue is still being made available on a digital form prior to being printed and sent out, in a few weeks from now.

Concerning the contents of this issue, we firstly provide a special contribution prepared by Pat Rogers, from AdaCore, which illustrates how to program in SPARK to reach a Platinum level implementation, starting from basic Ada. The article considers a sequential, bounded stack abstract data type as a case study, to show the evolution of the code through the several levels of software assurance.

Then we start the publication of the Proceedings of the HILT 2020 Workshop on Safe Languages and Technologies for Structured and Efficient Parallel and Distributed/Cloud Computing, including two papers in this issue. The first one, entitled "A Layered Mapping of Ada 202X to OpenMP", is authored by S. Tucker Taft (also from AdaCore). The paper describes some lightweight parallelism features to be included in Ada 202X, somehow making use of OpenMP lightweight thread scheduling capabilities. The second paper, by Jan Verschelde from the University of Illinois at Chicago, USA, is entitled "Parallel Software to Offset the Cost of Higher Precision". The author explains how to exploit parallelism to compensate for the extra cost of multiple precision representations and operations, which are necessary to reach sufficiently precise solutions for some scientific problems. The presented algorithms have been developed using the Ada language.

The Quarterly News Digest and the Calendar sections, respectively prepared by Alejandro R. Mosteo and Dirk Craeynest, are included as usual. It is worth noticing that the 25th Ada-Europe International Conference on Reliable Software Technologies (AEiC 2021), is planned to take place online, from the 7th to the 11th of June 2021. The preparations are on-going to make this event a memorable one, despite taking place as a virtual-only event. All the news can be obtained on http://www.ada-europe.org/conference2021.

We are happy to bring you another interesting puzzle prepared by John Barnes, this time about "Shrinking Squares and Colourful Cubes". The solution for the nested squares puzzle that we included in the previous issue is also provided.

Last but not the least, we sadly inform that William (Bill) Bail passed away suddenly at his home on December 7, 2020. Bill Bail was actively participating in Ada-Europe conferences for many years, namely by giving tutorials on several topics, always very appreciated by the attendees. The issue closes with an In Memoriam to Bill, written by Dirk Craeynest, in representation of the Ada-Europe Board.

Antonio Casimiro
Lisboa
December 2020
Email: AUJ_Editor@Ada-Europe.org

Join Ada-Europe!

Become a member of Ada-Europe and **support Ada-related activities** and the future **development of the Ada programming language**.

Membership benefits include **receiving the quarterly Ada User Journal** and a substantial **discount when registering for the annual Ada-Europe conference**.

To apply for membership, visit our web page at

http://www.ada-europe.org/**join**

Quarterly News Digest

Alejandro R. Mosteo

Centro Universitario de la Defensa de Zaragoza, 50090, Zaragoza, Spain; Instituto de Investigación en Ingeniería de Aragón, Mariano Esquillor s/n, 50018, Zaragoza, Spain; email: amosteo@unizar.es

Contents

[Messages without subject/newsgroups are replies from the same thread. Messages may have been edited for minor proofreading fixes. Quotations are trimmed where deemed too broad. Sender's signatures are omitted as a general rule. —arm]

Preface by the News Editor

Dear Reader,

As I write these lines I have the FOSDEM livestream on my second monitor. This brings me to the first topic that I want to highlight in this issue: sadly, during last quarter we knew [1] of the passing of fellow Adaist Vinzent "Jellix" Höfler. I "devirtualized" him precisely at FOSDEM'20, where he cracked a joke during my demo that was producing lots of "No C sources found in this project" warnings. To this, he had to say (filtered by my memory): "I don't see the problem."

As for regular discussions, this time around I selected a few interesting and sometimes amusing heated debates. We have a couple of technical rabbit holes, about the finer details of protected actions syntax (that started from an innocent-looking question about logging [2]) and properties of real-time clocks and durations [3]. Did you know that Duration'Range can legally be as short as a day? I am a bit ashamed to admit I did not. Also, an often-seen observation about array indexing syntax from an Ada newcomer led to many strongly-held opinions on the merits (or lack thereof) of some aspects of Ada syntax [4] that led us as far as when Ada prototypes had parentheses for subprograms without arguments.

To conclude, during this period also took place the Advent of Code, a scored competition in which a programming puzzle a day is presented for you to solve in your favorite language. A few members of c.l.a. took the bait and this led to some interesting exchanges of ideas around the solutions in a large number of threads which I have strived to summarize for you [5].

Sincerely,
Alejandro R. Mosteo.

[1] "Tragic News about Vinzent Hoefler", in Obituary.

[2] "Logging and Protected Actions", in Ada Practice.

[3] "Starting time of Real-time Clock", in Ada Practice.

[4] "Ada Syntax Questions", in Ada Practice.

[5] "Advent of Code" and "Advent of Code Thread Compilation", in Ada Practice.

Ada-related Events

ACM HILT 2020 at SPLASH 2020

[Event already in the past, for the record. —arm]

From: Richard Wai <ric.wai88@gmail.com>
Subject: ACM HILT 2020 (High Integrity Language Technologies) at SPLASH 2020 - Nov 16 & 17
Date: Sun, 1 Nov 2020 19:56:24 -0800
Newsgroups: comp.lang.ada

Hey everyone, just a reminder that the 6th HILT workshop this year is on Nov 16 & 17, and is part of the larger SPLASH 2020 conference (2020.spashcon.org). Unsurprisingly, this year's workshop will be fully virtual.

HILT 2020 focuses on the growing importance of large-scale, highly parallel, distributed and/or cloud applications.

For Ada specifically, we have talks on:

- A layered mapping of Ada 202X parallel constructs to OpenMP (Tucker Taft),
- Experience integrating FAA's NextGen ERAM (mostly Ada) with SWIM (Mixed languages) (Brian Kleinke, Leidos)
- A highly parallel multiple double precision polynomial solver framework in Ada (PHC Pack - Prof. Jan Verschelde of UoI at Chicago)
- A cloud-native/HPC-centric hyperscaling framework for Ada, and a supporting Ada-specific exokernel OS (Yours truly)

Please check out the workshop's website (https://2020.splashcon.org/home/hilt-2020) if you are interested in attending.

CfC 25th Ada-Europe Conf. on Reliable Software Technologies

From: Dirk Craeynest <dirk@orka.cs.kuleuven.be>
Subject: CfC 25th Ada-Europe Conf. on Reliable Software Technologies
Date: Sun, 6 Dec 2020 11:39:55 -0000
Newsgroups: comp.lang.ada, fr.comp.lang.ada,comp.lang.misc

[CfC is included in the Forthcoming Events Section —arm]

Ada-Europe 2021 Conference - Extended 14 January Deadline

From: Dirk Craeynest <dirk@orka.cs.kuleuven.be>
Subject: Ada-Europe 2021 Conference - EXTENDED 14 January deadline
Date: Thu, 31 Dec 2020 15:54:46 -0000
Newsgroups: comp.lang.ada, fr.comp.lang.ada,comp.lang.misc

The Ada-Europe 2021 Conference organizers decided to provide more time for authors to prepare their contributions. The deadline for most submissions is extended to Thursday 14 January 2020. 2 weeks remain!

[CfC is included in the Forthcoming Events Section —arm]

Happy Birthday, Lady Ada

From: AdaMagica <christ-usch.grein@t-online.de>
Subject: Happy birthday, Lady Ada
Date: Wed, 9 Dec 2020 19:00:53 -0800
Newsgroups: comp.lang.ada

Primeval times when Babbage dwelt:

not were bit nor byte

nor operating system,

not hardware below

nor above software,

abyss abundant,

but computer nowhere.

And lo, there was Ada,

and Ada separated the numbers

and split them,

in Zero and One did she split them.

Continuation see: https://www.ada-deutschland.de/sites/default/files/AdaTourCD/AdaTourCD2004/Ada%20Magica/20.html

From: Simon Wright <simon@pushface.org>
Date: Thu, 10 Dec 2020 10:08:56 +0000

> in Zero and One did she split them.

The Analytical Engine was a decimal machine

From: AdaMagica <christ-usch.grein@t-online.de>
Date: Thu, 10 Dec 2020 02:52:06 -0800

> The Analytical Engine was a decimal machine

That's OK.

I know Babbage's engine came before Zuse, C++ came after Ada.

But an ode need not be historically correct. Would you claim Edda is historically correct?

Ár var alda, þat er Ymir bygði,
Vara sandr né sær né svalar unnir;
iorð fannz æva né upphiminn,
gap var ginnunga, enn gras hvergi.

Translate this and it will give about the same as the first verse above.

Ada and Education

Strategies for Teaching Ada

[Cont. from "Strategies for Teaching Ada" in AUJ 41-2, June 2020 —arm]

From: Norman Worth <nworth@comcastnospam.net>
Subject: Re: Beginning Ada Programming, by Andrew T. Shvets (2020)
Date: Mon, 2 Nov 2020 14:14:03 -0700
Newsgroups: comp.lang.ada

>> There's nothing wrong with using integer to start off and then moving onto defined types.

> Yes there is! (see my paper at the last Ada-Europe). The first message when you teach Ada is that it is all about defining proper types. You have to start by fighting bad habits from other languages.

One of the most difficult things for programmers to graft these days is the concept and proper use of types, which is key to Ada. Ada makes this even more complicated with the very useful attributes of private and limited types. Unless a text clearly conveys the use of types and illustrates it throughout, it is useless for teaching people Ada. Since this is a foreign concept to most current programmers, illustrations and good exercises are needed, too.

Compare this text to Barnes, which most of us use as a quick reference.

From: Shark8 <onewingedshark@gmail.com>
Date: Thu, 12 Nov 2020 13:24:57 -0800

> So I can't learn Ada from docs online?

You can. But the best Ada resources are books and the Language Reference.

(The Language Reference is dry, but very readable compared to some of the other standards I've come across.)

Also, the compiler itself is typically very good because of generally high-quality error messages.

From: Chris Townley <news@cct-net.co.uk>
Date: Thu, 12 Nov 2020 22:31:59 +0000

> Also, the compiler itself is typically very good because of generally high-quality error messages.

Although the errors can be very confusing sometimes, if you make a big mistake…

Ada-related Resources

[Delta counts are from Nov 2nd to Feb 2nd. —arm]

Ada on Social Media

From: Alejandro R. Mosteo <amosteo@unizar.es>
Subject: Ada on Social Media
Date: Tue, 02 Feb 2021 17:31:21 +0100
To: Ada User Journal readership

Ada groups on various social media:

- LinkedIn: 3_078 (+53) members [1]
- Reddit: 5_233 (+513) members [2]
- Stack Overflow:1_973 (+49) questions [3]
- Freenode: 85 (-5) users [4]
- Gitter: 66 (+2) people [5]
- Telegram: 108 (+18) users [6]
- Twitter: 60 (-7) tweeters [7]
  95 (+3) unique tweets [7]

[1] https://www.linkedin.com/groups/114211/

[2] http://www.reddit.com/r/ada/

[3] http://stackoverflow.com/questions/tagged/ada

[4] https://netsplit.de/channels/details.php?room=%23ada&net=freenode

[5] https://gitter.im/ada-lang

[6] https://t.me/ada_lang

[7] http://bit.ly/adalang-twitter

Repositories of Open Source Software

From: Alejandro R. Mosteo <amosteo@unizar.es>
Subject: Repositories of Open Source software
Date: Mon, 02 Nov 2020 18:41:21 +0100
To: Ada User Journal readership

Rosetta Code: 761 (+14) examples [1]
37 (=) developers [2]

GitHub: 755 (+26) developers [3]

Sourceforge: 278 (+2) projects [4]

Open Hub: 212 (=) projects [5]

Alire: 146 (+16) crates [6]

Bitbucket: 88 (-2) repositories [7]

Codelabs: 52 (=) repositories [8]

AdaForge: 8 (=) repositories [9]

[1] http://rosettacode.org/wiki/Category:Ada

[2] http://rosettacode.org/wiki/Category:Ada_User

[3] https://github.com/search?q=language%3AAda&type=Users

[4] https://sourceforge.net/directory/language:ada/

[5] https://www.openhub.net/tags?names=ada

[6] https://alire.ada.dev/crates.html

[7] https://bitbucket.org/repo/all?name=ada&language=ada

[8] https://git.codelabs.ch/?a=project_index

[9] http://forge.ada-ru.org/adaforge

Language Popularity Rankings

From: Alejandro R. Mosteo <amosteo@unizar.es>
Subject: Ada in language popularity rankings
Date: Mon, 20 Jul 2020 09:38:21 +0100
To: Ada User Journal readership

[Positive ranking changes mean to go up in the ranking. The IEEE ranking has published its 2020 edition. —arm]

- TIOBE Index: 32 (+7) 0.4% (+0.05%) [1]
- PYPL Index: 19 (=) 0.65% (+0.3%)[2]
- IEEE Spectrum (general): 39 43 (+4) Score: 32.8 24.8 [3]
- IEEE Spectrum (embedded): 12 13 (+1) Score: 32.8 24.8 [3]

[1] https://www.tiobe.com/tiobe-index/

[2] http://pypl.github.io/PYPL.html

[3] https://spectrum.ieee.org/static/interactive-the-top-programming-languages-2020

Ada-related Tools

Zip-Ada v.57

From: gautier_niouzes@hotmail.com
Subject: Ann: Zip-Ada v.57
Date: Fri, 2 Oct 2020 09:57:42 -0700
Newsgroups: comp.lang.ada

New in v.57 [rev. 799]:

- UnZip: fixed bad decoding case for the Shrink (LZW) format, on some data compressed only by PKZIP up to v.1.10, release date 1990-03-15.
- Zip.Create: added Zip_Entry_Stream_Type for doing output streaming into Zip archives.
- Zip.Compress: Preselection method detects Audacity files (.aup, .au) and compresses them better.

Zip-Ada is a pure Ada library for dealing with the Zip compressed archive file format. It supplies:

- compression with the following sub-formats ("methods"): Store, Reduce, Shrink (LZW), Deflate and LZMA
- decompression for the following sub-formats ("methods"): Store, Reduce, Shrink (LZW), Implode, Deflate, Deflate64, BZip2 and LZMA
- encryption and decryption (portable Zip 2.0 encryption scheme)
- unconditional portability - within limits of compiler's provided integer types and target architecture capacity
- input archive to decompress can be any kind of indexed data stream
- output archive to build can be any kind of indexed data stream
- input data to compress can be any kind of data stream
- output data to extract can be any kind of data stream
- cross format compatibility with the most various tools and file formats based on the Zip format: 7-zip, Info-Zip's Zip, WinZip, PKZip, Java's JARs, OpenDocument files, MS Office 2007+, Google Chrome extensions, Mozilla extensions, E-Pub documents and many others
- task safety: this library can be used ad libitum in parallel processing
- endian-neutral I/O

Main site & contact info:

http://unzip-ada.sf.net

Project site & subversion repository:

https://sf.net/projects/unzip-ada/

GitHub clone with git repository:

https://github.com/zertovitch/zip-ada

GNAT CE 2020, arm-eabi, for macOS

From: Simon Wright <simon@pushface.org>
Subject: GNAT CE 2020, arm-eabi, for macOS
Date: Tue, 06 Oct 2020 16:59:05 +0100
Newsgroups: comp.lang.ada

There were few downloads of this from the AdaCore site, so they've not produced a 2020 edition. This is my attempt at that missing edition!

At https://sourceforge.net/projects/gnuada/files/GNAT_GPL%20Mac%20OS%20X/2020-arm-eabi-darwin-bin/

Simple Components v4.51

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Subject: ANN: Simple Components for Ada v4.51
Date: Sun, 18 Oct 2020 08:43:41 +0200
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, streams, multiple connections server/client designing tools and protocols implementations.

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

Changes to the previous version:

- Memory leak fixed in the package Generic_Unbounded_Ptr_Array;
- Bug fix in data selector initialization (in the package GNAT.Sockets.Connection_State_Machine);
- An exception-free variant of Put was added to the Generic_Indefinite_FIFO package;
- ModBus TCP/IP implementation bug fix (the package GNAT.Sockets.Connection_State_Machine.MODBUS_Client);
- Code modified to work around GCC 10.0.1 optimization bug.

Simple Components v4.53

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Subject: ANN: Simple components v4.53
Date: Sun, 13 Dec 2020 10:02:03 +0100
Newsgroups: comp.lang.ada

[...]

Changes to the previous version:

- Get_Reader_Timeout, Set_Reader_Timeout, Wait_For_Tasks were added to the package GNAT.Sockets.Server.Blocking;
- JSON parser fixed to accept empty objects {} and empty array [];
- OpenSSL bindings were extended;
- The procedure Next_Unbiased was added to the package Generic_Random_Sequence.

Ahven 2.8

From: Tero Koskinen <tero.koskinen@iki.fi>
Subject: ANN: Ahven 2.8
Date: Sun, 18 Oct 2020 21:47:38 +0300
Newsgroups: comp.lang.ada

Ahven version 2.8 is now available from https://www.ahven-framework.com/

Direct links to tar.gz and zip packages:

* https://www.ahven-framework.com/releases/ahven-2.8.tar.gz
* https://www.ahven-framework.com/releases/ahven-2.8.zip

Ahven is a simple unit test library (or a framework) for Ada programming language. It is loosely modelled after JUnit and some ideas are taken from AUnit. Ahven is free software distributed under permissive ISC license and should work with any Ada 95, 2005, or 2012 compiler.

Version 2.8 is a minor maintenance release. The changes are:

* Source code repository of Ahven is now hosted at

  Sourcehut: https://hg.sr.ht/~tkoskine/ahven
* Improvements to Janus/Ada build scripts
* Improvements to GNAT build scripts
* Minor documentation updates

HAC v.0.076

From: gautier_niouzes@hotmail.com
Subject: Ann: HAC v.0.076
Date: Sat, 24 Oct 2020 00:38:57 -0700
Newsgroups: comp.lang.ada

HAC (HAC Ada Compiler) is a small, quick, open-source Ada compiler, covering a subset of the Ada language.

You find below the changes since the last post about HAC in this forum.

Links to the project and contact (tracing ;-)) addresses are available from the blog posts cited below.

0.071 Discrete type range is stored in type definition; "subtype T1 is T2;"

0.072 Subtype_Indication (e.g. "for B in Boolean loop", "array (States) of Prob")

https://gautiersblog.blogspot.com/2020/06/hac-v0072-subtype-indication.html

0.073 The VM can be aborted via the Feedback procedure

0.074 Types: Time and Duration

0.075 Added Ada.Calendar-like functions

https://gautiersblog.blogspot.com/2020/10/hac-v0075-time-functions-goodies-for.html

0.076 Added Ada.Directories-like subprograms

https://gautiersblog.blogspot.com/2020/10/hac-v0076-adadirectories-like.html

XNAdaLib 2020 Binaries for macOS Catalina

From: Blady <p.p11@orange.fr>
Subject: [ANN] XNAdaLib 2020 binaries for macOS Catalina including GTKAda and more.
Date: Sun, 25 Oct 2020 10:11:49 +0100
Newsgroups: comp.lang.ada

This is XNAdaLib 2020 built on macOS 10.15 Catalina for Native Quartz with GNAT Community 2020 including:

- GTKAda 20.2 (www.adacore.com/gtkada) with GTK+ 3.24.20 (www.gtk.org) complete,
- Glade 3.22.1 (glade.gnome.org),
- GnatColl 20.2 (github.com/AdaCore/gnatcoll),
- Florist mid-2020a (github.com/Blady-Com/florist),
- AdaCurses 6.2 (invisible-island.net/ncurses/ncurses-Ada95.html),
- Gate3 0.5c (sourceforge.net/projects/lorenz),
- Components 4.50 (www.dmitry-kazakov.de/ada/components.htm),
- AICWL 3.24 (www.dmitry-kazakov.de/ada/aicwl.htm),
- Zanyblue 1.4.0 (zanyblue.sourceforge.net),
- PragmARC mid-2020 (pragmada.x10hosting.com/pragmarc.htm),
- GNOGA 1.6-beta (www.gnoga.com),
- SparForte 2.3.1 (sparforte.com),
- Alire 0.6.1 (alire.ada.dev), NEW

and as side libraries:

- Template Parser 20.2,
- gtksourceview 3.24.4,
- GNUTLS 3.6.14,
- GMP 6.1.2,
- make 4.2.1,
- Python 2.7.17.

XNAdaLib binaries have been post on Source Forge:

https://sourceforge.net/projects/gnuada/files/GNAT_GPL%20Mac%20OS%20X/2020-catalina

Report preferably all comments to MacAda.org mailing list:

http://macada.org/macada/Contacts.html

See list archive:

https://hermes.gwu.edu/archives/gnat-osx.html

From: Simon Wright <simon@pushface.org>
Date: Sun, 25 Oct 2020 09:39:41 +0000

Great stuff, just a couple of comments -

> - Python 2.7.17.

Not maintained since 1 Jan. There are excellent downloads of 3 (currently 3.9) at python.org.

> Report preferably all comments to MacAda.org mailing list:

> http://macada.org/macada/Contacts.html

Gives a (Korean?) 404.

You can subscribe at https://hermes.gwu.edu/cgi-bin/wa?A0=GNAT-OSX

RFC UXStrings Package.

From: Blady <p.p11@orange.fr>
Subject: RFC UXStrings package.
Date: Wed, 11 Nov 2020 21:18:17 +0100
Newsgroups: comp.lang.ada

UXStrings is now a standalone library available on Github.

https://github.com/Blady-Com/UXStrings

Comments on specifications are welcome.

A first implementation POC is provided. UTF-8 encoding is chosen for internal representation. The Strings_Edit library is used for UTF-8 encoding management.

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

This implementation which is only to demonstrate the possible usages of UXString has many limitations as for instance there is no memory deallocation. Only a few API are implemented.

A test program is also provided with some features working.

See readme for full details.

https://github.com/Blady-Com/UXStrings/blob/master/readme.md

From: Vadim Godunko <vgodunko@gmail.com>
Date: Fri, 27 Nov 2020 00:38:56 -0800

There are few more options to forget about encodings and related issues:

New AdaCore's VSS
https://github.com/AdaCore/VSS

Old Matreshka's League
http://forge.ada-ru.org/matreshka

From: Luke A. Guest <laguest@archeia.com>
Date: Fri, 27 Nov 2020 11:05:08 +0000
> There are few more options to forget about encodings and related issues:

My very basic utf-8 string - https://github.com/Lucretia/uca

Ada-12 Version of PragmARC

From: PragmAda Software Engineering <pragmada@pragmada.x10hosting.com>
Subject: [Ann] Ada-12 Version of the PragmAda Reusable Components
Date: Sun, 1 Nov 2020 19:20:42 +0100
Newsgroups: comp.lang.ada

Now that there are 2 (count 'em!) Ada-12 compilers*, an Ada-12 version of the PragmARCs is available at https://github.com/jrcarter/PragmARC

In addition to making use of Ada-12 features, this version has a restructured package hierarchy and is released under the 3-clause BSD license.

These have only been compiled with the GNAT compiler. Feedback from those with access to the other compiler would be welcome.

*Defined as a compiler that implements the entire Ada-12 core language.

SweetAda 0.1g

From: Gabriele Galeotti <gabriele.galeotti.xyz@gmail.com>
Subject: SweetAda 0.1g released
Date: Sun, 15 Nov 2020 13:16:55 -0800
Newsgroups: comp.lang.ada

I've just released SweetAda 0.1g.

This is a maintenance release, and introduces new toolchains based on Binutils 2.35, GCC 10.2.0 and GDB 10.1.

Along with new tools, the basic support libraries, e.g., GMP, MPFR, MPC, and all auxiliary libraries were used at the highest stable version during the builds.

Sorry for a significant delay in releasing, but it is very time-consuming to keep everything in-sync, especially when toolchains change. Neither I had the time to complete the manual, I'll try to do that in the near future.

SweetAda itself gets few changes:

- due to a deeper Ada code analysis, the new compiler front-end showed possible superfluous aspects; they are removed and warnings made silent
- slightly better menu scripts
- echo_log() and echo_log_error() functions in Bash scripts are now renamed as log_print() and log_print_error()
- minor changes and typos here and there

Of course, LibGCC and RTS packages are synchronized with new toolchains, so download them as well.

I am working on Insight too, hopefully packages will be available ASAP, but it is still at 20200417 timestamp. Please note that if you install Insight, it will overwrite the standard GDB executable, and you're stuck at 9.1. GPRbuild remains at 20200417 timestamp as well.

I discovered a mismatch in QEMU for Linux 20200817 targeted for ARM, AVR, AArch64, x86 and M68k CPUs, where executables end up being objects for an OS X platform, because of bad naming. This is now corrected. Sorry for that, please re-download the following packages:

qemu-aarch64-20200817L.tar.xz

qemu-arm-20200817L.tar.xz

qemu-avr-20200817L.tar.xz

qemu-i386-20200817L.tar.xz

qemu-m68k-20200817L.tar.xz

Furthermore, QEMU for Windows packages lack libffi-6.dll. This is now corrected. Please re-download

qemu-<every_cpu>-20200817W.zip (or place a libffi-6.dll library taken from a random MinGW64 package, along the QEMU executable).

Find everything at https://www.sweetada.org.

By the way, the connection to SweetAda website is now completely secure. Many thanks to the Certbot team.

From: Keith Thompson <keith.s.thompson+u@gmail.com>
Date: Mon, 16 Nov 2020 12:51:39 -0800

I suggest that an announcement like this should include, at or near the top of the article, a brief description of what SweetAda is.

From the web site:

SweetAda is a lightweight development framework whose purpose is the implementation of Ada-based software systems.

[...]

AdaStudio-2021 Release 01/01/2021 Free Edition

From: Leonid Dulman <leonid.dulman@gmail.com>
Subject: Announce : AdaStudio-2021 release 01/01/2021 free edition
Date: Wed, 30 Dec 2020 00:51:09 -0800
Newsgroups: comp.lang.ada

I'm pleased to announce AdaStudio-2021.

In the new AdaStudio release it was added Qt6Ada support for new framework Qt-6.0.0.

I added some packages from Qt-5.15.0 open source (qtcharts qtconnectivity qtgraphicaleffects qtimageformats qttexttospeech qtlocation qtlottie qtmultimedia qtsensors qtserialbus qtserialport qtwebchannel)

Qt6ada version 6.0.0 open source and qt6base.dll, qt6ext.dll (win64), libqt6base.so, libqt6txt.so(x86-64) built with Microsoft Visual Studio 2019 x64bin Windows, gcc x86-64 in Linux.

Package tested with GNAT gpl 2020 Ada compiler in Windows 64bit, Linux x86-64 Debian 10.0

I built Qt6 binaries for win64 and x86-64 and include them into AdaStudio-2021 (qt6ada directory)

Known problems:

1) for quick3d and quickcontrols2 plugins I have got unresolved entry points ml_registr_types_QtQuick3D(), so some examples do not work properly.
2) in Linux multimedia plugins do not built properly and services do not work (qtavada works fine)
3) webengine does not work and it is not added to qt6ada

Qt 6 is a new long time project and I hope to solve these problems in the next release.

Qt6Ada is built under a GNU GPLv3 license: https://www.gnu.org/licenses/lgpl-3.0.html.

Qt6Ada and VTKAda for Windows, Linux (Unix) is available from

https://r3fowwcolhrzycn2yzlzzw-on.drv.tw/AdaStudio

web page or Google drive

https://drive.google.com/folderview?id=0B2QuZLoe-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 "Qt6 classes to Qt6Ada packages relation table.docx"

The latest hacker attacks will force many companies to reconsider technologies based on scripting languages such as Python, Ruby, Perl, JavaScript and others, in which it is much easier to replace code than in translated modules. Therefore, interest in a language such as Ada should greatly increase.

If you have any problems or questions, let me know.

Ada-related Products

Adalog's "Back to Quality" Program

From: J-P. Rosen <rosen@adalog.fr>
Subject: [Ann] Adalog's "Back to Quality" program
Date: Sat, 24 Oct 2020 09:03:33 +0200
Newsgroups: comp.lang.ada

Adalog annonces the "Back to quality" program.

Thanks to our experience and advanced tools, we offer technical assistance to relieve your technical dept by fixing non-conformities to your coding standard that you never have time to fix by yourself.

For more details, see:
https://adalog.fr/en/btq_program.html
or write to info@adalog.fr

State Preserving Fault Tolerance for Ada Applications

From: Thomas Wetmore <tom.wetmore@gmail.com>
Subject: State Preserving Fault Tolerance for Ada Applications
Date: Wed, 9 Dec 2020 13:37:51 -0800
Newsgroups: comp.lang.ada

Our small startup has developed a new software fault tolerant (FT) architecture, implemented as an SDK and library, that we are currently adapting for use with Ada and SPARK. It will enable developers to create true state preserving, fault tolerant Ada applications by either developing new or modifying existing code. The architecture provides additional levels of availability and security by providing resilience against both

hardware failures and software anomalies (attacks). The port will enable Ada users to create FT Ada applications that can be adapted for most COTS h/w - s/w platforms. Such applications can even be run on heterogeneous, geographically distributed configurations - using bare metal, virtual machines, or containers.

Note that this new application-based FT software technology was created by our veteran computer design engineers who have developed multiple generations of fault tolerant systems currently in world-wide use. The Ada implementation of the technology is being created by a veteran Ada expert who has been developing with Ada since its inception.

We are looking for users with whom we can collaborate to 1) provide needs input, 2) assist with QC & real-world use case testing, and/or 3) create prototypes and/or proofs of concept. Please let me know if you are interested in learning more and we will be glad to share additional information.

Ada and Operating Systems

Developing on a Mac

From: Marius Amado-Alves <amado.alves@gmail.com>
Subject: Developing on a Mac
Date: Wed, 14 Oct 2020 09:39:58 -0700
Newsgroups: comp.lang.ada

I searched but could not find it. How to develop Ada programs on a Mac today (Catalina)? GNAT CE 2020 for Mac has no GPS anymore. Must one use Xcode? How to make Xcode Ada-aware and integrate it with GNAT? Some other Ada-aware IDE for Mac?

From: Simon Wright <simon@pushface.org>
Date: Wed, 14 Oct 2020 20:02:59 +0100

> How to develop Ada programs on a Mac today (Catalina)? GNAT CE 2020 for Mac has no GPS anymore.

If you want GPS the best bet is probably to use the GPS from GNAT CE 2019 with the new compiler. Have CE 2020 bin first on your PATH, then explicitly call up gps: I just used /opt/gnat-ce-2019/bin/gps.

There is a port of GNAT Studio to Catalina[1], but ISTR it's not all working 100%?

> Must one use Xcode? How to make Xcode Ada-aware and integrate it with GNAT?

Last time I heard, Xcode is proprietary and closed, and no one has ever reported extending it for Ada. But of course I haven't been looking.

> Some other Ada-aware IDE for Mac?

Emacs[2], with ada-mode[3]].

[1] https://sourceforge.net/projects/gnuada/files/GNAT_GPL%20Mac%20OS%20X/2020-catalina/GNATStudio-20.2-a.dmg/download

[2] https://emacsformacosx.com

[3] https://www.nongnu.org/ada-mode/ada-mode.html

From: Simon Wright <simon@pushface.org>
Date: Thu, 15 Oct 2020 10:35:39 +0100

> How to develop Ada programs on a Mac today (Catalina)?

[...]

> Some other Ada-aware IDE for Mac?

Just announced:

https://github.com/thindil/vim-ada/releases/tag/v10.0

https://github.com/thindil/Ada-Bundle

From: Jerry <list_email@icloud.com>
Date: Thu, 15 Oct 2020 16:41:40 -0700

> How to develop Ada programs on a Mac today (Catalina)?

Some of the following is kind of vague but I hope it is useful. Many listers will know much more.

One time a long time ago someone (on this list?) made Xcode work with Ada. It was fantastic. Even a debugger IIRC. But apparently Apple likes to change the underpinnings and after some time Xcode ceased to work with Ada. (There also is or was a FPC Pascal way with Xcode that was even more capable but I haven't checked into that for a long time.)

There also used to be Carbon bindings to Ada, possibly made by the same person. (The words "Blady" and "Pascal" come to mind for this person.) They were on the macada.org web site which doesn't seem to do much these days, as well as being linked from AdaPower. Of course the Carbon API has been long-deprecated but I'm sure it is still used. (How does Microsoft keep Word et al working on Macs?)

It's not a full IDE in some opinions but Visual Studio Code runs on Macs, even my now-ancient 2008 PowerBook and macOS 10.11.6. There is an Ada plug-in but make sure you get the right one. I think this plug-in might be supported by AdaCore. And there's something about an Ada Language Server. I don't really understand all of this. I've tried to get this running but the instructions are minimal so it is taking more effort than it should. (Why are installation instructions so frequently written assuming that you already know how to install stuff?)

IntelliJ IDEA CE also has an Ada plug-in. I guess Eclipse has an Ada plug-in as well. I think AdaCore supports this but I'm not sure if the Mac version is well-supported.

None of the above except Xcode is a native Mac app so you'll have to deal with a certain amount of cross-platform-turdism. I would happily pay hundreds of $US for a native Mac Ada IDE but that will never happen. The previously-mentioned Xcode hack was close enough, though.

There are lots of text editors that aren't too bad. I have used Textmate with its Ada plug-in (bundle) which I've modified for my own purposes for many years. Not an IDE but it does have the capability to link from parsed error reports back to your code. Textmate was a leader in this area and its bundle architecture has been used by several other editors.

Sorry if this is all a little sketchy.

Now for something OT. If you are doing technical work in Ada and want to store or examine or plot results, I have made Igor Pro (wavemetrics.com) work with Ada. This is a fantastic arrangement. It's almost as nimble as working in a notebook (think Jupyter or Jupyter Lab) but you get the awesomeness of Igor Pro to plot, post-process, and document.

From: Blady <p.p11@orange.fr>
Date: Fri, 16 Oct 2020 22:21:30 +0200

> There also used to be Carbon bindings to Ada, possibly made by the same person.

If I remember well, the Carbon bindings were provided by James E. Hopper from a Pascal to Ada translation with p2ada of Apple Carbon API in Pascal. Though Carbon may still work, Apple wasn't maintaining the Pascal API, but only the C API.

Thus Ada Carbon Bindings weren't used anymore as far as I know. I provided some Xcode support to Ada but after, as you said, Xcode was no more customizable.

You'll find here some historical material:

https://blady.pagesperso-orange.fr/alpha.html

Ada on QNX

From: DrPi <314@drpi.fr>
Subject: Ada on QNX
Date: Thu, 10 Dec 2020 08:50:53 +0100
Newsgroups: comp.lang.ada

Anyone has cross-compiled Ada for QNX SDP 6.6.0 (ARM target)?

From: Quentin Ochem <qochem@gmail.com>
Date: Thu, 10 Dec 2020 08:03:48 -0800

Hi Nicolas,

FWIW, there's an AdaCore port that has been done specifically targeting QNX/ARM. If you want to discuss, feel free to drop me an e-mail (ochem@adacore.com).

From: DrPi <314@drpi.fr>
Date: Fri, 11 Dec 2020 10:49:57 +0100

Yes, I know. I've been in contact with someone from Adacore about 2 years ago. But the port is for QNX SDP 7.0.0 and later only.

It seems that there is provision for a QNX compilation in FSF GNAT. Not sure of that and not tried to go this way yet.

Read/Write Access to UNIX Character Devices

From: philip.munts@gmail.com
Subject: Read/write access to Unix character devices
Date: Sun, 20 Dec 2020 20:59:28 -0800
Newsgroups: comp.lang.ada

Lately I have been working with Unix (really Linux, FreeBSD, and OpenBSD) character devices (these happen to be USB raw HID devices, but the problem is more general than that). The way these work is that each hardware device has a character device node file in /dev/, like /dev/hidraw1. You open the file for both read and write access. Then you can send a command to the device by writing a binary blob and get a response by subsequently reading a binary blob. For what I am doing, it is important not to block on reads forever if there is no response forthcoming, so I need at least read timeouts.

So far, I have been binding the C library functions open(), close(), read(), write(), and poll() with pragma Import. That works, but I have wondered if there is some way of accomplishing the same thing more portably. The packages GNAT.Sockets and GNAT.Serial_Communicatons can be viewed as special case solutions, but I would like a general solution.

What I would really like is Ada.Sequential_IO with InOut_File and a timeout mechanism, perhaps like the select() wrapper in GNAT.Sockets.

So far I haven't found anything in the Ada. or GNAT. that supports InOut_File semantics (other than Direct_IO) let alone timeouts. Does anybody have any suggestions?

From: Randy Brukardt <randy@rrsoftware.com>
Date: Mon, 21 Dec 2020 19:23:08 -0600

I would use Stream_IO for this, but you'd need help from your implementer to get timeouts/nonblocking I/O. If they have them, they'd be some sort of Form parameter (that's what the typically ignored Form parameter is for).

Stream_IO is a lot more flexible that Sequential_IO and Direct_IO. (Some implementations implement those older Ada 83 packages in terms of Stream_IO.)

Ada and Other Languages

Importing Python Library into Ada

From: Roger Mc <rogermcm2@gmail.com>
Subject: Import Python library into an Ada package?
Date: Thu, 3 Dec 2020 23:36:13 -0800
Newsgroups: comp.lang.ada

Is it possible to import a Python library, such as graphviz, into an Ada package? So far I have only been able to find information on exporting Ada to Python.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Fri, 4 Dec 2020 11:23:21 +0100

I am not sure what you mean. Python is not a compiled language, so formally speaking a Python library is not a library and you cannot import it in the sense of linking it to your application and calling subprograms from it using certain calling conventions.

If you rather meant whether you could execute a Python script from Ada while passing parameters to it and taking results from, yes you can. If that script were a part of some Python module, yes you can load it and once loaded call (interpret) functions from the module.

P.S. Before you proceed, Python is a huge mess and interfacing it is a pain in the ... So you should consider if Graphviz is worth the effort. If you find a GTK or Qt library that is doing approximately the same, that would be a wiser choice, IMO.

From: Roger Mc <rogermcm2@gmail.com>
Date: Fri, 4 Dec 2020 03:37:53 -0800

Many thanks for your prompt response and comments Dmitry; they are well appreciated with some of the contents somewhat expected.

I think that I misused the term " Python library"; I think "Python module" is what I should have used.

In this context, in Python, is a module a script? I'll investigate this.

[...]

The project that I am embarking on is to use Ada for an on-line course in machine learning that uses Python as its teaching platform. The importing that I was contemplating concerns special machine learning Python modules used in the course.

Of course, the alternative is for me to translate the Python modules into Ada which is something I've done in the past; generally, in my opinion, yielding much better and more readable code. Again, thanks for your very helpful comments which, hopefully, have focused my mind on the way ahead.

Regarding your comment that "Python is a huge mess" and my own opinion of Python; I am mortified that Python seems to have become the standard language for teaching computer programming and, particularly, that it seems to be the choice of leading university computer science courses. It seems that the old well-established rules of quality computer program design have been completely abandoned by these institutions.

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Fri, 4 Dec 2020 15:22:46 +0200

> Is it possible to import a Python library, such as graphviz, into an Ada package?

If you mean the Graphviz tool-set, https://en.wikipedia.org/wiki/Graphviz, that seems to be written in C and to be open source. You should be able to call Graphviz functions from Ada in the same way as one calls any C code from Ada. The Python module you refer to is probably just a binding from Python to the C code in Graphviz.

If you want to use Graphviz just to draw automatically laid-out graphs, there is another way, that I have used: make the Ada program write out the graph definition as a text file in the "dot" language, and then invoke the "dot" program from Graphviz to lay out and draw the graph into some graphical format. However, it may be troublesome to make this method work interactively -- I was satisfied with non-interactive post-processing of the "dot" file generated by my Ada program.

From: gautier_niouzes@hotmail.com
Date: Fri, 4 Dec 2020 05:41:08 -0800

As a side note, there is a cool utility called DePlo (https://sites.google.com/site/depplot/ , sources here : https://launchpad.net/deplo) that creates a dependency graph of Ada units from the .ali files that GNAT produces when building a project.

This graph is in Graphviz's DOT format.

And indeed, graphviz is not specific to Python. The sources are in C, and the Web site mentions bindings to: guile, perl, python, ruby, C#, tcl .

From: Simon Wright <simon@pushface.org>
Date: Fri, 04 Dec 2020 13:55:15 +0000

> Regarding your comment that "Python is a huge mess" and my own opinion of Python; [...]

I'd certainly agree that interfacing to Python from Ada is a huge mess (specifically, unsupported hand management of garbage collection, as you have to do if invoking Python objects).

From: Björn Lundin <b.f.lundin@gmail.com>
Date: Fri, 4 Dec 2020 19:32:58 +0100

> If you want to use Graphviz just to draw automatically laid-out graphs [...]

And if you really just want to draw graphs - and can use another tool - gnuplot can be controlled by spawning it and sending commands on stdin via pipes.

From: Per Sandberg <per.s.sandberg@bahnhof.se>
Date: Fri, 4 Dec 2020 22:12:11 +0100

> Is it possible to import a Python library, such as graphviz, into an Ada package?

gnatcoll.python + a lot of binding work

From: Roger Mc <rogermcm2@gmail.com>
Date: Fri, 4 Dec 2020 13:19:10 -0800

> gnatcoll.python + a lot of binding work

I have been trying to figure out how to use gnatcoll.python. Unfortunately it doesn't seem to provide any supporting documentation.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sat, 5 Dec 2020 00:17:12 +0100

> Unfortunately it doesn't seem to provide any supporting documentation.

What about this:
https://docs.adacore.com/gnatcoll-docs/scripting.html

I have a rudimentary Python bindings independent of GNATColl, which I use to run Python scripts from Ada. They were designed to load Python dynamically, I did not want to make the application dependent on Python installed. If you want, you can use them as a template. There is no documentation, but the code using them. But as I said, better not... (:-))

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sat, 5 Dec 2020 10:38:10 +0100

> I would really appreciate seeing your "rudimentary Python bindings ".

Download sources of this:

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

The project is large. Only these packages are related to Python:

1. Py is the bindings
2. Py.Load_Python_Library is an OS-dependent part for loading Python dynamically from a DLL (Linux or Windows)
3. Py.ELV_MAX_Cube is an implementation of a Python module in Ada. I.e. calling Ada from Python.
4. MAX_Control_Page contains a task that periodically runs a Python script. I.e. calling Python from Ada.

Ada Practice

Logging and Protected Actions

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Subject: Re: is there a version of unix written in Ada
Date: Thu, 1 Oct 2020 11:28:10 +0200
Newsgroups: comp.lang.ada

[...] BTW, I still do not know how to design an Ada-conform tracing/logging facility such that you could trace/log from anywhere, protected action included, and without knowing statically which protected object is involved.

From: J-P. Rosen <rosen@adalog.fr>
Date: Thu, 1 Oct 2020 11:59:58 +0200

> BTW, I still do not know how to design an Ada-conform tracing/logging facility such that you could trace/log from anywhere [...]

Did you have a look at package Debug?

(https://www.adalog.fr/en/components#Debug)

It features, among others, a trace routine which is guaranteed to not be potentially blocking.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Thu, 1 Oct 2020 12:21:46 +0200

> It features, among others, a trace routine which is guaranteed to not be potentially blocking.

It calls a protected operation on a different protected object, yes, this is non-blocking, and I considered the same, but is this legal? Maybe I am wrong, but I have an impression that walking away to another object is not OK. Or is that limited to protected entries only?

Another issue is having two different calls: Trace and protected Trace. If one is used instead of another, you have a ticking bomb in the production code. I remember that there was a GNAT pragma to catch it, but it was a run-time check, so it just replaced one type of explosive with another.

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Thu, 1 Oct 2020 14:38:27 +0300

> It calls a protected operation on a different protected object, yes, this is non-blocking [...], but is this legal?

Yes.

If the program is using ceiling-priority-based protection, the priority of the calling object must be less or equal to the priority of the called object.

> Or is that limited to protected entries only?

An entry call is potentially blocking and therefore not allowed in a protected operation.

> Another issue is having two different calls: Trace and protected Trace. If one is used instead of another, you have a ticking bomb in the production code.

I assume that is a "feature" of the referenced Debug package, not of the basic method it uses to implement a logging facility.

I haven't looked at the Debug package, but I would have suggested a logging facility that consists of:

1. A FIFO queue of log entries implemented in a protected object of highest priority. The object has a procedure "Write_Log_Entry".
2. A task that empties the FIFO queue into a log file. The task calls an entry of the FIFO protected object to get a log entry from the queue, but executes the file-writing operations in task context, not in a protected operation.

From: J-P. Rosen <rosen@adalog.fr>
Date: Thu, 1 Oct 2020 13:48:26 +0200

> I remember that there was a GNAT pragma to catch it, but it was a run-time check

Well, just use AdaControl with the rule:

```
check Potentially_Blocking_Operations;
```

;-)

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Thu, 1 Oct 2020 14:51:54 +0200

> If the program is using ceiling-priority-based protection, the priority of the calling object must be less or equal to the priority of the called object.

My mental picture was protected procedure calls executed concurrently on different cores of a multi-core processor. Would that sort of implementation be legal?

If so, then let there be protected procedure P1 of the object O1 and P2 of O2. If P1 and P2 call to P3 of O3 that would be a problem. Ergo either wandering or concurrent protected protected calls must be illegal.

> 1. A FIFO queue of log entries implemented in a protected object of highest priority. The object has a procedure "Write_Log_Entry".

Yes, that was what I thought and what Debug.adb does. However Debug.adb allocates the body of the FIFO element in the pool. I would rather use my implementation of indefinite FIFO which does not use pools. I don't want allocators/deallocators inside protected stuff.

> 2. A task that empties the LIFO queue into a log file.

A simpler approach is to flush the queue by the first call to an unprotected variant of Trace. I believe Debug.adb does just this.

From: J-P. Rosen <rosen@adalog.fr>
Date: Thu, 1 Oct 2020 16:18:58 +0200

> My mental picture was protected procedure calls executed concurrently on different cores of a multi-core processor. Would that sort of implementation be legal?

No. Protected objects guarantee that only one task at a time can be inside (ignoring functions). Multi-cores don't come into play.

> I don't want allocators/deallocators inside protected stuff.

As surprising as it may seem, allocators/deallocators are NOT potentially blocking operations. But I understand your concerns...

> A simpler approach is to flush the queue by the first call to an unprotected variant of Trace. I believe Debug.adb does just this.

Yes. Moreover, there is a Finalize of a controlled object to make sure that no trace is lost if the program terminates without calling any (unprotected) Trace.

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Thu, 1 Oct 2020 18:38:12 +0300

> My mental picture was protected procedure calls executed concurrently on different cores of a multi-core processor. Would that sort of implementation be legal?

If the protected procedures belong to different protected objects, yes it is legal. But not if they belong to the same object, as J-P noted.

Note that the ordinary form of the ceiling-priority-locking method does not work for multi-cores, because a task executing at the ceiling priority of a protected object does not prevent the parallel execution of other tasks (on other cores) at the same or lower priority.

[...]

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Thu, 1 Oct 2020 19:37:01 +0200

[...] Now let's continue the example. What happens when the calling paths are:

O1.P1 --> O3.P3 --> O2.Q
O2.P2 --> O3.P3 --> O2.Q

Let Q1.P1 blocks Q2.P2 on an attempt to enter O3.P3:

O1.P1 --> O3.P3
O2.P2 --> blocked

Then O3.P3 calls O2.Q:

O1.P1 --> O3.P3 --> O2.Q
|
O2.P2 --> blocked V

This will either re-enter O2 or deadlock.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Thu, 1 Oct 2020 17:10:10 -0500

[...]

A task has to wait to get access to a PO. This is *not* blocking, it is not allowed to do anything else during such a period. (This is why protected operations are supposed to be fast!). It's canonically implemented with a spin-lock, but in some cases one can use lock-free algorithms instead.

For a single core, one can use ceiling locking instead (and have no waiting), but that model seems almost irrelevant on modern machines.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Thu, 1 Oct 2020 17:13:14 -0500

> [...] you have a problem when two independently running protected procedures of *different* objects call a procedure of a third object. You must serialize these calls, and that is effectively blocking.

Not really: blocking implies task scheduling (and possible preemption and priority inversion), whereas no scheduling happens on a protected call. There's just a possible wait. It's a subtle difference, admittedly, but it makes a world of difference to analysis.

From: J-P. Rosen <rosen@adalog.fr>
Date: Fri, 2 Oct 2020 07:36:07 +0200

To continue on Randy's response: mutual exclusion is not blocking. "Blocking" (as in "potentially blocking operation") means "being put on a queue", i.e. when the waiting time is potentially unbounded. The waiting time due to mutual exclusion is bounded by the execution time of the protected operation, and then can be included in the execution time of the waiting task. (In reality, it can be slightly more complicated, but the idea is that it is bounded).

[...]

In summary, the model of PO is two levels:

1) mutual exclusion, which is not "blocking"

2) for entries: queuing, which is "blocking"

Once you realize this, it should make this whole thread clearer....

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Fri, 2 Oct 2020 08:56:38 +0200

> mutual exclusion is not blocking. "Blocking" (as in "potentially blocking operation") means "being put on a queue", i.e. when the waiting time is potentially unbounded.

It would be a poor definition, because deadlock is not bounded as well. If jumping from one protected object to another is legal, we can construct a deadlock out of mutual exclusion. We also have a situation when multiple tasks executing protected procedures are awaiting their turn to enter a procedure of some object. They will continue (if not deadlocked) in some order, which is obviously a queue.

From: J-P. Rosen <rosen@adalog.fr>
Date: Fri, 2 Oct 2020 09:42:02 +0200

> It would be a poor definition, because deadlock is not bounded as well. If jumping from one protected object to another is legal, we can construct a deadlock out of mutual exclusion.

But this would necessarily involve an "external call to the same protected object", which is defined as a potentially blocking operation. Note that AdaControl is quite powerful at detecting that situation (by following the call graph).

> We also have a situation when multiple tasks executing protected procedures are awaiting their turn to enter a procedure of some object. They will continue (if not deadlocked) in some order, which is obviously a queue.

No, it can be implemented with a spin lock. It is bounded by the number of waiting tasks x service time. You don't have to wait for some unpredictable barrier.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Fri, 2 Oct 2020 22:14:49 -0500

> But this would necessarily involve an "external call to the same protected object", which is defined as a potentially blocking operation.

Note that such an operation doesn't really block, it is a deadlocking operation; Ada lumped it into "potentially blocking" in order to save some definitional overhead. (A mistake, in my view, it should simply have been defined to raise Program_Error or maybe Tasking_Error.) "Potentially blocking", in normal use, means something else.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Thu, 1 Oct 2020 17:21:01 -0500

> O2.P2 --> O3.P3 --> O2.Q

This latter path is always going to deadlock, since the second call to O2 is necessarily an external call (you're inside of O3, not O2). An external call has to get the lock for the protected object, and since the lock is already in use, that will never proceed.

[If O3 was nested in O2, then the second call to O2 could be internal. But in that case, the first path would be impossible as O1 could not see O3 to call it.]

Remember that the decision as to whether a call is internal or external is purely syntactic: if a protected object is given explicitly in the call, one needs to trigger the mutual exclusion mechanisms again. The only time one doesn't need to do that is when the call does not include the object (that is, directly from the body of an operation).

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Fri, 2 Oct 2020 08:55:56 +0200

> This latter path is always going to deadlock, since the second call to O2 is necessarily an external call

Is that implementation or requirement? The lock can be task-re-entrant.

> Remember that the decision as to whether a call is internal or external is purely syntactic: if a protected object is given explicitly in the call, one needs to trigger the mutual exclusion mechanisms again.

Even when the object in the call is statically known to be the same?

From: Randy Brukardt <randy@rrsoftware.com>
Date: Fri, 2 Oct 2020 22:09:19 -0500

> Is that implementation or requirement? The lock can be task-re-entrant.

Language requirement. An external call requires a separate mutual exclusion. If Detect_Blocking is on, then Program_Error will be raised. Otherwise, any pestilence might happen.

> Even when the object in the call is statically known to be the same?

Yes. An external call *always* gets the lock again. I believe that was made the rule to make it obvious as to what will happen based on the form of call.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sat, 3 Oct 2020 08:42:03 +0200

> Yes. An external call *always* gets the lock again. I believe that was made the rule to make it obvious as to what will happen based on the form of call.

I mean this:

```
protected body O is
  procedure P1 is
  begin
    ...
  end P1;
  procedure P2 is
  begin
    P1;   -- OK
    O.P1; -- Deadlock or Program_Error
  end P2;
end O;
```

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Sat, 3 Oct 2020 10:44:59 +0300

> I mean this:
>
> protected body O is
> procedure P1 is
> begin
> ...
> end P1;
> procedure P2 is
> begin
> P1; -- OK
> O.P1; -- Deadlock or Program_Error

That is an internal call, so no deadlock nor error.

See RM 9.5(4.e), which is this exact case.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sat, 3 Oct 2020 10:16:15 +0200

> That is an internal call, so no deadlock nor error.

I.e. it is *not* based on the syntax of the call.

Anyway the rather disappointing result is that protected procedures may deadlock (or Program_Error) in a legal program.

So my initial disinclination to jump from one protected object to another is reasonable advice. Or at least the order in which protected objects are navigated must be the same.

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Sat, 3 Oct 2020 13:44:47 +0300

> I.e. it is *not* based on the syntax of the call.

At least not on /that/ syntactical difference.

> Anyway the rather disappointing result is that protected procedures may deadlock (or Program_Error) in a legal program.

Legal programs can run into all sorts of problems, starting with use-before-elaboration.

> So my initial disinclination to jump from one protected object to another is reasonable advice.

Quite conservative advice, though.

> Or at least the order in which protected objects are navigated must be the same.

I would say that it is advisable to arrange the POs (or PO types) in a layered architecture and make inter-PO calls only from a higher-layer PO to a lower-layer PO.

GDNative Thick Binding Design

From: Michael Hardeman <mhardeman25@gmail.com>
Subject: GDNative thick binding design
Date: Thu, 15 Oct 2020 14:08:19 -0700
Newsgroups: comp.lang.ada

I'm working on a binding to the Godot game engine for Ada.

Project link here: https://github.com/MichaelAllenHardeman/gdnative_ada

Once the game engine has loaded your dynamic library it will call the function *_nativescript_init (where * is the symbol_prefix defined in the library resource config file). This function is responsible for registering objects, object methods, and allocating any memory needed.

What I want to discuss here is that I'm a bit at a loss as to how to design a thick binding wrapper around this object registration pattern. So let me describe the pattern.

I have a very simple example translated from C using the thin binding here: https://github.com/MichaelAllenHardeman/gdnative_ada/blob/master/examples/gdnative_c_example/src/simple.adb#L44

The objects must have a name, but may or may not override the constructor/destructor life cycle functions (which you pass in during registration)

There are

https://docs.godotengine.org/en/stable/classes/class_object.html#class-object

There is kind of a hierarchy at play as well:

the Node type extends Object

https://docs.godotengine.org/en/stable/classes/class_node.html#node

and has addition life cycle events like _process (callback on each frame) https://docs.godotengine.org/en/stable/classes/class_node.html#class-node-method-process

Now I don't even know where to start defining something nice in Ada that would match this pattern and would hide all the nastiness from the C binding. I kind of want the tagged record hierarchy structure, with overriding functions, but it should only register methods with godot you've overridden. How would I know what methods have been overridden? I feel like I need some kind of generic or helper functions?

I'm hoping some more experienced people might have some suggestions?

From: Luke A. Guest <laguest@archeia.com>
Date: Fri, 16 Oct 2020 07:38:05 +0100

> I'm working on a binding to the Godot game engine for Ada.

Ok, so what you have now is a gcc generated binding, which isn't the nicest to work with.

What you really need to do is to start by wrapping up the thin inside a thick binding such that the plug-ins only use the thick binding and that any of the calls such as simple_constructor are wrapped, i.e.

Godot.Make(Instance : Godot.Root_Class; parameters...) -> calls simple_constructor(Instance.Internal_Pointer, parameters). Use overloads for this kind of stuff.

The way I bind to C is like this:

1) If it's a simple function that takes no parameters and returns nothing, then bind directly.

2) If it's a simple return type, use an expression function to bind.

3) Anything else gets a thick binding.

4) Types are mapped onto the C ones, so I lift out the definition from the thin binding and put it in the root package of the thick. I also rename so there's less repetitive stuff like GODOT_VARIANT_* and I case properly, this will be difficult for situations where identifiers are Ada keywords, so rename to something else completely if you have to, just document the change.

Essentially you want all the C nastiness inside the thick binding.

Look at SDLAda for some ideas, but this was done by hand. Anything generated by GCC needs to be hand massaged to be nicer imo.

From: Michael Hardeman <mhardeman25@gmail.com>
Date: Fri, 16 Oct 2020 09:39:17 -0700

Thanks for the detailed reply. Unfortunately I think I didn't get my question across correctly.

I'm pretty familiar with most of the basic stuff I can do in Ada. I'm not asking for general advice on making a thick binding, I'm asking for help with one specific data structure/pattern.

What is the best way to make Ada types/functions that wrap a particular thing:

I just pushed a work in progress branch where you can see what I'm struggling with:

https://github.com/MichaelAllenHardeman/gdnative_ada/blob/feature/adventure_game/examples/adventure_game/src/engine_hooks.adb#L29

https://github.com/MichaelAllenHardeman/gdnative_ada/blob/feature/adventure_game/examples/adventure_game/src/example_object.adb#L90

Is it possible to create a type (tagged record maybe) whose dispatching methods automatically register in some way?

From: Luke A. Guest <laguest@archeia.com>
Date: Fri, 16 Oct 2020 19:09:13 +0100

> Is it possible to create a type (tagged record maybe) who's dispatching methods automatically register in some way?

If you mean call Register(Context); on construction of the object, then have you looked at the factory stuff?

http://www.ada-auth.org/standards/12rm/html/RM-3-9.html#I2118

From: Michael Hardeman <mhardeman25@gmail.com>
Date: Fri, 16 Oct 2020 11:28:30 -0700

not when the object is constructed. I was wondering if something like the following were possible:

```
package GDNative.Thick.Objects is
  type Object is abstract tagged private;
  -- create abstract or null subprograms for
  -- each subprogram here:
  -- https://docs.godotengine.org/en/stable/
  -- classes/class_object.html#class-object
  function Name (Self : in Object'class)
return Wide_String is abstract;
  procedure Initialize  (Self : in out
              Object'class) is null;
  -- etc...
private
  type Object is abstract tagged null
record;
end;
```

But I need some way of knowing here: https://github.com/MichaelAllenHardeman/gdnative_ada/blob/feature/adventure_game/examples/adventure_game/src/engine_hooks.adb#L28

what all the types that extend that object tagged type are, and what all the null methods they've chosen to override are. Kind of like the Java Class() style introspection.

I'm sure there must be some way of doing it better tho, with generics? I'm just not creative enough to see the solution atm.

From: Luke A. Guest <laguest@archeia.com>
Date: Fri, 16 Oct 2020 19:31:56 +0100

> But I need some way of knowing here: https://github.com/MichaelAllenHardeman/gdnative_ada/blob/feature/adventure_game/examples/adventure_game/src/engine_hooks.adb#L28

That's what the generic constructor would allow.

> what all the types that extend that object tagged type are, and what all the null methods they've chosen to override are. Kind of like the Java Class() style introspection.

>

> I'm sure there must be some way of doing it better tho, with generics? I'm just not creative enough to see the solution atm.

You can't know what the null methods are. Why do you even need to know?

I'm probably not understanding this, tbf.

From: AdaMagica <christ-usch.grein@t-online.de>
Date: Sat, 17 Oct 2020 03:09:04 -0700

> package GDNative.Thick.Objects is

> type Object is abstract tagged private;

>

> -- create abstract or null subprograms for each subprogram here:

> -- https://docs.godotengine.org/en/stable/classes/class_object.html#class-object

> function Name (Self : in Object'class) return Wide_String is abstract;

> procedure Initialize (Self : in out Object'class) is null;

> -- etc...

>

> private

> type Object is abstract tagged null record;

> end;

I do not know what you are trying to do, but I see a basic misunderstanding here wrt keyword abstract on operations. It has two fundamentally different purposes:

* When used on a primitive operation of a non-tagged type, it makes an inherited

operation disappear, i.e. this operation does no longer exist, e.g.:

```
type T is range -42..42;
function "/" (L, R: T'Base) return T'Base
        is abstract;
```

* When used on a primitive operation of a tagged type, this operation is dispatching and must be overridden for derived types; e.g.

```
type T is abstract tagged private;
procedure Op(X:T) is abstract;
type T1 is new T with private;
procedure Op(X:T1);
```

Now your

```
function Name (Self : in Object'class)
        return Wide_String is abstract;
```

is a classwide operation, not a primitive operation, so it cannot be overridden. It is not a primitive operation of any type, so it just declares that such an operation cannot exist - a rather useless declaration.

From: Per Sandberg <per.s.sandberg@bahnhof.se>
Date: Sun, 18 Oct 2020 22:21:38 +0200

My usual path is:

1) find bindings in other languages and try to understand their intention.

2) Generate a 1:1 binding to the C API since that will provide a sound ground (this is an 100% automatic process).

3) Write the high-level binding trying to mimic other language bindings while keeping an Ada twist to it,

With a minor effort I managed to do step one and two but step three is the hard one. Have a look on https://github.com/Ada-bindings-project/godot

From: Luke A. Guest <laguest@archeia.com>
Date: Wed, 21 Oct 2020 07:59:00 +0100

>> I'm probably not understanding this, tbf.

>

> Can you explain the Generic Constructor some more? I need to use it now, but I can't exactly figure it out. I found this example: https://www.adacore.com/gems/ada-gem-19 but I have no idea how they can use the 'Input attribute as the constructor function. It doesn't match the signature requested by the generic at all.

>

> I have a simple example I was trying to get working: https://ideone.com/f5bpr9

> Do you think you could help me understand where I'm going wrong here?

>

I've never used it, but this might help https://www.adaic.org/resources/add_content/standards/05rat/html/Rat-2-6.html

From: Michael Hardeman <mhardeman25@gmail.com>
Date: Sun, 25 Oct 2020 20:38:36 -0700

https://github.com/MichaelAllenHardeman/gdnative_ada

I've done an initial pass on the thick binding. [...]

Still, as is, it's pretty nice to use. It only takes just a tiny bit of user code to get an object registered and running a function on each frame.

https://github.com/MichaelAllenHardeman/gdnative_ada/tree/master/examples/adventure_game/src

Windows GUI Frameworks

From: DrPi <314@drpi.fr>
Subject: Which GUI framework?
Date: Thu, 29 Oct 2020 19:48:36 +0100
Newsgroups: comp.lang.ada

I'd like to create a PC (Windows) GUI program. This program needs to be able to create many Windows and tabs in one of them. A working thread receives data from a serial line and sends messages to the GUI to print received content.

I know the most common way is to use GtkAda. The problem is I'm an Ada beginner and I never used Gkt. So, the effort is double.

I have a quite good knowledge of wxWidgets since I have used wxPython for years. I thought I could use wxAda but it seems the project is dead.

Any other binding to wxWidgets that I'm not aware of?

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Thu, 29 Oct 2020 20:23:55 +0100

> Any other binding to wxWidgets that I'm not aware of?

If that is only Windows (are you serious?), you do not need any. Simply use Windows GDI API directly. They are callable from Ada more or less out of the box because Windows handles all objects internally as graphic resources.

There are Win32Ada thin bindings, but it is incomplete and most of the time you do not need it.

The Microsoft's way of defining and using types is so idiotic that no reasonably usable thin Ada bindings are possible. I just declare an Ada counterpart new as appropriate with parameters of types I want in order to avoid casting types.

In short, Windows GDI is ugly but it is native and task-safe. (GtkAda is neither)

From: Randy Brukardt <randy@rrsoftware.com>
Date: Thu, 29 Oct 2020 15:45:09 -0500

> If that is only Windows you do not need any. Simply use Windows GDI API directly. [...] There are Win32Ada thin bindings, but it is incomplete and most of the time you do not need it.

For Win32, both Claw (www.rrsoftware.com) and GWindows provide thick Ada bindings. Much easier to use than raw Win32.

From: DrPi <314@drpi.fr>
Date: Fri, 30 Oct 2020 10:37:13 +0100

> If that is only Windows (are you serious?),

Did I say that? ;)

I currently do my dev on a Windows machine but a cross-platform framework is welcome.

> In short, Windows GDI is ugly but it is native and task-safe. (GtkAda is neither)

Windows GDI... I used it a long time ago. Not my best memory.

From: Luke A. Guest <laguest@archeia.com>
Date: Fri, 30 Oct 2020 09:52:11 +0000

> know the most common way is to use GtkAda. The problem is I'm an Ada beginner and I never used Gtk. So, the effort is double.

Gtk isn't all that pleasant either.

> I have a quite good knowledge of wxWidgets since I have used wxPython for years. I thought I could use wxAda but it seems the project is dead.

Yup, I agree that wxWidgets is much simpler as it was based on MFC, only portable.

At this time wxAda is dead on my hdd right now and not going to be resurrected until I get some money coming in.

> Any other binding to wxWidgets that I'm not aware of?

No, both efforts were abandoned as it was too much work. I have a start to a generator, but like I said, it's not happening right now.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Fri, 30 Oct 2020 10:54:54 +0100

> I currently do my dev on a Windows machine but a cross-platform framework is welcome.

Cross-platform would be:

1. GTK (GtkAda)

2. Qt (not sure about the project name)

3. HTTP (Gnoga)

From: Chris M Moore <zmower@ntlworld.com>
Date: Fri, 30 Oct 2020 11:36:27 +0000

> Cross-platform would be:

Or Tk via https://github.com/simonjwright/tcladashell (or https://github.com/thindil/tashy but I've not used that).

From: Jeffrey R. Carter
Date: Fri, 30 Oct 2020 13:31:45 +0100

Gnoga (https://sourceforge.net/projects/gnoga/) is all Ada (not a binding) and platform independent.

From: DrPi <314@drpi.fr>
Date: Sat, 31 Oct 2020 12:20:41 +0100

Gnoga is very interesting when the GUI is remotely run.

I think using such a system locally is nonsense (very resource hungry).

From: DrPi <314@drpi.fr>
Date: Sat, 31 Oct 2020 12:14:46 +0100

Binding to C++ libraries is a problem.

In the Python world, there are many ways to achieve this.

If I remember well, the author of wxPython has written its own binding system for version 3. Before version 3, he used a "standard" one but with many manual patches.

PySide (Python binding for Qt) authors also have written their own binding system after using one that was not fulfilling their needs.

It's a pity since I like wxWidgets' way of working.

From: DrPi <314@drpi.fr>
Date: Sat, 31 Oct 2020 17:30:15 +0100

Do you know SWIG (http://swig.org/)?

SWIG manages C++ bindings to many languages... but not Ada. However, SWIG tools might be of interest, like the tree parser outputting xml. Maybe SWIG can be modified to manage Ada. Just an idea. But not my skills.

From: Luke A. Guest <laguest@archeia.com>
Date: Sat, 31 Oct 2020 16:35:26 +0000

> Do you know SWIG (http://swig.org/)?

I know of it and no thanks. My generator would actually be simpler.

Publisher/Subscriber for Ada

From: DrPi <314@drpi.fr>
Subject: PubSub
Date: Sat, 31 Oct 2020 18:58:03 +0100
Newsgroups: comp.lang.ada

Another question indirectly concerning GUI programming:

Does an Ada "PubSub" package exist?

Something like this: https://pypubsub.readthedocs.io/en/v4.0.3/

Search on Alire returned no result.

Global search on the internet is "polluted" by many Ada answers.

From: Jeffrey R. Carter
Date: Sat, 31 Oct 2020 19:23:55 +0100

> Global search on the internet is "polluted" by many Ada answers.

There's Google custom search for Ada programming topics at https://thindil.github.io/adasearch/ and the Ada-specific search from the AdaIC at https://www.adaic.org/ada-resources/ada-on-the-web/

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sat, 31 Oct 2020 19:38:09 +0100

> Another question indirectly concerning GUI programming: Does a Ada "PubSub" package exist?

Yes. We have a commercial middleware 100% in Ada. We use that thing in automation and control systems. Naturally, it provides publisher/subscriber services, distributed or not with controlled QoS. That is so to say horizontal communication between applications or tasks. It also has a vertical communication aspect abstracting hardware/protocols from application. E.g. you can publish/subscribe to a MQTT topic, or to an EtherCAT object, or to a CANOpen dictionary object etc without even knowing if that's really the thing, something else or another application.

Having said that, for horizontal communication inside a single process you do not need that in Ada. Many things done for other languages are not needed in Ada.

Ada protected objects and tasks provide much more efficient, safer (typed) and easier to use way to communicate between tasks.

From: DrPi <314@drpi.fr>
Date: Sun, 1 Nov 2020 11:36:37 +0100

> Ada protected objects and tasks provide much more efficient, safer (typed) and easier to use way to communicate between tasks.

What I'm looking for is not inter-task communication. It is some sort of message dispatcher (which is not thread safe). It is like a GUI event manager but for custom events.

A simple description here: https://wiki.wxpython.org/WxLibPubSub

This is very useful when using a GUI since it allows to directly send messages to windows/dialogs/controls.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sun, 1 Nov 2020 12:18:20 +0100

> It is some sort of message dispatcher (which is not thread safe). It is like a GUI event manager but for custom events.

You do not need that stuff. Even less if that is not task safe. In the context of the same task, it is just a call. You need no marshalled arguments because the call is synchronous and it must be synchronous because it is the same task. The very term "event" makes no sense if the task that emits it is the task that consumes it.

> This is very useful when using a GUI since it allows to directly send messages to windows/dialogs/controls.

It is not useful, it is a mess, e.g. in GTK.

Anyway, the standard Ada library contains implementation of FIFO queues. If you want it 1-n rather than 1-1 use a blackboard instead of a FIFO.

Dueling Compilers

From: Jeffrey R. Carter
Subject: Dueling Compilers
Date: Wed, 25 Nov 2020 15:08:40 +0100
Newsgroups: comp.lang.ada

Consider the package

```
with Ada.Containers.Bounded_
Doubly_Linked_Lists;
generic
   type E is private;
package Preelaborable is
   package EL is new
         Ada.Containers.Bounded_
         Doubly_Linked_Lists (
         Element_Type => E);
end Preelaborable;
```

Two Ada-12 compilers give different results on this. Compiler G accepts it without problem. Compiler O rejects it with the error message preelaborable.ads: Error: line 6 col82 LRM:10.2.1(11.8/2), If a pragma Preelaborable_Initialization has been applied to the generic formal, the corresponding actual type must have preelaborable initialization AFAICT from the ARM, the generic formal Element_Type of Ada.Containers. Bounded_Doubly_Linked_Lists does not have pragma Preelaborable_Initialization applied to it. However, the type List, which probably has [sub]components of Element_Type, does.

Which compiler is correct? What is the intent of the ARM?

From: Randy Brukardt <randy@rrsoftware.com>
Date: Wed, 25 Nov 2020 20:19:34 -0600

I'd say both compilers are wrong, in that the RM clearly has a bug here and one of the implementers should have complained about it to the ARG long ago. :-)

I'd suggest you post this question to Ada-Comment so that it gets on the ARG's radar.

(I'll call Preelaborable_Initialization "PI" in the following for my sanity. :-)

It's clear from 10.2.1 that a type with pragma PI which has components of a generic formal type has to have components that have a type with PI. It isn't possible to initialize such components without a function call, so the other possibility does not exist. The Bounded containers are designed such that there are components of the element type (more accurately, a component of an array of the element type). In order for there to be such a component, the formal type must have PI. Ergo, anybody for a bounded container written in Ada is necessarily illegal. This is a problem that someone should have brought up at the ARG.

Since it is not required to write language-defined package bodies in Ada, one could imagine that both compilers are correct in the sense that they are using some non-Ada language to implement the containers. But that is a fiction in the case of the containers (every implementation I know of is in Ada), and in any case, we intended the containers to be implementable in Ada. If they are not, that is a bug.

I don't know what the fix ought to be: adding PI to the formal private type would work, but it would reduce the usability of the containers in non-preelaborated contexts. Similarly, removing the PI from the container would work, but would reduce the usability of the containers in preelaborated contexts. Both seem pretty bad.

I'd be in favor of removing PI and Preelaboration in general from the language (it serves no purpose other than to encourage implementers to make optimizations that they should make anyway - the other intentions don't work or are better handled with other mechanisms), but I doubt that I'd get any support for that.

So this will have to be an ARG question -- I can't answer it definitively.

P.S. If you post this question to Ada-Comment, do me a favor and post this analysis along with it. That will save me having to reproduce it later.

From: Jeffrey R. Carter
Date: Fri, 27 Nov 2020 08:32:41 +0100

> Ergo, anybody for a bounded container written in Ada is necessarily illegal.

I think both compilers are doing macro-expansion of generics, so a generic is only really compiled when it is instantiated. Presumably any test code used actual parameters that the compiler could tell were PI, so they compiled OK.

> adding PI to the formal private type would work, but it would reduce the usability of the containers in non-preelaborated contexts. Similarly, removing the PI from the container would work, but would reduce the usability of the containers in preelaborated contexts. Both seem pretty bad.

I presumed that leaving PI on the container was an oversight.

> So this will have to be an ARG question -- I can't answer it definitively.

OK, I'll research the format of submissions to Ada-Comment and send it in.

> P.S. If you post this question to Ada-Comment, do me a favor and post this analysis along with it. That will save me having to reproduce it later.

I would have done that anyway. Thanks for confirming my suspicion that something is rotten in Denmark.

From: Randy Brukardt
<randy@rrsoftware.com>
Date: Fri, 27 Nov 2020 20:35:57 -0600

> I think both compilers are doing macro-expansion of generics, so a generic is only really compiled when it is instantiated.

That would be an incorrect implementation of generic units in Ada. One has to enforce the language rules only knowing the guaranteed properties of the formal types (knowing nothing about the actual). There is a later legality recheck in the specification of an instance, but that would be irrelevant in this case since the generic unit already is illegal.

> I presumed that leaving PI on the container was an oversight.

It definitely is intended, if the unit is Preelaborated, we definitely want any private types in it to be PI (lest they be unable to be used in Preelaborated units.

From: Jeffrey R. Carter
Date: Thu, 17 Dec 2020 21:22:50 +0100

For those who are interested, this became AI12-0409-1, approved 2020-12-09

From: Randy Brukardt
<randy@rrsoftware.com>
Date: Fri, 18 Dec 2020 20:00:02 -0600

> For those who are interested, this became AI12-0409-1, approved 2020-12-09

For what it's worth, that approval included moving most of AI12-0399-1 to this AI, and making this AI a Binding Interpretation so it applies to Ada 2012 as well. We agreed not to require in the ACATS that implementations define the Preelaborable_Initialization aspect (if they have some other existing way to do this, that's fine by us for Ada 2012), but they can if they want. We will insist that bounded containers have P_I if the element type has P_I, and that they can be instantiated if the element type does not have P_I.

Advent of Code

From: John Perry <john.perry@usm.edu>
Subject: Advent of Code
Date: Fri, 27 Nov 2020 19:12:21 -0800
Newsgroups: comp.lang.ada

Does anyone know about Advent of Code, and has anyone ever participated for Ada? It's typically a sequence of programming puzzles posed as an Advent calendar: one for each new day.

https://adventofcode.com/2020/about

Older examples are here:

https://adventofcode.com/2020/events

I had thought of it, but I don't have too much time. Some languages maintain their own mini-communities and leaderboards, and it might be a way to raise Ada's profile (or even SPARK'S?).

From: Jeremy Grosser
<jeremy@synack.me>
Date: Sat, 28 Nov 2020 19:36:48 -0800

I did Advent of Code in Ada last year. I got distracted by other projects and didn't finish it, but found it to be a very good way to learn with focused problems. My solutions are up on GitHub if you're curious, but knowing what I know now, they're far from optimal and some parts are definitely in need of refactoring.

https://github.com/JeremyGrosser/advent

From: Bojan Petrovic
<bojan_petrovic@fastmail.fm>
Date: Sun, 29 Nov 2020 15:03:45 +0100

I solved a couple of challenges from the last year's AoC in both Ada and Rust, just to get a feel for the differences between them in a puzzle solving context:

https://github.com/ALPHA-60/advent-of-code-2019

I've been organising a weekly recreational coding workshop at my company for the last couple of years, and we've been solving Project Euler and Codility tasks. I stopped doing it in March because of the Covid-19 situation, but we'll reboot it online on December 1st, when AoC 2020 starts, though our schedule will remain the same - one AoC problem per week.

A while ago we did some interview question exercises on #Ada Telegram group, so maybe we can do it again there.

From: John Perry <john.perry@usm.edu>
Date: Mon, 30 Nov 2020 23:08:29 -0800

Well, the first day wasn't too bad. It took me an hour, mainly because I'm not as familiar with Ada as I'd like. Once I re-learned file input & remembered the declare clause, it was quick.

I'll follow Jeremy Grosser's example and post my solutions to GitHub, too.

https://github.com/johnperry-math/AoC2020.git

From: Max Reznik <reznik@adacore.com>
Date: Tue, 1 Dec 2020 03:37:06 -0800

Someone posted on reddit: https://www.reddit.com/r/ada/comments/k4fn9w/anyone_else_participating_in_advent_of_code/

From: gautier_niouzes@hotmail.com
Date: Wed, 2 Dec 2020 12:51:04 -0800

Thanks John for the reminder about the Advent of Code. It's lots of fun!

Just before starting with today's puzzle, I had the idea of programming the solution with HAC (and the LEA editor). The quick edition-compilation-run cycle of HAC is an advantage for this contest. However, today, I was not quick enough to get points. Perhaps another day?

Links to my solutions are at the end of the following post: https://gautiersblog.blogspot.com/2020/12/advent-of-code-2020-with-hac-and-lea.html

From: Max Reznik <reznik@adacore.com>
Date: Wed, 2 Dec 2020 13:29:43 -0800

I gathered a list of GitHub repositories from this topic on a page, if someone wants to see all of them in one place.

https://github.com/reznikmm/ada-howto/tree/advent-2020

I also provided mine Ada solutions as Jupyter Notebooks. You can read them in Markdown or launch in the browser with "launch | binder" button.

Have fun :)

From: Stephen Leake <stephen_leake@stephe-leake.org>
Date: Wed, 02 Dec 2020 14:59:18 -0800

> Just before starting with today's puzzle, I had the idea of programming the solution with HAC (and the LEA editor). The quick edition-compilation-run cycle of HAC is an advantage for this contest.

On these small files, can you really tell the difference in speed between GNAT and HAC? or (insert other favorite editor, mine is Emacs) and LEA? For me, everything is instantaneous.

From: Gautier Write-Only Address <gautier_niouzes@hotmail.com>
Date: Mon, 14 Dec 2020 09:43:05 -0800

> On these small files, can you really tell the difference in speed between GNAT and HAC? or (insert other favorite editor, mine is Emacs) and LEA? For me, everything is instantaneous.

From GNAT Studio I get a range of 1.5 sec (an i5 PC @2.9 GHz) to 9 sec (a lightweight laptop) for building aoc_2020_12.adb (almost a benchmark for easy puzzles ;-)).

On the same source, I run hac -v2 aoc_2020_12.adb:

Compilation finished in 0.000335500 seconds.

Part 1: Manhattan distance of the ship to (0,0): 1631 (1631.0)

Part 2: Manhattan distance of the ship to (0,0): 58606 (58606.0)

VM interpreter done after 0.008894500 seconds.

So, for this kind of puzzle, it makes a difference (correct solution to part 1 was sent at 00:11:01).

But agreed, it's quite rare.

Especially on today's puzzle, I didn't even consider using HAC...

From: John Perry <john.perry@usm.edu>
Date: Mon, 14 Dec 2020 13:56:02 -0800

What follows is a long way of saying "Thank you." :-)

I spend about 2 hours on each puzzle, which probably doesn't speak well of my programming prowess (I've programmed for decades, so I can't really say it's because I'm learning Ada). Somehow I enjoy it enough to come back day after day.

The puzzles themselves are usually easy (to me), and most of the ones with a non-trivial solution can probably be solved trivially, with one exception. At least the mathematics has gotten a little more sophisticated; I used the Chinese Remainder Theorem recently, which I got a kick out of implementing in Ada as a one-line function (not including a support function to compute a modular inverse). I noticed that Maxim used Fermat's Little Theorem.

I sometimes roll my eyes at the puzzles, but the one thing I've really enjoyed so far is how each new puzzle has nudged me to learn a different Ada feature with each new puzzle. I'd spend a lot less time on it if I allowed myself to use a computer algebra system, but the point is to learn Ada, and the really nice surprise has been how people have helped out, some of them even commenting directly on GitHub.

Advent of Code Thread Compilation

From: Alejandro R. Mosteo <amosteo@unizar.es>
Subject: Advent of Code Thread Compilation
Date: Fri, 05 Feb 2021 17:59:27 +0100
To: Ada User Journal Readership

[This is a special message in that I am directly writing it to the Ada User Journal readership. Besides the previous thread on Advent of Code, there were a number of threads for each day. These threads refer to unstated off-groups problems and the discussion is too informal and disjointed to make a coherent post-hoc read, even after summarizing. For that reason, I am not including these threads as-is in the Digest. For the interested readers, I have compiled all the related threads in the newsgroup at the end of this message.

There are nonetheless some interesting tidbits and snippets discussing Ada features, libraries and resources that, even without context, may be useful pointers to follow. I am keeping these in the following messages, with the title of the thread they belong to. —arm]

Day 2: https://groups.google.com/g/comp.lang.ada/c/ASTsQiya1yQ/m/sx27Sb3XAgAJ

Day 3: https://groups.google.com/g/comp.lang.ada/c/zsZV1RSf01c/m/Fl7CTEB2AAAJ

Day 4: https://groups.google.com/g/comp.lang.ada/c/7CmcyU37SkA/m/aI2k3YxfAwAJ

Day 5: https://groups.google.com/g/comp.lang.ada/c/aOF1sirDOiY/m/GEDagaqpAwAJ

Day 6: https://groups.google.com/g/comp.lang.ada/c/co9hjh6F1Ng/m/xbdMecnjAwAJ

Day 8: https://groups.google.com/g/comp.lang.ada/c/jxx-4c2hPng/m/3EO7rO30BAAJ

Day 10: https://groups.google.com/g/comp.lang.ada/c/Z4mmw_t94Ls/m/X2MG3IDfAQAJ

Day 11: https://groups.google.com/g/comp.lang.ada/c/BIBRIl7iirw/m/1tO_250LAgAJ

Day 12: https://groups.google.com/g/comp.lang.ada/c/lqb0iuLXm5E/m/FVGVnyNlAgAJ

Day 17: https://groups.google.com/g/comp.lang.ada/c/lqb0iuLXm5E/m/FVGVnyNlAgAJ

Day 19: https://groups.google.com/g/comp.lang.ada/c/lqb0iuLXm5E/m/FVGVnyNlAgAJ

Day 23: https://groups.google.com/g/comp.lang.ada/c/lqb0iuLXm5E/m/FVGVnyNlAgAJ

Day 25: https://groups.google.com/g/comp.lang.ada/c/zcMzC_q9KmA/m/Aa7iA3q4BAAJ

From: John Perry <john.perry@usm.edu>
Subject: Advent of Code Day 2
Date: Wed, 2 Dec 2020 15:45:25 -0800
Newsgroups: comp.lang.ada

> ...I should have used Gnatcoll.regexp.

I was wondering if there was a pattern matching library I could use, and had wanted to ask that, but forgot.

From: Stephen Leake <stephen_leake@stephe-leake.org>
Date: Thu, 03 Dec 2020 03:52:47 -0800

'Reduce is a new Ada 2020 attribute

(www.ada-auth.org/standards/2xrm/html/RM-4-5-10.html); it can sum an array.

From: John Perry <john.perry@usm.edu>
Subject: Advent of Code Day 3
Date: Sat, 5 Dec 2020 07:11:06 -0800

> Day 4 task is dull :)

>

> https://github.com/reznikmm/ada-howto/blob/advent-2020/md/04/04.md

Flourishes like this:

```
return Passport (byr .. pid) =
          (byr .. pid => True);
```

illustrate idioms that I really want to learn, thanks for sharing.

From: John Perry <john.perry@usm.edu>
Subject: Advent of Code day 5
Date: Sat, 5 Dec 2020 09:57:00 -0800

According to the Internet (And Therefore It Is True (TM)) the A380 can seat up 853 people. My problem had up to 894 seats, with the first 5 missing, so it wasn't that far beyond the realm of reason.

Then again, I don't know if anyone would want to fly an A380 configured for 853 people.

From: Stephen Leake <stephen_leake@stephe-leake.org>
Subject: Advent of Code day 5
Date: Sun, 06 Dec 2020 08:21:24 -0800

> and ran it through cut/sort/uniq

Next time, try ada.containers.generic_array_sort;

http://www.ada-auth.org/standards/2xrm/html/RM-A-18-26.html

From: Stephen Leake <stephen_leake@stephe-leake.org>
Subject: Advent of Code day 5
Date: Sun, 06 Dec 2020 08:27:54 -0800

> Next time, try ada.containers.generic_array_sort;

> http://www.ada-auth.org/standards/2xrm/html/RM-A-18-26.html

Or Doubly_Linked_Lists.Generic_Sorting:

http://www.ada-auth.org/standards/2xrm/html/RM-A-18-3.html

From: Randy Brukardt <randy@rrsoftware.com>
Subject: Advent of Code Day 7
Date: Mon, 7 Dec 2020 17:44:44 -0600

> Entry: Bag_Entry := (Quantity => 10);

>

> However, GNAT says this is invalid [...]

In Ada 2005 and later, write:

```
Entry: Bag_Entry := (Quantity => 10,
          Description => <>);
```

In an aggregate, <> means a default initialized component. Following the Ada Way TM ;-), one has to explicitly ask for a default initialized component - just leaving it out might have been a mistake or intended -- neither the compiler nor a reader can tell. The above is clearly intended.

From: Jeffrey R. Carter
Subject: Advent of Code Day 7
Date: Tue, 8 Dec 2020 12:25:54 +0100

>

> type Bag_Entry is record

> Description: Bag_Description := "";

Humans are notoriously bad at counting things, and even worse at counting things they can't see, so this kind of literal can be a source of errors, especially during modification. (At least with Ada these tend to be compiler errors, not run-time errors.)

Of course, Ada offers a Better Way. You can write

```
Description: Bag_Description :=
          (Bag_Description'range => ' ');
```

or

```
Description: Bag_Description :=
          (others => ' ');
```

and be proof against any changes to Bag_Description's bounds.

From: Stephen Leake <stephen_leake@stephe-leake.org>
Subject: Advent of Code Day 10
Date: Fri, 11 Dec 2020 09:04:27 -0800

> My answer was able to fit in a Long_Long_Integer on my machine. But, due to a bug, I did play with the Big_Integers package. It worked well, and I'd recommend taking a look at it for upcoming

Yes; GNAT Community 2020 with -gnat2020 and -gnatX supports Ada.Numerics.Big_Integer. I updated my solution to use that.

From: Jeffrey R. Carter
Subject: Advent of Code Day 10
Date: Sat, 12 Dec 2020 23:25:41 +0100

> hmm. I got constraint error when I used Long_Integer; maybe that's not 64 bits? Using Ada.Big_Numbers.Big_Integers was a good exercise anyway.

That sounds like C thinking. If you need 64 bits, say so, don't hope that optional language-defined types will be big enough.

```
type S is range -(2 ** 63) + 1 .. 2 ** 63 - 1;
type U is mod 2 ** 64;
```

I used

```
type U is mod
          System.Max_Binary_Modulus;
```

From: Gautier Write-Only Address <gautier_niouzes@hotmail.com>
Subject: Advent of Code Day 17
Date: Fri, 18 Dec 2020 11:47:59 -0800

> Advent of Code hasn't been a *complete* waste of time. ;-)

Far from that: now the major part of the test suite for HAC stems from AoC:

[Omitted output of 27 successful tests for HAC, 15 of them being Advent of Code entries. —arm]

Starting Time of Real-time Clock

From: Simon Wright <simon@pushface.org>
Subject: Ada.Real_Time.Time_First
Date: Wed, 09 Dec 2020 12:30:44 +0000
Newsgroups: comp.lang.ada

I opened an issue[1] on Cortex GNAT RTS, saying

> You'd expect Ada.Real_Time.Time_First to be quite a long time before any possible value of Ada.Real_Time.Clock; but in fact the system starts with Clock equal to Time_First.

On the other hand, I had written

```
Last_Flight_Command_Time :
Ada.Real_Time.Time
:= Ada.Real_Time.Time_First;
...
Quad_Is_Flying :=
  Ada.Real_Time.To_Duration (Now -
    Last_Flight_Command_Time)
  < In_Flight_Time_Threshold;
```

but Now - Last_Flight_Command_Time is going to be quite small, to start with, so Quad_Is_Flying is going to be True when it shouldn't be.

The workaround I used was

```
Quad_Is_Flying :=
  Last_Flight_Command_Time /=
  Ada.Real_Time.Time_First
```

and then

```
Ada.Real_Time.To_Duration (Now -
  Last_Flight_Command_Time)
  < In_Flight_Time_Threshold;
```

In other words, I was using Time_First as a flag to indicate that Last_Flight_Command_Time was invalid.

What would your standard pattern for this sort of problem be? Especially considering that if I make Time_First a large negative number I'll get the opposite problem, e.g. predicting ahead for a very large interval, possibly even leading to numeric overflows.

I'm thinking of a Time type with the concept of validity, possibly built round

```
type Time (Valid : Boolean := False) is
record
  case Valid is
    when True  => Value :
          Ada.Real_Time.Time;
    when False => null;
  end case;
end record;
```

and addition, etc. with appropriate preconditions.

(not so sure about the discriminated record, might be more trouble than it's worth)

[1] https://github.com/simonjwright/*cortex-gnat-rts/issues/33

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Wed, 9 Dec 2020 14:16:10 +0100

> What would your standard pattern for this sort of problem be?

I would use Next_Time instead of Last_Time:

```
Next_Flight_Command_Time : Time :=
Time_First;
begin
  loop
    Now := Clock;
    if Now >= Next_Flight_Command_Time
then
      Fire_All_Rockets;
      Next_Flight_Command_Time :=
        Next_Flight_Command_Time +
        In_Flight_Time_Threshold;
    end if;
  end loop;
exception
  when Constraint_Error =>
   -- the End of Times!
    Put_Line ("Thank you for your
              cooperation!");
    Fire_Death_Star;
    Self_Destroy;
end;
```

From: Simon Wright <simon@pushface.org>
Date: Wed, 09 Dec 2020 20:07:32 +0000

> I would use Next_Time instead of Last_Time:

Great idea; the name isn't right in my context, but the method applies very well. (It's the time by which the next flight command has to have been given before we decide we're not flying anymore. I plead that (a) this logic seems not to be our Earth logic, (b) it's a translation from someone's C, (c) the original code has a comment expressing doubt)

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Wed, 9 Dec 2020 16:21:02 +0200

> I opened an issue[1] on Cortex GNAT RTS, saying

>

> You'd expect Ada.Real_Time.Time_First to be quite a long time before

> any possible value of Ada.Real_Time.Clock; but in fact the system

> starts with Clock equal to Time_First.

I don't see any reason for expecting Time_First to be far in the past relative to program start. In fact, RM D.8(19) says "For example, [the start of Time] can correspond to the time of system initialization".

Contrariwise, it could be useful to know that Clock actually starts from Time_First, because I have often needed a "Start_Time" object that records the Clock at the start of the program, and it would be much simpler to use Time_First, if Time_First is known to equal the initial Clock.

> Quad_Is_Flying :=

> Ada.Real_Time.To_Duration (Now - Last_Flight_Command_Time)

> < In_Flight_Time_Threshold;

If Time_First, as the initial value of Last_Flight_Command_Time, would really be in the far past compared to Now, that computation risks overflowing the range of Duration, which may be as small as one day (86_400 seconds), RM 9.6(27).

> The workaround I used was [...] I was using Time_First as a flag to indicate that Last_Flight_Command_Time was invalid.

Even that can still overflow Duration, if more than one day can pass since the last flight command.

> What would your standard pattern for this sort of problem be?

You have two problems: your assumption about Time_First (or perhaps it's not an assumption, if you make your own RTS) and the possible overflow of Duration.

To indicate an invalid Last_Flight_Command_Time, I would either use a discriminated type wrapping a Time value that depends on a Valid discriminant, as you suggested, or just have a Boolean flag, say Flight_Commands_Given that is initially False. I would use the discriminated type only if there is more than one such variable or object in the program.

For the overflow, I suggest changing the comparison to

```
Now < Last_Flight_Command_Time
    + To_Time_Span
     (In_Flight_Time_Threshold)
```

assuming that Last_Flight_Command_Time is valid in the sense we are discussing. That will overflow only when Last_Flight_Command_Time approaches Time_Last, and the program is likely to fail then anyway.

From: Simon Wright <simon@pushface.org>
Date: Wed, 09 Dec 2020 20:16:24 +0000

[...] This conversation has been very valuable, particularly in the case of other similar tests. I suspect, though, that "are we still flying?" is a question that'll take more thinking to resolve!

Possible to Recover Default Value of Scalar Type?

From: reinert <reinkor@gmail.com>
Subject: Possible to recover default value of scalar type?
Date: Sun, 13 Dec 2020 01:54:40 -0800
Newsgroups: comp.lang.ada

Assume the following code:

```
type A_Type is new Natural range 0..9 with
Default_Value => 9;
A : A_Type;
```

Is it later on here possible to get access to the default value (9)? If A was a component of a record, one could get it "9" via

```
some_record'(others =><>).A
```

But more directly? [Without declaring a variable, as is made clear in some omitted posts. —arm]

From: AdaMagica <christ-usch.grein@t-online.de>
Date: Mon, 14 Dec 2020 01:01:21 -0800

I do not really understand the problem. It seems you want to be able to access the default value like so:

```
N: Natural := Natural(A_Type'Default_Value);
```

This is not possible. There is no corresponding attribute 'Default_Value.

If this presents a real problem, submit it to Ada comment stating why this is important.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Mon, 14 Dec 2020 10:38:40 +0100

> If this presents a real problem, submit it to Ada comment stating why this is important.

It could in the cases like this:

```
procedure Library_Foo (Bar : Baz :=
        Baz'Default_Value)
```

You can declare constants in some places, but not at the library level. But in any case, being forced to declare a constant each time you need to get at the default value?

The same problem arises with container generics. If you have an array keeping container elements, logically freed elements need to be "destroyed" in some way. The default type value would be that thing as well as a default for Null_Element, if used.

I think that all non-limited types one could declare uninitialized, must have S'Default_Value equal to the default value the compiler would use. And it should produce same warnings uninitialized values do:

```
Put_Line (String (1..10)'Default_Value);
-- print garbage
```

From: AdaMagica <christ-usch.grein@t-online.de>
Date: Mon, 14 Dec 2020 07:56:29 -0800

> procedure Library_Foo (Bar : Baz := Baz'Default_Value)

Suppose type Baz has no default value aspect. Then a call to Library_Foo without parameter would use what?

A solution could be that the attribute is illegal if there is no aspect. The compiler knows.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Mon, 14 Dec 2020 17:31:29 +0100

> Suppose type Baz has no default value aspect. Then a call to Library_Foo without parameter would use what?

The default used by the compiler in this:

```
declare
   Bar : Baz;
begin
```

with an appropriate warning of course.

[It was a language design bug to allow implicitly uninitialized variables in the first place. Declarations like above should have been illegal.]

> A solution could be that the attribute is illegal if there is no aspect. The compiler knows.

I would argue that if

```
declare
   Bar : Baz;
begin
```

is legal, then it must be logically equivalent to:

```
declare
   Bar : Baz := Baz'Default_Value;
begin
```

From: Simon Wright <simon@pushface.org>
Date: Mon, 14 Dec 2020 18:24:54 +0000

> [It was a language design bug to allow implicitly uninitialized variables in the first place. Declarations like above should have been illegal.]

There is an argument that you should only initialise variables at the point of declaration if you know what value they should take; so that the compiler can detect the use of uninitialised variables.

If you always initialize variables, even if you don't know what value they should take, the compiler can't help you if you forget to assign the correct value.

Personally I always try hard not to declare an uninitialised variable.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Mon, 14 Dec 2020 19:53:23 +0100

> There is an argument that you should only initialise variables at the point of declaration if you know what value they should take; so that the compiler can detect the use of uninitialised variables.

I think Robert Dewar argued that variables must be declared in the narrowest possible scope. Which would imply that at the beginning of that scope you should know the value, because it would be the first use of the variable.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Mon, 14 Dec 2020 19:21:53 -0600

> N: Natural := Natural(A_Type'Default_Value);

We considered an attribute like that, but it becomes a semantic problem if the type doesn't have a Default_Value and you are in a context where you don't know (such as for a generic formal type). I vaguely remember some other semantic problem, but I don't remember the details. These things could be worked out, but it seemed messy.

I've long wanted <> to work as it does in aggregates generally (if that existed, I'd also have a restriction to require all objects to be initialized; that would provide an encouragement to initialize as many objects as possible; right now, the iffy thing (not initializing) is the easiest).

From: Randy Brukardt <randy@rrsoftware.com>
Date: Mon, 14 Dec 2020 19:26:10 -0600

> procedure Library_Foo (Bar : Baz := Baz'Default_Value)

I would have suggested to write this as:

```
procedure Library_Foo (Bar : Baz := <>)
```

since this is the syntax used in aggregates (and why should aggregates have all the fun??).

> Put_Line (String (1..10)'Default_Value); -- print garbage

The above isn't a legal attribute prefix in any case (can't slice a type). And you don't need to because this is clearly an aggregate (which is legal in Ada 2012):

```
Put_Line (String'(1..10 => <>));
-- print garbage
```

From: Randy Brukardt <randy@rrsoftware.com>
Date: Mon, 14 Dec 2020 19:27:39 -0600

> The compiler knows.

Not always. Never forget generics. One would hope to be able to use this on generic formal types, as most of them are going to have default values (at least in new code).

From: J-P. Rosen <rosen@adalog.fr>
Date: Tue, 15 Dec 2020 07:47:32 +0100

> I think Robert Dewar argued that variables must be declared in the narrowest possible scope.

Not applicable if your variable is used in a loop:

```
   V : Integer;
begin
   loop
      Get (V);
      exit when V =0;
      -- do something with V
   end loop;
```

Clearly, initializing V makes no sense.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Tue, 15 Dec 2020 08:23:33 +0100

> Not applicable if your variable is used in a loop

```
   loop
      declare
         V : constant Integer := Get;
      begin
         exit when V = 0;
         -- do something with V
      end;
   end loop;
```

It is related to another long standing issue with returning values (multiple values) from functions and functions with in out parameters (resolved recently).

[...]

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Tue, 15 Dec 2020 08:35:32 +0100

>> Put_Line (String (1..10)'Default_Value); -- print garbage

> The above isn't a legal attribute prefix in any case (can't slice a type).

I mean a subtype.

> And you don't need to because this is clearly an aggregate (which is legal in Ada 2012):

> Put_Line (String'(1..10 => <>)); -- print garbage

Yes, I would prefer the box notation too. However having a proper name would has some advantages too:

```
subtype S is T range T'Default_Value - 100..T'Default_Value + 100;
```

From: Randy Brukardt <randy@rrsoftware.com>
Date: Wed, 16 Dec 2020 18:43:56 -0600

> subtype S is T range T'Default_Value - 100..T'Default_Value + 100;

If box was generally allowed, you could qualify it to get this effect:

```
subtype S is T range T'(<>) - 100 .. T'(<>) + 100; -- Not Ada, but should be IMHO. :-)
```

and it's shorter, too. Of course, if T doesn't have a default value, neither of the above is a good idea. :-)

From: J-P. Rosen <rosen@adalog.fr>
Date: Tue, 15 Dec 2020 10:07:02 +0100

> V : constant Integer := Get;

Well, you can push anything in a function, but it's not always clear/readable/simpler...

> V : Integer := <>; -- Invented syntax for explicit lack of initialization

That would make more sense: make initialization required, and say so if you don't care.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Wed, 16 Dec 2020 18:48:06 -0600

> Clearly, initializing V makes no sense.

Saying that you *meant* to have an uninitialized value does make sense, though:

```
V : Integer := <>;
-- Not Ada, but should be IMHO.
```

Whenever something is omitted, one never knows whether it was on purpose or a mistake. You get similar issues when "else" is omitted (RR's style guide only allows that in very specific circumstances). It's unfortunate that Ada doesn't have a positive way to indicate default initialization, outside of aggregates.

From: AdaMagica <christ-usch.grein@t-online.de>
Date: Tue, 15 Dec 2020 10:14:59 -0800

Just a story about my work (long ago):

Our coding standard required for every type declaration a default value that indicated an uninitialised value:

```
type T is ...
Nd_T : constant T := ...;  -- Nd: not defined
X: T := Nd_T;  -- required
```

The idea was that this Nd value should be thus that it would be likely to produce an exception when used in an expression. Also any change of this value should have absolutely no effect on the code. In any case, at some time it was decided that the Nd value for numeric types was 0. The effect: It was no longer possible to see whether in a declaration like

```
X: T := Nd_T;
```

denoted a truly undefined value or a concrete and correct initial value.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Wed, 16 Dec 2020 18:53:06 -0600

> It was no longer possible to see whether in a declaration [...] this value denoted a truly undefined value or a concrete and correct initial value.

Typically, values like this, at least those used in debuggers, use some permutation of 16#DEADBEEF# since it is obvious in data dumps, and is a rather unlikely value to be intended. The next version of Janus/Ada will initialize all "uninitialized" objects to this value unless you tell it not to. (Essentially, a version of Normalize_Scalars, except that these days it doesn't make much sense for that not to be the default. Optimization can remove most unneeded initializations, and if they are actually needed, it's better to have a known dubious value than stack garbage.)

Ada Syntax Questions

From: DrPi <314@drpi.fr>
Subject: Ada syntax questions
Date: Thu, 17 Dec 2020 23:39:44 +0100
Newsgroups: comp.lang.ada

Ada claims to have a better syntax than other languages. I'm fine with, but...

1) What about array indexing ?

In some other languages, arrays are indexed using square brackets. In Ada, parentheses are used for function calls and for array indexing. In the code "status := NewStatus(some_var);", you can't tell if NewStatus is a function or an array.

2) In Ada, a function without arguments is called without any parentheses.

In the code "status := NewStatus;", you can't tell if NewStatus is a function or a variable.

For my knowledge, are there good reasons for these syntaxes?

From: Gabriele Galeotti <gabriele.galeotti.xyz@gmail.com>
Date: Thu, 17 Dec 2020 15:18:34 -0800

1) This allows you to replace your array with a function with the same name, which takes the subscript as an argument and returns a value, without touching your client code. Think about an expensive lookup table -vs- a simple function which computes your data. Do not see this as an ambiguity but rather a nice uniformity of calling something for a value.

2) Nearly the same, but in another context and without an argument. "NewStatus" could be, e.g., a constant, as long as types match.

From: Jeffrey R. Carter
Date: Fri, 18 Dec 2020 09:26:39 +0100

> 1) What about array indexing?

The requirements for the language included a restricted set of characters for source code that did not include brackets. So that is the primary reason parentheses are used.

However, both arrays and functions are often used as maps, and so an after-the-fact rationalization is that using the same syntax for both array indexing and function calls makes it easy to switch between the two.

> 2) In Ada, a function without arguments is called without any parentheses.

> In the code "status := NewStatus;", you can't tell if NewStatus is a function or a variable.

That's because Newstatus is a terrible name. If you'd used New_Status there would be no confusion.

Seriously: Ada 80 required empty parentheses for a subprogram call with no explicit parameters. During the review process that resulted in Ada 83, these were universally reviled and so were eliminated.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Fri, 18 Dec 2020 10:18:45 +0100

1. Separation of interface and implementation. Being an array or function is an implementation detail of a map or a named entity.

Another example is pointer dereferencing. In Ada X.A is the same as P.A. In C you have X.A vs P->A.

Yet another one. All instances of parameterization in Ada deploy () parentheses. In C++ it would be <>, [], (), depending on semantically irrelevant context.

2. Languages that like C use bottom-up matching are forced to distinguish certain

things prematurely on the syntax level. This is also the reason why you cannot use the result type to distinguish signatures in C++, but you can in Ada. Thus in C++ you would have something as disgusting as

```
123ull
```

while in Ada it is just

```
123
```

Long time ago anything but strictly bottom-up matching was considered too complicated or impossible. So artificial distinctions like () vs [] were invented and then promoted into orthodoxy.

From: Mart van de Wege <mvdwege@gmail.com>
Date: Fri, 18 Dec 2020 17:55:56 +0100

> 1) What about array indexing ?

Why would you care? It is obvious that NewStatus will return something based on the value of some_var. How it does that, by array dereference or function call should make no difference to the caller; they are only interested in the final value of status.

Or another look at it: array indexing is effectively a function call anyway. It is "return value of array_base + index".

> 2) In Ada, a function without arguments is called without any parentheses.

Again, why would you care how NewStatus returns a value? Either by returning the value of a function or by dereferencing a variable, all you're interested in is the value assigned to status.

From: Björn Lundin <b.f.lundin@gmail.com>
Date: Fri, 18 Dec 2020 18:38:27 +0100

> 2) In Ada, a function without arguments is called without any parentheses.

As others have stated, why do you care?

I often mock up a function with a constant, add a pragma compile_time_warning/error ("fix implementation later") and only later write the body of the function. And that is the only code change - I don't need to add an useless empty pair of () just because it is a function to all the callers

> For my knowledge, are there good reasons for these syntaxes?

Yes

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Fri, 18 Dec 2020 21:35:37 +0200

> Ada claims to have a better syntax than other languages.

I would say the claim is that the Ada syntax was rationally designed to have certain properties, which are desired by certain users (us Ada programmers) so it is "better" for us, although some aspects are subjective for sure.

In addition to what others have said, here are some further comments on

the examples you gave:

> 1) What about array indexing?

There are proposals to allow [] as well as (), mainly to increase familiarity for new Ada users.

> 2) In Ada, a function without arguments is called without any parentheses.

Parameterless functions are rare, and properly so.

Parameterless procedures are much more common. Writing

```
Frobnicate_Widget();
```

is longer than

```
Frobnicate_Widget;
```

and seems to have no advantages over the shorter form.

From: Stephen Leake <stephen_leake@stephe-leake.org>
Date: Fri, 18 Dec 2020 15:09:19 -0800

> 1) What about array indexing?

This is true.

You seem to be implying this is bad; why?

> 2) In Ada, a function without arguments is called without any parentheses.

This is true.

You seem to be implying this is bad; why?

> For my knowledge, are there good reasons for these syntaxes?

Yes. See the Ada Rationale: http://ada-auth.org/standards/rationale12.html

From: DrPi <314@drpi.fr>
Date: Sat, 19 Dec 2020 12:50:40 +0100

Thanks all for your answers.

> Why would you care?

Calling a function can have side effects. Accessing an array or a variable can't have side effects.

> You seem to be implying this is bad; why?

Reading the code can't tell you the writer's intentions.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sat, 19 Dec 2020 13:40:25 +0100

> Calling a function can have side effects. Accessing an array or a variable can't have side effects.

Untrue. Both array and variable access have side effects on the registers, on the cache, on the process memory paging, in the form of exception propagation etc. Even direct effects on the outside world are possible when using machine memory load instructions. E.g. on some hardware reading memory at the specific address location means physical serial input.

All these effects are either desired parts of the implementation or else bugs to be fixed. If desired, why do you care?

> Reading the code can't tell you the writer's intentions.

What intentions? Unless you are talking about the intention to deploy a specific machine instruction, function or array gives you no clue. But even then. PDP-11 FORTRAN IV used subprogram calls to implement basically everything, elementary arithmetic operations. If the function is inlined, where is any call? Functions can be tabulated into lookup tables. Arrays can be compressed into functions.

From: AdaMagica <christ-usch.grein@t-online.de>
Date: Sat, 19 Dec 2020 09:01:53 -0800

> Calling a function can have side effects. Accessing an array or a variable can't have side effects.

The declaration of the function is a contract about pre and post conditions, albeit in Ada incomplete. In SPARK, the contract is firm. As a user of the function, you have to believe the programmer that he follows the contract. If the implementation needs a side effect, so be it.

If on the other hand you are a maintainer or are chasing a bug, you have to check the requirements first, not the body of the function. This comes later.

> Reading the code can't tell you the writer's intentions.

The intentions are in the requirements (or in the accompanying comments, you hope they are up to date and not wrong). If there are none, good luck.

From: Andreas Zuercher <zuercher_andreas@outlook.com>
Date: Sat, 19 Dec 2020 09:13:56 -0800

> Untrue. Both array and variable access have side effects on the registers, on the cache, on the process memory paging, in the form of exception propagation etc.

Dmitry, DrPi here is referring to side-effects as viewed from the functional-programming paradigm's perspective. Some programming languages have a "pure" designator (usually the keyword: pure) that assures that this subroutine and all invoked subroutines therein are pure (i.e., have no FP side effects).

The side effects of which you speak are at the machine-code level: e.g., setting/clearing comparison flag(s), setting/clearing carry flag, setting/clearing overflow/underflow flag(s), evictions from L1/L2/L3 cache, (on RISC processors) latching an address in preparation of a load/store, and so forth. None of these are externally observable side effects from FP's perspective above the machine-code level. DrPi's FP goals are valid.

> > Reading the code can't tell you the writer's intentions.

> What intentions?

The intentions of the Ada programmer to design an overtly FP-pure or either an overtly FP-impure subroutine or an FP-impure subroutine by happenstance. Subroutine here is preferably a function, preferably at that a single-parameter function (for ability to utilize over a century of mathematical-analysis techniques). Ada is showing its 1970s vintage by unfortunately omitting overtly expressing FP pureness as a fundamental principle (among a few other FP features). DrPi's FP goals are valid.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sat, 19 Dec 2020 18:49:08 +0100

> The side effects of which you speak are at the machine-code level

Memory paging is pretty much observable.

What you are saying is a question of contracts. The contract must include all effects the user may rely on. The contract of a function may include observable effects or have none (to some extent).

If contracts were indeed relevant to the syntax then functions without contracted side effects must have been called using [] instead of ().

No? Then it is not about the contracts.

>>> Reading the code can't tell you the writer's intentions.

>> What intentions?

> The intentions of the Ada programmer to design an overtly FP-pure or either an overtly FP-impure subroutine or an FP-impure subroutine by happenstance.

Intentions are constraints expressed by contracts. Everything else is implementation details.

Ada programmers are not motivated by pureness of a subroutine. These are totally irrelevant. What is relevant is the strength of the contract. Functions without side effects are preferable just because they have weakest preconditions and strongest postconditions. Side effects weaken postconditions.

For the clients these are of no interest, even less to deserve a different syntax. The user must simply obey the contract whatever it be, ignoring the implementation as much as possible.

Ada's unified syntax is a great help here. I quite often replace arrays and variables with functions. It would be great if literals were fully equivalent to parameterless functions.

From: Andreas Zuercher <zuercher_andreas@outlook.com>
Date: Sat, 19 Dec 2020 10:40:53 -0800

> No? Then it is not about the contracts.

As witnessed by your final sentence quoted below and multiple other replies along this thread, the key tactical advantage of Ada's usage parentheses for array indexing is to accomplish a switcheroo days, weeks, months, years, or decades later: to substitute a function invocation later for what was formerly an array index. Cute trick. Advantageous in some situations. But for people like DrPi who seek contractual assurance of FP-purity of (all?) invoked functions (and overt declaration of impurity of other functions), Ada's 1) implicit switcheroo there in unfortunate combination with Ada's 2) lack of flamboyantly advertising impurity in the replacement function does in fact violate the purity portion of the contract that the mere offset-into-array implementation had—and indeed •overtly• declared in its specification as a mere offset-into-array operation-of-unquestionable-purity.

It is okay for a 1970s Ada to not foresee this, because FP was not a mainstream programming practice back then. (But, btw, it is not as okay for there to be a lack of HOLWGn each decade since the 1980s to revisit whether HOLWG1 forgot anything, where $n>1$, $n \in \mathbb{Z}$.) This 1970s faux pas in letting a silent slip-streamed switcheroo into the core contract-definition declaration mechanism of Ada (not comments! btw, tisk tisk) is merely some tarnish that an AdaNG (next-generation Ada) would fix: e.g., by mandating that all functions (and procedures?) shall be overtly declared & enforced to be pure or impure, which would then mean that only pure functions could substitute for array indexing is the ()-based switcheroo on which so many replies in this thread hang their hat. And DrPi would enjoy seeing the compile-time errors emerge when some cavalier programmer over yonder changed an array index to an •impure• function invocation as contract violation. The cute implicit switcheroo isn't evil, but the lack of compile-time detection of impurity in the switcherooed function is what is evil. (While drinking tea as none of my business as the meme goes,) I actually claim that Ada's usage of parentheses for array indexing was merely happenstance copying the Fortran-PL/I-PL/1-Simula-PL/P-PL/M/CHILL heritage popular in the 1970s*, which itself mimicked mathematics' usage of parentheses around each matrix. Because there was no way to represent mathematics' subscripts as the notation for indexing, the next best punctuation for matrix/vector indexing was borrowed: parentheses.

* as opposed to the ALGOL58's, ALGOL60's, ALGOL68's, BCPL's, C's square brackets, so the big split was somewhere around 1957 for FORTRAN (and whichever predecessor languages influenced it) and 1958 for ALGOL58 (and whichever predecessor languages influenced it), as opposed to APL's ⍳ iota which uses neither parentheses nor square brackets to pull out an element since 1966

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sat, 19 Dec 2020 20:37:31 +0100

> [...] the key tactical advantage of Ada's usage parentheses for array indexing is to accomplish a switcheroo

Not substitute, but to provide whatever implementation necessary. In fact Ada is limited in terms of abstractions. There still exist things which cannot be implemented by user-defined subprograms. Ideally there should be none. Whatever syntax sugar, there should be always a possibility to back it by a user-provided primitive operation.

> But for people like DrPi who seek contractual assurance of FP-purity of (all?) invoked functions (and overt declaration of impurity of other functions),

If they are unsatisfied with the higher abstraction level of Ada, they can switch to lower-level languages where implementation details are exposed in syntax. The best we can do is to explain why such exposure is a bad idea.

[Conceptually Ada has nothing to do with FP and I sincerely hope it never will.]

> This 1970s faux pas [...] is merely some tarnish that an AdaNG would fix

This would be highly undesired. On the contrary impure array implementations are all OK to implement various heuristics and caching schemes on the container side. In fact, Ada moved in that direction already by providing crude user-defined array indexing. Clearly as hardware evolves towards parallel architectures with partitioned memory, low-level arrays will be less frequently exposed in interfaces. Comparing older and newer Ada code we can see that trend of moving away from plain arrays.

Furthermore, purity of implementation is not contract, per definition of. Purity is a non-functional requirement.

There is only few areas of interest for such:

1. Compile-time evaluation/initialization of static objects and constraints.
2. Optimization, especially in the cases of fine grained parallelism.

In any case there is no reason to reflect that in the syntax, whatsoever.

From: Andreas Zuercher <zuercher_andreas@outlook.com>
Date: Sat, 19 Dec 2020 14:11:59 -0800

> If they are unsatisfied with the higher abstraction level of Ada, they can switch to lower-level languages where implementation details are exposed in syntax.

No, Dmitry, that is where you are wrong. In this regard, Ada is the lower-level, grungier, cruder, uncouther programming language, closer to assembly language or ALGOL60. Languages that have a pure keyword (or equivalent elective designator for compile-time purity enforcement throughout a call-tree of subroutines) are the ones that are high-level, cleaner, more-sophisticated, more-refined programming languages, closer to the lofty heaven of mathematics. This is actually a sad commentary on software engineering as a professed practice that we cannot even agree which programming-language feature-sets are higher-level versus lower-level, grungier versus cleaner, cruder versus more sophisticated, and uncouther versus more refined.

There is no good reason for Ada to lack all of the mechanisms to support FP (other than historical happenstance, then being substantially frozen in a Steelman mindset without any follow-on Stainlessman (arguably Ada95's, Ada2005's, Ada2012's would-be set of requirements that they have incrementally grown into) then Silverman (arguably SPARK's would-be set of requirements that is an ever-closer-to-finished work-in-progress) then Iridiumman then Goldman then Palladiummand then Platinumman evermore sophisticated requirements for a best-practices programming language to live up to as humankind's understanding of programming, system engineering, software engineering, and mathematics advances over time).

> Furthermore, purity of implementation is not contract, per definition of. Purity is a non-functional requirement.

So is all of Ada's rich typing/subtypes. Ada is simply capable of expressing some categories of nonfunctional requirements of the design (e.g., rich typing) but not other more-modern categories of nonfunctional requirement (e.g., a pure keyword).

From: Stephen Leake <stephen_leake@stephe-leake.org>
Date: Sat, 19 Dec 2020 13:51:35 -0800

> Reading the code can't tell you the writer's intentions.

That's what comments and design documents are for.

From: Andreas Zuercher <zuercher_andreas@outlook.com>
Date: Sat, 19 Dec 2020 14:20:52 -0800

> > Reading the code can't tell you the writer's intentions.

> That's what comments and design documents are for.

For decades, assembly-language programmers said the same thing about structured-programming feature-set as being representable in mere comments & design documents. For decades, C programmers said the same thing about Ada's and C++'s and now Rust's feature-sets as being representable in mere comments & design documents. Arguably, the entire history of programming from Fortran (1957) and ALGOL (1958) forward is to encode the designer's intentions in source code that is vetted by a compiler instead of merely letting comments and design documents bit-rot as the declarative & imperative source code marches onward in the flow of time during initial greenfield completion (after all the “then a miracle occurs” on the blackboard sketches become rubber meeting road) and then during maintenance (as the design incrementally changes).

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sun, 20 Dec 2020 09:47:50 +0100

> No, Dmitry, that is where you are wrong. In this regard, Ada is the lower-level, grungier, cruder, uncouther programming language, closer to assembly language or ALGOL60.

Then we disagree on the definition of higher level. Mine is the level of abstraction away from calculus toward the problem space entities.

[...]

> So is all of Ada's rich typing/subtypes. Ada is simply capable of expressing some categories of nonfunctional requirements of the design (e.g., rich typing) but not other more-modern categories of nonfunctional requirement (e.g., a pure keyword).

The abstract datatype (in its original sense, rather than as abstract type in Ada) is meant to be a part of abstraction expressing the problem space. Purity of whatever implementation has nothing to do with the problem space. It is a design artifact.

Moreover, from the standpoint of programming paradigm, the whole procedural decomposition is lower level than OO decomposition done in terms of types and sets of types.

FP sits firmly in the procedural world. Even ignoring all fundamental flaws of FP concept, you will find no interest in FP from my side.

From: DrPi <314@drpi.fr>
Date: Sun, 20 Dec 2020 15:10:47 +0100

>> Reading the code can't tell you the writer's intentions.

> That's what comments and design documents are for.

A good IDE with code analysis showing you object declaration/use is very useful. Especially when comments are out of sync with the code.

I'm surprised that no modern tool/language allows the programmer to embed a "complete" documentation in source files. I'm not talking about comments formatted to suit a specific tool convention, like Python or Perl doc-strings. I'm talking about embedding schematics, drawings, bitmaps, mathematical equations, etc directly in the source code. Or maybe the reverse: embed source code in standard document. Like javascript in SVG files. Why not a .odt file with code sections? Ok, a specific file format would be better. Of course, the editor should be specific. No more a simple text editor.

From: Andreas Zuercher <zuercher_andreas@outlook.com>
Date: Sun, 20 Dec 2020 08:53:36 -0800

> Then we disagree on the definition of higher level. Mine is the level of abstraction away from calculus toward the problem space entities.

Ada's inexpressiveness of imprecision of vagueness of misrepresenting design intent in this regard (of inability to compile-time enforce purity of subroutines) is clearly not abstraction. It is mere self-imposed blindness, ignoring the purity-enforcement topic altogether. Assembly language and Ada have the same inability to overtly express and enforce a declaration of FP-purity. Other languages have a pure keyword or equivalent for subroutines (i.e., functions, procedures, lambdas, coroutines, generators) to overtly express compile-time-enforced purity of the subroutine not making modifications to any data outside of its parameter data and callstack-based transient data. Clearly when a programming language (i.e., Ada) and assembly language share the same lack of feature, they are the more-primitive. Clearly when other pure-keyword-equipped programming languages can facilitate & enforce a higher civilization to capture the finer points of a

mathematical description of the problem domain via a rule-declaration & compile-time enforcement that assembly language lacks, they are higher-order and less primitive. There is no valid definition of "higher-order programming language" that permits assembly language's lack of a pure keyword (or equivalent purity-enforcement mechanism) to be a higher-order language than, say, Scala with a pure keyword. Dmitry, your line of reasoning here of what constitutes a higher-order language is preposterous!

From: Stéphane Rivière <stef@genesix.fr>
Date: Tue, 22 Dec 2020 11:05:10 +0100

> Ada's inexpressiveness of imprecision of vagueness of misrepresenting design intent in this regard (of inability to .../...

Thanks for your message. It makes my day. I'm not fluent as you in english, nor in Ada concepts (I just use it with joy), but let me express my admiration for assertions such as:

> Assembly language and Ada have the same inability to overtly express and enforce a declaration of FP-purity.

Although this thought also plunges me into an abyss of reflection:

> Ada's inexpressiveness of imprecision of vagueness of misrepresenting design intent in this regard (of inability to compile-time enforce purity of subroutines) is clearly not abstraction.

There remains a mystery.

Why does your message remind me of this scene from another genius, Stanley Kubrick?
https://www.youtube.com/watch?v=iAHJCPoWCC8

No need to answer me, I don't have your skills to debate it. Just be assured that this post is not mocking and more expressing amazement.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Mon, 21 Dec 2020 18:58:51 -0600

>Ada's inexpressiveness of imprecision of vagueness of misrepresenting design intent in this regard (of inability to compile-time enforce purity of subroutines) ...

Which Ada? Ada 202x has Global aspects specifically for this purpose, and they are compile-time enforced. Methinks are you simply looking to troll Ada rather than any serious intent.

There's no implementation of Global yet, sadly. Hopefully coming soon.

From: Andreas Zuercher <zuercher_andreas@outlook.com>
Date: Mon, 21 Dec 2020 18:39:45 -0800

> Ada 202x has Global aspects specifically for this purpose, and they are compile-time enforced.

This is very good news. I will need to investigate those AIs further. I take it from your wording that Global aspects are a general mechanism that a codebase could use to implement e.g. the purity check that FP seeks. If a general mechanism, it will be interesting to foresee what other categories of axioms can be enforced/assured beyond purity. Btw, I botched my example of extant programming languages in a prior comment that has a purity check on a call tree. D has it currently, but it has been proposed but not yet incorporated into Scala.

> Methinks are you simply looking to troll Ada rather than any serious intent.

No, absolutely not, at least not in the pejorative [sense] that your wording implies. As a system-engineer •critic• of finding the flaws in the system at large, I am always performing gap analysis on current Ada versus desired state of a universal programming language, using a technique not unlike FMEA. At some level you are coincidentally correct: I am negatively disappointed with Ada as much as C++ as much as Scala as much as D as much as Kotlin as much as Swift as much as C# as much as OCaml, but in different ways and to different degrees for each language.

For example, I admire so many portions of Ada, especially its declarative rich typing expressivity and its 35-year lead in accomplishing much of what C++20 will finally get with their oft-pursued concept feature. Conversely, it is sad that few people realize that Ada has had much of the new whizbang C++20 concept feature for 35 years.

It is as if Ada is a mostly superior product whose salesmen don't consummate as many sales contracts as they ought. It is useful to study in depth precisely why the superior product partially fails to achieve its potential glory.

One of the most interesting successes of Ada is that its user community seems to have fairly consistently utilized the vast majority of the features of the language on a regular basis. Despite C++'s perceived popularity by comparison, each C++ codebase utilizes 10% of C++, but worse it is a different 10% of C++ utilized for each different codebase with vast rivalry between codebases regarding which portions of C++ are God's gift to humankind and which portions of C++ are uncouth. Hence, C++'s perceived popularity is more of a mirage than it first appears because there is no one C++ that is popular, but rather a hundred subsets of C++, 75 of which are intensely unpopular to each of the others and 24 of which are eye-rollingly barely tolerable to each of the others.

As no small achievement, Ada achieves Scott McNealy's "all the wood behind one arrow" vastly more than, say, C++'s or D's everything-and-the-kitchen-sink pandering to me-too-isms. Scala/JVM, Scala/Native, Scala/OO, and Scala/FP are constantly in a multi-way tug-of-war of sorts (actually 2 orthogonal tugs-of-wars at 2 different ontological levels) that again isn't "all the wood behind one arrow" that Ada better achieves than Scala (so far).

> There's no implementation of Global yet, sadly. Hopefully coming soon.

It will be interesting to see the furthest push-the-limits extent of applicability of Global aspects.

From: Keith Thompson <keith.s.thompson+u@gmail.com>
Date: Sun, 20 Dec 2020 13:59:20 -0800

I've never found any of the arguments in favor of using parentheses for array indexing convincing, and I've never liked the way Ada does it. But of course the decision was made in the early 1980s, and it can't be changed now.

At least part of the reason was that Ada needed to be used on systems that didn't have '[' and ']' in their character sets. I don't know to what extent that necessity has been used as an after the fact rationalization.

Function calls and array indexing can be substituted for one another in *some* circumstances, but not in all. But they really are very different things. A function call executes user-written code, and may have side effects; an array indexing expression refers to an object. An array indexing expression can appear on the LHS of an assignment; a function call can't.

If Ada had originally used '[' and ']' for array indexing, I doubt that anyone would be complaining that it would have been better to use '(' and ')' (other than some Fortran programmers, I suppose).

Why not use parentheses for record components, Object(Component) rather than Object.Component Doesn't the same argument apply?

> There are proposals to allow [] as well as (), mainly to increase familiarity for new Ada users.

Ick. The only thing more confusing than using () for array indexing would be allowing either () or [] at the programmer's whim. (Well, not the only thing; I'm sure I could come up with something even worse.)

> Parameterless procedures are much more common. Writing

> Frobnicate_Widget();

> is longer than

> Frobnicate_Widget;

> and seems to have no advantages over the shorter form.

I wouldn't have expected the designers of Ada to be concerned about saving two characters.

I see your point about procedure calls. A statement consisting of an identifier followed by a semicolon can only be a procedure call (I think), so there's no ambiguity. My mild dislike for the function call syntax is that it needlessly treats the zero-parameter case as special.

There could also be some potential ambiguities, though I'm not aware of any actual ambiguous cases in Ada. In some languages, the name of a function not followed by parentheses refers to the function itself (or its address) and does not call it. I can easily imagine an attribute for which Func'Attribute could sensibly refer either to the function Func itself or to the value returned by calling it.

Again, if Ada 83 had required empty parentheses on parameterless procedure and function calls, I'm skeptical that anyone would now be arguing that it was a bad decision.

And again, it would be impossible to change it without breaking existing code.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Mon, 21 Dec 2020 09:08:30 +0100

> Function calls and array indexing can be substituted for one another in *some* circumstances, but not it all.

IMO the only circumstances violating this substitutability are language design bugs and deficiencies:

- Passing array elements in in-out mode

- Assigning array elements

- Multidimensional indices

- Slices

all these must be substitutable with user-defined subprograms.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Mon, 21 Dec 2020 19:04:43 -0600

> Function calls and array indexing can be substituted for one another in *some* circumstances, but not it all.

This is false in modern languages with user-defined indexing (Ada and C++ included), since what looks like array indexing can actually be implemented with a function call.

Not having variable returning functions is a flaw in Ada, IMHO. These days, I think there are still too many special cases in Ada. If I was starting today, () would be a function call, and . would be selection/dereferencing, and there would not be anything else (which means getting rid of type conversions, array indexing and slicing, and anything else I've forgotten about). Compilers are smart enough to generate better code when they know something about the function involved (including if it is that of a predefined container). Doing that would allow overloading to be more general and to allow for the complication of variable returning functions.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Tue, 22 Dec 2020 09:00:14 +0100

> If I was starting today, () would be a function call, and . would be selection/dereferencing, and there would not be anything else

But you cannot get rid of X(...) syntax, where X is an object. It is not only indexing, e.g. in declarations:

```
X : T (Y);
```

Then what is wrong with indexing? It should simply apply to all types [from some predefined class]:

```
X (...) ::= CALL (<index-operation>, X, ...)
(...) ::= CALL (<aggregate-operation>, ...)
```

From: Randy Brukardt <randy@rrsoftware.com>
Date: Tue, 22 Dec 2020 19:23:51 -0600

> Then what is wrong with indexing?

Nothing is "wrong" with it, it is just redundant. As others have noted here, both indexes and function calls represent a mapping. What's the point of having two ways to represent a mapping? In an Ada-like language, there's no syntax nor semantic difference.

Ada (and most other languages) are full of redundant stuff. Simplify the basics and then one has more room for interesting stuff (static analysis, parallel execution, etc.).

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Wed, 23 Dec 2020 09:59:46 +0100

>> But you cannot get rid of X(...) syntax, where X is an object.

> That's a prefixed view, of course. No one would want to get rid of that.

Hmm, where is the operation? A prefixed view is

```
<expression>.<operation>(...)
```

Indexing is

```
<expression>(...)
```

In particular:

```
"abc"(1)
```

>> It is not only indexing, e.g. in declarations:

```
>>    X : T (Y);
```

> That's not an expression and is not resolved (that is, there is no possible overloading).

I see no fundamental difference between "first-class" expressions and type-expressions.

>> Then what is wrong with indexing?

> Nothing is "wrong" with it, it is just redundant. As others have noted here, both indexes and function calls represent a mapping. What's the point of having two ways to represent a mapping? In an Ada-like language, there's no syntax nor semantic difference.

Both are mappings, but unless you make functions first-class citizens there exist language level differences between a function and a container object.

> Ada (and most other languages) are full of redundant stuff. Simplify the basics and then one has more room for interesting stuff (static analysis, parallel execution, etc.).

Yes, but I would rather keep all this stuff in the language making it overridable primitive operations.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Wed, 23 Dec 2020 22:06:03 -0600

> Hmm, where is the operation? A prefixed view is

> <expression>.<operation>(...)

> Indexing is

> <expression>(...)

I neglected to mention that what Ada calls objects are also function calls in this proposed generalization. (Much like enumeration literals are in Ada.) So for static semantics (that is, compile-time), pretty much everything is a function call. This gets rid of the anomalies associated with constants (which don't overload and thus hide more than a parameterless function - which is otherwise the same thing); combined with variable-returning functions, everything is overloadable and treated the same in expressions. Almost no special cases (operators still require some special casing, but we can make them always visible which would eliminate more issues).

Clearly a compiler for this language (which can't be Ada, unfortunately, way too incompatible) would special-case some kinds of built-in functions for things like objects and indexing. But that doesn't need to hair up the semantic model, just the implementations.

>> Ada (and most other languages) are full of redundant stuff. Simplify the basics and then one has more room for

interesting stuff (static analysis, parallel execution, etc.).

> Yes, but I would rather keep all this stuff in the language making it overridable primitive operations.

Yeah, you don't plan to formally describe nor implement this language, so you don't really care about how complex it gets. :-) Well, at least not until performance suffers. Ada is reaching the limit of what can be done without substantial incompatibility. If we're going to allow that, we need to start with a cleaner base, and part of that is getting rid of redundancies.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Thu, 24 Dec 2020 10:37:10 +0100

> I neglected to mention that what Ada calls objects are also function calls in this proposed generalization.

Well, you must stop the recursion somewhere. It is fine to treat access to objects as calls, e.g. to getter/setter, or to indexation, or to dereferencing, but you must finish at some point with something spelled as a call to a subprogram. In the case of a subprogram call you are already there. With "objects" you need a few hops to get there.

[...]

> Ada is reaching the limit of what can be done without substantial incompatibility. If we're going to allow that, we need to start with a cleaner base, and part of that is getting rid of redundancies.

We see that differently. So far new features were added on top which naturally leads to the mess we observe. The problem is lack of generalization not inconsistency. If the new Ada cannot express the old messy, but consistent Ada, then this new Ada is not general enough and it will arrive at the same amount of mess sooner or later.

Getting Integers from Strings

From: John Perry <john.perry@usm.edu>
Subject: Help parsing the language manual on Get'ing integers from Strings
Date: Sun, 20 Dec 2020 16:11:43 -0800
Newsgroups: comp.lang.ada

Sorry if the subject is unclear. I recently tried to use

```
Get(S, Value, Last);
```

...in a program where Value was a Natural and S has the value "29: 116 82 | 119 24". GNAT gave me a Data_Error.

I don't understand why. [...]

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Mon, 21 Dec 2020 09:44:30 +0200

[...]

It seems that the Get procedure understands ':' as a base indicator, as in

"12#44#" works, Value = 52, Last = 6.

"12#44" fails with Data_Error.

[...]

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Mon, 21 Dec 2020 08:57:36 +0100

[...]

Colon: is a replacement character for # (see allowable replacements of characters). So it might think of 29: 116 as a malformed base-29 number with wrong base and missing closing:.

[...]

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Mon, 21 Dec 2020 10:06:47 +0200

I see, an "obsolescent feature" in RM J.2. I learn something new every day (I hope).

Ok, so no bug.

From: Jeffrey R. Carter
Date: Mon, 21 Dec 2020 10:40:17 +0100

> I see, an "obsolescent feature" in RM J.2.

Yes. I never worked with a system that required such substitutions, even in 1984 when it was not an obsolescent feature, but as we can see, it's important to be aware of them.

These days they are sometimes used for obfuscation.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Mon, 21 Dec 2020 19:11:32 -0600

> Yes. I never worked with a system that required such substitutions, even in 1984 when it was not an obsolescent feature, but as we can see, it's important to be aware of them.

I believe that restriction had to do with certain keypunches. But hardly anyone used keypunches even in 1981. (The Unisaur computer that our CS compiler-construction class used still had a few keypunches, but they had mostly transitioned to terminals by that time. I think that was the last class to use the Unisaur; they just had installed some VAX 780s for research and they soon got some for student use as well. My first few programming classes at UW used the Unisaurs keypunches.) I think that requirement was obsolete by the time Ada was completed (it probably wasn't when the Ada design was started).

From: Randy Brukardt <randy@rrsoftware.com>
Date: Mon, 21 Dec 2020 19:19:44 -0600

> Perhaps RM-A.10.8(8) should be clarified/corrected.

For what it's worth, we once tried to do that, but couldn't come to an agreement on precisely what to change the wording to. As a change is not critical, we didn't make one. The ACATS has long had tests in this area that require something subtly different than the wording requires, and it didn't make any sense to change them (since presumably all implementers are passing them, rather than strictly following the RM wording).

In any case, the ":" replacement trips up people from time-to-time, as pretty much no one remembers it. I recall we had to change some piece of new syntax because the possibility of a colon in a number made it ambiguous.

On the Future of the Distributed Systems Annex

From: Rod Kay <rodakay5@gmail.com>
Subject: 2dsa | !2dsa?
Date: Tue, 22 Dec 2020 12:00:48 -0800
Newsgroups: comp.lang.ada

I've heard that the Distributed Systems Annex (DSA) may be dropped from the Ada standard soon. Can anyone confirm this?

I've been using the PolyORB implementation of DSA for some time and find it very useful. The way in which it abstracts away socket 'plumbing' details makes for very simple/understandable comms.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Tue, 22 Dec 2020 19:32:37 -0600

> I've heard that the Distributed Systems Annex (DSA) may be dropped from the Ada standard soon. Can anyone confirm this?

Annex E remains in the proposed Ada 202x standard.

Compiler support, of course, is up to vendors. Dunno if anyone is still supporting it.

> I've been using the PolyORB implementation of DSA for some time and find it very useful. The way in which it abstracts away socket 'plumbing' details makes for very simple/understandable comms.

That was the promise, not sure it ever really was realized. Since the Annex was weakened enough that third-party support isn't really possible anymore (necessary to allow it to be used with current middleware), it's really a vendor-specific thing these days.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Wed, 23 Dec 2020 09:44:26 +0100

> Compiler support, of course, is up to vendors. Dunno if anyone is still supporting it.

It should be moved to the user level. As specified in the Annex there seems no obvious way to provide a user-defined transport for DSA, and there seems no way to have different implementations of DSA in the same program.

[...]

> [...] it's really a vendor-specific thing these days.

Yes, I always wished to include DSA support based on various communication protocols I have implemented in Ada, rather than plain sockets. E.g. I have a ready-to-go DSA implementation for interprocess communication over shared memory, but no idea how to make GNAT aware of it. Or AQMP and ASN.1 look like a straightforward candidate as a DSA transport as they have detailed type description systems to map Ada objects.

From: Maxim Reznik <reznikmm@gmail.com>
Date: Thu, 24 Dec 2020 04:02:24 -0800

I forked an older (Garlic) GNAT DSA implementation and found it quite hackable. :)

My idea is to implement a WebSocket/WebRTC transport and compile it by GNAT-LLVM to WebAssembly to have distributed Ada applications in a browser. I have a working proof of concept demo already :)

https://github.com/reznikmm/garlic/tree/web_socket

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Thu, 24 Dec 2020 14:30:54 +0100

> I forked an older (Garlic) GNAT DSA implementation and found it quite hackable. :)

My question is how to proceed without GLADE/Garlic etc. I have DSA implemented, e.g. System.RPC. I need GNAT to accept it as such.

In a more distant perspective I need a work-around of stream attributes. They are non-portable, so there must be an option to replace them for DSA and/or provide a non-stream parameter marshaling when the transport is a higher-level protocol, e.g. CANopen, EtherCAT, ASN.1, AMQP etc. For these you must map Ada types into the types defined by the protocol. Without this DSA is pretty much pointless.

From: Rod Kay <rodakay5@gmail.com>
Date: Sun, 27 Dec 2020 11:34:10 -0800

Is it likely that the ARG might address the aforementioned issues?

From: Randy Brukardt <randy@rrsoftware.com>
Date: Mon, 28 Dec 2020 17:41:39 -0600

>Is it likely that the ARG might address the aforementioned issues?

As of now, it doesn't appear that there would be any point. Annex E is an optional annex, and so far as we're aware, no compiler vendor has any plans for increasing support for that annex. So the ARG could change the annex but it seems unlikely that any changes would make it into implementations. (We've been told not to expect even the implementation of bugs fixes included in Ada 202x, even from the vendor that originally requested the bug fixes.)

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Tue, 29 Dec 2020 15:56:35 +0100

>> Is it likely that the ARG might address the aforementioned issues?

> As of now, it doesn't appear that there would be any point.

Why should there be any vendor support in the first place? Why not to redefine it as a set of abstract tagged types allowing custom user implementations like storage pools and streams do?

The idea of having an IDL, statically assigned partitions, linking everything together before start is not the way the distributed systems are designed and work today. CORBA died for a reason.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Tue, 29 Dec 2020 16:51:19 +0100

> Would the compiler still need any support for this or would it just be a set of interfaces at library level?

Yes, because the idea is to have remote objects and remote calls looking exactly the same as local objects and local calls.

So the compiler must translate a call to an RPC to a call to some user primitive operation like System.RPC does. The operation would have a controlling parameter "connection" or "remote partition". The actual input values of the original call must be marshaled, e.g. as an output stream. The output values and the result will be returned via an input stream and deserialized from there into the actual parameters/result or else into a remote exception occurrence to re-raise locally if that was the outcome.

Here lie a lot of problems. First is non-portability of stream attributes. Second is lack of support for bulky transfers and multiplexing. It is highly desirable that the output stream could be written in chunks as well as reading the input stream. E.g. if you pass large objects or if you want to multiplex RPCs made from different tasks rather than interlock them (which for synchronous RPC would result in catastrophic performance).

The current Annex E is very crude to allow efficient, low-latency, real-time implementations.

P.S. If Ada supported delegation, introspection and getter/setter interface, then, probably, all remote call/objects stuff could be made at the library level. But for now, compiler magic is needed.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Thu, 31 Dec 2020 17:43:39 -0600

> Why should there be any vendor support in the first place? Why not to redefine it as a set of abstract tagged types allowing custom user implementations like storage pools and streams do?

Marshalling/unmarshalling surely require vendor support, and there has to be a standard interface for the marshalling stuff to talk to. That to me was always the value of Annex E. My understanding is that there is not much interest in doing any work at all, even to correct the mistakes in the existing definitions.

In any case, Storage_Pools and Streams are some of the most expensive features of Ada to support. That's not a model for "lightweight" support of anything.

My advice would be to talk to your vendor if you feel strongly about this sort of support.

Easiest Way to Use Regular Expressions?

From: reinert <reinkor@gmail.com>
Subject: Easiest way to use redular expressions?
Date: Sun, 27 Dec 2020 00:20:11 -0800
Newsgroups: comp.lang.ada

I made the following hack to match a string with a regular expression (using a named pipe and grep under linux):

[Omitted example of spawning a process and capturing the output. —arm]

OK, I assume it somehow breaks the philosophy on Ada and security/reliability. Could someone therefore show a better and more simple way to do this? gnat.expect?

From: J-P. Rosen <rosen@adalog.fr>
Date: Sun, 27 Dec 2020 09:36:49 +0100

AdaControl uses Gnat.Regpat, and is quite happy with it...

From: Emmanuel Briot <briot.emmanuel@gmail.com>
Date: Sun, 27 Dec 2020 03:14:47 -0800

> AdaControl uses Gnat.Regpat, and is quite happy with it...

GNAT.Regpat is a package I wrote 18 years ago or so (time flies..), basically manually translating C code from the Perl implementation of regular expressions. Nowadays, I think it would be better to write a small binding to the pcre library (which has quite a simple API, so the binding should not be too hard). This will provide much better performance, support for unicode, and a host of regexp features that are not supported by GNAT.Regpat.

Never did that while I was working for AdaCore because we would have ended up with too many regexp packages (there is also GNAT.Regexp, which is very efficient but limited in features because it is based on a definite state machine).

I think libpcre might even be distributed with gcc nowadays, although I did not double-check so might be wrong.

This binding would be a nice small project for someone who wants to get started with writing Ada bindings

From: Maxim Reznik <reznikmm@gmail.com>
Date: Mon, 28 Dec 2020 05:58:06 -0800

The Matreshka library has rather advanced regexp engine with full Unicode support

https://forge.ada-ru.org/matreshka/wiki/League/Regexp

From: Jeffrey R. Carter
Date: Mon, 28 Dec 2020 22:07:24 +0100

You can use PragmARC.Matching.Regular_Expression or its instantiation for Character and String, PragmARC.Matching.Character_Regular_Expression

https://github.com/jrcarter/PragmARC/tree/Ada-12

Renames Usage

From: DrPi <314@drpi.fr>
Subject: renames usage
Date: Thu, 31 Dec 2020 12:48:25 +0100
Newsgroups: comp.lang.ada

One can read here https://github.com/AdaCore/svd2ada/blob/master/src/descriptors-field.adb#L83 this line:

```
Tag   : String renames Elements.Get_Tag_Name (Child);
```

Is it equivalent to the following line?

```
Tag: String:= Elements.Get_Tag_Name (Child);
```

From: John Perry <john.perry@usm.edu>
Date: Thu, 31 Dec 2020 04:10:21 -0800

No. Assignment copies the object, and changes to the copy don't affect the original, while renaming obtains a reference to the object. [...]

From: Gautier write-only address <gautier_niouzes@hotmail.com>
Date: Thu, 31 Dec 2020 04:33:30 -0800

> Tag : String renames Elements.Get_Tag_Name (Child);

> Is it equivalent to the following line ?

> Tag : String := Elements.Get_Tag_Name (Child);

Since the temporary object containing the result of the call "Elements.Get_Tag_Name (Child)" is not accessible anywhere else, the effect is the same.

But, perhaps in some implementations, the "renames" accesses that temporary object, which means the memory containing it must not be freed until Tag is out of scope. Perhaps it is even required. Any compiler specialist here?

From: Jeffrey R. Carter
Date: Thu, 31 Dec 2020 15:49:04 +0100

> Tag : String renames Elements.Get_Tag_Name (Child);

> Is it equivalent to the following line ?

> Tag : String := Elements.Get_Tag_Name (Child);

No. A function result is a constant, so the 1st version gives you a constant. The second gives you a variable with the same value.

From: DrPi <314@drpi.fr>
Date: Thu, 31 Dec 2020 16:55:34 +0100

> No. A function result is a constant, so the 1st version gives you a constant. The second gives you a variable with the same value.

Good to know.

What disturbed me was the function call associated with "renames".

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Thu, 31 Dec 2020 19:48:43 +0100

> What disturbed me was the function call associated with "renames".

Renaming a call to a function does not rename it in some functional-programming manner. It renames only the result of.

So if you do

```
X : Float renames Random (Seed);
Y : array (1..10) of Float := (others => X);
```

That would not give you ten pseudo-random numbers. But this will:

```
Z : array (1..10) of Float := (others => Random (Seed));
```

Obituary

Tragic News about Vinzent Hoefler

From: Dirk Craeynest <dirk@orka.cs.kuleuven.be>
Subject: Tragic news about Vinzent Hoefler
Date: Wed, 28 Oct 2020 11:09:09 -0000
Newsgroups: comp.lang.ada

Dear all,

Many of you may know Vinzent Höfler.

I am sad to pass the most tragic news that Vinzent died last Wednesday 21 October...

Below is the message Vinzent's wife Katja Saranen asked me earlier today to share with the Ada community.

He was active in various forums and newsgroups as Vinzent aka "Jellix" aka "JeLlyFish.software@gmx.net" aka "ada.rocks@jlfencey.com" aka "vinzent@heisenbug.eu". He worked on professional as well as open-source Ada projects, was a member and participated in events of ACM SIGAda, Ada-Europe and Ada-Belgium, and helped with several recent Ada DevRooms at FOSDEM events.

I first met Vinzent what seems an eternity ago at the SIGAda 2002 conference in Houston. Our paths crossed many times since, until five years ago he became an "Ada" colleague at Eurocontrol.

I will miss Vinzent, as a colleague, as a like-minded spirit on various issues, and most of all as an intelligent human being. I will miss our interesting discussions: we had too few...

Dirk

Dirk.Craeynest@cs.kuleuven.be (for Ada-Belgium/Ada-Europe/SIGAda/WG9)

Message from Katja Saranen, Wed Oct 28 2020:

**

With the deepest sorrow I have to share with you a devastating tragedy.

Our beloved Vinzent has left this world, he is not with us anymore.

Vinzent "Jellix" Saranen (Höfler, Fritzsche)

09.01.1974 - 21.10.2020

Unspeakable loss for so many. A father, son, brother, grandfather, husband, friend, colleague and much more.

The love of my life. My soulmate. My person. My husband. My safe place. Incredible, wonderful, beautiful, weird, intelligent, talented. So special in so many ways.

We were supposed to grow old together and move to wilderness. I was not supposed to outlive you. I was not supposed to face the world without you. I don't know yet how am I even expected to keep going without you on my side.

This is not a farewell. You're not gone. Love is not any less. You're always with me until we meet again. Love you, forever.

starlingc/katja

"Death is that state in which one exists only in the memory of others, which is why it is not an end. No goodbyes. Just good memories. Hailing frequencies closed, sir."

(Star Trek TNG; Tasha Yar)

There will be no funeral or grave. He has been cremated yesterday and next summer I will take his ashes to the place where he was happiest and where he wanted to go to grow old. For a place to remember him, you can go to nature anywhere and you'll always be close. Memorial(s) will be planned at later time, I am not able to now.

From: Shark8 <onewingedshark@gmail.com>
Date: Wed, 28 Oct 2020 07:24:32 -0700

Tragic news indeed.

Sorry to see him go.

From: Anh Vo <anhvofrcaus@gmail.com>
Date: Wed, 28 Oct 2020 11:35:56 -0700

> Tragic news indeed.

> Sorry to see him go.

Rest in peace. Sincere condolences.

Conference Calendar

Dirk Craeynest

KU Leuven, Belgium. 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.

The COVID-19 pandemic had a catastrophic impact on conferences world-wide. Where available, the status of events is indicated with the following markers: "(c)" = event is cancelled, "(v)" = event is held online, and "(h)" = event is held in a hybrid form (i.e. partially online).

2021

January 18-20 (v)
16th **International Conference on High Performance and Embedded Architecture and Compilation** (HiPEAC'2021). Budapest, Hungary. Topics include: computer architecture, programming models, compilers and operating systems for embedded and general-purpose systems.

January 19-21 (c)
13th **Software Quality Days** (SWQD'2021), Vienna, Austria. Topics include: improvement of software development methods and processes, testing and quality assurance of software and software-intensive systems, domain specific quality issues (such as embedded, medical, automotive systems), novel trends in software quality, etc.

☺ January 17-22 (v)
48th ACM SIGPLAN **Symposium on Principles of Programming Languages** (POPL'2021), Copenhagen, Denmark. Topics include: all aspects of programming languages and programming systems.

January 25-29 (v)
47th **International Conference on Current Trends in Theory and Practice of Computer Science** (SOFSEM'2021), Bozen-Bolzano, Italy. Topics include: methods and tools for improved software processes, software architecture of complex software-intensive systems, model-based software engineering methods and tools, methods and tools for software engineering applications, empirical software engineering, trustworthiness and qualities of modern software systems, etc.

February 01-05 (v)
19th **Australasian Symposium on Parallel and Distributed Computing** (AusPDC'2021), Australia. Topics include: all areas of parallel and distributed computing; multi-core systems; GPUs and other forms of special purpose processors; middleware and tools; parallel programming models, languages and compilers; runtime systems; resource scheduling and load balancing; reliability, security, privacy and dependability; etc.

☺ February 06-07 (v)
Free and Open source Software Developers' European Meeting (FOSDEM'2021), Brussels, Belgium. FOSDEM 2021 is a two-day event (Sat-Sun 06-07 Feb), exceptionally held fully online. After the 10th Ada DevRoom last year there won't be an Ada DevRoom in 2021, but there will be some Ada-related content after all.

☺ Feb 27 - Mar 03 (v)
26th ACM SIGPLAN **Symposium on Principles and Practice of Parallel Programming** (PPoPP'2021). Seoul, South Korea.

Feb 27 - Mar 03 (v)
30th ACM SIGPLAN 2021 **International Conference on Compiler Construction** (CC'2021), Seoul, South Korea. Co-located with CGO, HPCA, and PPoPP. Topics include: processing programs in the most general sense (analyzing, transforming or executing input that describes how a system operates, including traditional compiler construction as a special case); compilation and interpretation techniques (including program representation, analysis, and transformation; code generation, optimization, and synthesis; the verification thereof); run-time techniques (including memory management, virtual machines, and dynamic and just-in-time compilation); programming tools (including refactoring editors, checkers, verifiers, compilers, debuggers, and profilers); techniques, ranging from programming languages to micro-architectural support, for specific domains such as secure, parallel, distributed, embedded or mobile environments; design and implementation of novel language

constructs, programming models, and domain-specific languages. Deadline for submissions: January 5, 2021 (artifacts).

March 04-06 (h) — 25th **International Conference on Engineering of Complex Computer Systems** (ICECCS'2020), Singapore. ICECCS'2020 was postponed from 28-31 October 2020 to 4-6 March 2021, and later moved to a hybrid format. Topics include: all areas related to complex computer-based systems, including the causes of complexity and means of avoiding, controlling, or coping with complexity, such as verification and validation, security and privacy of complex systems, model-driven development, reverse engineering and refactoring, software architecture, design by contract, agile methods, safety-critical and fault-tolerant architectures, real-time and embedded systems, systems of systems, cyber-physical systems and Internet of Things (IoT), tools and tool integration, industrial case studies, etc.

March 09-12 (v) — 28th IEEE **Conference on Software Analysis, Evolution, and Reengineering** (SANER'2021), Honolulu, Hawaii, USA. Topics include: theory and practice of recovering information from existing software and systems; software analysis, parsing, and fact extraction of multi-language systems; mining software repositories; empirical studies in software re-engineering, maintenance, and evolution; software architecture evolution; software maintenance and re-engineering economics; software release engineering, continuous integration and delivery; evaluation and assessment of reverse engineering and re-engineering tools; software analysis and comprehension; education related issues; etc.

March 10-12 (v) — 29th **Euromicro International Conference on Parallel, Distributed and Network-Based Processing** (PDP'2021), Valladolid, Spain.

March 22-26 (v) — IEEE **International Conference on Software Architecture** (ICSA'2021), Stuttgart, Germany. Topics include: model driven engineering for continuous architecting; component based software engineering; architecture evaluation and quality aspects of software architectures; automatic extraction and generation of software architecture descriptions; refactoring and evolving architecture design decisions and solutions; architecture frameworks and architecture description languages; linking architecture to requirements and/or implementation; architecture conformance; reusable architectural solutions; software architecture for legacy systems and systems integration; architecting families of products; software architect roles and responsibilities; training, education, and certification of software architects; industrial experiments and case studies; etc. Deadline for submissions: January 4, 2021 (Early Career Researchers Forum abstracts), January 11, 2021 (Early Career Researchers Forum papers), January 20, 2021 (workshops).

☺ Mar 22-26 (v) — **International Conference on the Art, Science, and Engineering of Programming** (Programming'2021), Cambridge, UK

- Mar 22 (v) — **5th International Workshop on Programming Technology for the Future Web** (ProWeb'2021). Topics include: programming technology and formalisms for implementing web applications and for maintaining their quality, as well as experience reports about their usage; such as on web app quality (static and dynamic program analyses, development tools, automated testing, contract systems, type systems, migration from legacy architectures, API conformance checking, ...); designing for and hosting novel languages on the web; new languages and runtimes; security on the new web; surveys and case studies using state-of-the-art web technology; ideas on and experience reports about how to reconcile the need for quality with the need for agility on the web, how to master and combine the myriad of tier-specific technologies required to develop a web application; etc. Deadline for submissions: February 8, 2021.

March 22-26 (v) — 36th ACM **Symposium on Applied Computing** (SAC'2021), Gwangju, Korea.

- Mar 22-26 (v) — **Software Verification and Testing Track** (SVT'2021). 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, model-based testing, software testing, static and dynamic analysis, abstract interpretation, analysis methods for dependable systems, software certification and proof carrying code, fault diagnosis and debugging, verification and validation of large scale software systems, real world applications and case studies applying software testing and verification,

etc.

Mar 22-26 (v) — **Embedded Systems Track** (EMBS'2021). Topics include: the application of both novel and well-known techniques to the embedded systems development.

☺ Mar 22-26 (v) — **Track on Programming Languages** (PL'2021). Topics include: technical ideas and experiences relating to implementation and application of programming languages, such as 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, etc.

Mar 22-26 (v) — **16th Track on Dependable, Adaptive, and Secure Distributed Systems** (DADS'2021). Topics include: Dependable, Adaptive, and secure Distributed Systems (DADS); modeling, design, and engineering of DADS; foundations and formal methods for DADS; etc.

March 27 (v) — IEEE **International Conference on Code Quality** (ICCQ'2021), Moscow, Russia. Topics include: static analysis, program verification, bug detection, and software maintenance.

Mar 27 - Apr 01 (v) — 24th **European Joint Conferences on Theory and Practice of Software** (ETAPS'2021), Luxembourg, Luxembourg. Events include: ESOP (European Symposium on Programming), FASE (Fundamental Approaches to Software Engineering), FoSSaCS (Foundations of Software Science and Computation Structures), TACAS (Tools and Algorithms for the Construction and Analysis of Systems), SV-COMP (the 10th Competition on Software Verification). Deadline for submissions: January 17, 2021 (nominations ETAPS doctoral dissertation award).

☺ April 07-09 (v) — 29th **International Conference on Real-Time Networks and Systems** (RTNS'2021), Nantes, France. Topics include: real-time applications design and evaluation (automotive, avionics, space, railways, telecommunications, process control, multimedia), real-time aspects of emerging smart systems (cyber-physical systems and emerging applications, ...), real-time system design and analysis (real-time tasks modeling, task/message scheduling, mixed-criticality systems, Worst-Case Execution Time (WCET) analysis, security, ...), software technologies for real-time systems (model-driven engineering, programming languages, compilers, WCET-aware compilation and parallelization strategies, middleware, Real-time Operating Systems (RTOS), ...), formal specification and verification, real-time distributed systems, etc.

April 12-15 (v) — 27th **International Working Conference on Requirements Engineering: Foundation for Software Quality** (REFSQ'2021), Essen, Germany. Deadline for submissions: February 8, 2021 (OpenRE track, posters, tools, workshop papers), February 15, 2021 (Doctoral Symposium), March 15, 2021 (Graduate Students' event).

April 12-16 (v) — 14th IEEE **International Conference on Software Testing, Verification and Validation** (ICST'2021), Porto de Galinhas, Brazil. Topics include: manual testing practices and techniques, security testing, model based testing, test automation, static analysis and symbolic execution, formal verification and model checking, software reliability, testability and design, testing and development processes, testing in specific domains (such as embedded, concurrent, distributed, ..., and real-time systems), testing/debugging tools, empirical studies, experience reports, etc. Deadline for submissions: January 10, 2021 (workshop papers).

April 19-23 (v) — 12th ACM/SPEC **International Conference on Performance Engineering** (ICPE'2021), Rennes, France. Deadline for submissions: January 20, 2021 (posters, demos, tutorials, work-in-progress papers).

May 11-13 (v) — ACM **International Conference on Computing Frontiers 2021** (CF'2021), Catania, Sicily, Italy. Topics include: embedded, IoT, and Cyber-Physical Systems; large-scale system design and networking; system software, compiler technologies, and programming languages; fault tolerance and resilience (solutions for ultra-large and safety-critical systems, e.g. infrastructure, airlines; hardware and software approaches in adverse environments such as space); security (methods, system support, and hardware for protecting against malicious code; ...); etc. Deadline for submissions: January 28, 2021 (abstracts), February 4, 2021 (papers).

May 18-21 (v) 14th **Cyber-Physical Systems and Internet of Things Week** (CPS Week'2021), Nashville, Tennessee, USA. Event includes: 5 top conferences, HSCC, ICCPS, IPSN, RTAS, and IoTDI, multiple workshops, tutorials, competitions and various exhibitions from both industry and academia. Deadline for submissions: Deadline for submissions: February 3 - March 18, 2021 (workshop papers).

☺ May 18-21 27th IEEE **Real-Time and Embedded Technology and Applications Symposium** (RTAS'2021). Topics include: systems research related to embedded systems and time-sensitive systems, ranging from traditional hard real-time systems to embedded systems without explicit timing requirements; papers describing original systems, applications, case studies, methodologies, and algorithms that contribute to the state of practice in design, implementation, verification, and validation of embedded systems or time-sensitive systems.

May 19-21 (h) 9th **International Conference on Fundamentals of Software Engineering** (FSEN'2021), 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 systems; cyber-physical software systems; model checking and theorem proving; software verification; CASE tools and tool integration; industrial applications; etc.

May 23-29 (v) 43rd **International Conference on Software Engineering** (CSE'2021), Madrid, Spain. Topics include: the full spectrum of Software Engineering, such as testing and analysis (software testing, program analysis, validation and verification, fault localization, formal methods, programming languages), empirical software engineering (mining software repositories, software ecosystems, ...), software evolution (evolution and maintenance, debugging, program comprehension, API design and evolution, configuration management, release engineering and DevOps, software reuse, refactoring, reverse engineering, ...), social aspects of software engineering (human aspects of software engineering, agile methods and software processes, software economics, ethics in software engineering, ...), requirements, modeling, and design (requirements engineering, modeling and model-driven engineering, software architecture and design, tools and environments, variability and product lines, parallel, distributed, and concurrent systems, ...), dependability (software security, privacy, reliability and safety, performance, embedded / cyber-physical systems, ...), etc. Deadline for submissions: February 22, 2021 (workshop papers), February 1, 2021 (student volunteers).

May 17-19 (v) 9th **International Conference on Formal Methods in Software Engineering** (FormaliSE'2021). Topics include: approaches and tools for verification and validation; application of formal methods to specific domains, e.g., autonomous, cyber-physical, and IoT systems; scalability of formal methods applications; integration of formal methods within the software development lifecycle; model-based software engineering approaches; formal methods in a certification context; formal approaches for safety and security-related issues; usability of formal methods; guidelines to use formal methods in practice; case studies developed/analyzed with formal approaches; experience reports on the application of formal methods to real-world problems; etc. Deadline for submissions: January 5, 2021 (abstracts), January 12, 2021 (papers).

May 19-21 (v) 4th **International Conference on Technical Debt** (TechDebt'2021). Topics include: technical debt management and decision making; tools and indicators for identifying technical debt; technical debt remediation strategies, methods, and tools; experiences, approaches and tools for teaching technical debt topics in academic courses or industrial training; etc. Deadline for submissions: January 12, 2021 (research and experience papers), January 27, 2021 (short papers).

May 24-28 (v) 13th **NASA Formal Methods Symposium** (NFM'2021), Norfolk, Virginia, USA. Topics include: challenges and solutions for achieving assurance for critical systems; formal verification, model checking, and static analysis techniques; theorem proving; techniques and algorithms for scaling formal methods; design for verification and correct-by-design techniques; experience report of application of formal methods in industry; use of formal methods in education; applications of formal methods in the development of autonomous systems, safety-critical systems, concurrent and distributed systems, cyber-physical, embedded, and hybrid systems, ...; etc.

☺ June 01-03 (h)
24th IEEE **International Symposium On Real-Time Distributed Computing** (ISORC'2021), Daegu, South Korea. Topics include: all aspects of object/component/service-oriented real-time distributed computing (ORC) technology; real-time distributed computing; Internet of Things (IoT); real-time scheduling theory; resilient cyber-physical systems; autonomous systems (e.g., autonomous driving); optimization of time-sensitive applications; applications based on ORC technology, for example, medical devices, intelligent transportation systems, industrial automation systems and industry 4.0, smart grids, ...; etc. Deadline for submissions: February 5, 2021 (main track), April 5, 2021 (posters, demos).

♦ June 07-11 (v)
25th **Ada-Europe International Conference on Reliable Software Technologies** (AEiC 2021 aka Ada-Europe 2021). Santander, Spain. AEiC'2020 was postponed from 8-12 June 2020 to 7-11 June 2021, then moved to a hybrid and later to a full virtual event format. Sponsored by Ada-Europe, in cooperation with ACM SIGAda, SIGPLAN, SIGBED, and the Ada Resource Association (ARA). Deadline for submissions: January 14, 2021 (journal-track papers, workshop proposals), 31 March 2021 (Work-in-Progress papers, industrial presentation outlines, tutorial and invited presentation proposals).

June 21-23 (v)
25th **International Conference on Evaluation and Assessment in Software Engineering** (EASE'2021), Trondheim, Norway. Topics include: assessing the benefits/costs associated with using chosen development technologies; empirical studies using qualitative, quantitative, and mixed methods; evaluation and comparison of techniques and models; replication of empirical studies and families of studies; software technology transfer to industry; etc. Deadline for submissions: January 22, 2021 (workshops), February 19, 2021 (Full Research Track abstracts), February 26, 2021 (Full Research Track papers, Vision and Emerging Results Track papers), March 26, 2021 (PhD symposium papers, workshop papers).

June 21-25 (v)
Software Technologies: Applications and Foundations (STAF'2021). Bergen, Norway. STAF'2020 was postponed from 22-26 June 2020 to 21-25 June 2021.

June 21-25 (v)
15th **International Conference on Tests And Proofs** (TAP'2021). 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 supporting the automated generation of test vectors and oracles, and supporting novel definitions of coverage criteria; program analysis techniques combining static and dynamic analysis; testing and runtime analysis of formal specifications; verification of verification tools and environments; applications of test and proof techniques in new domains, such as security, configuration management, learning; combined approaches of test and proof in the context of formal certifications (Common Criteria, CENELEC, ...); case studies, tool and framework descriptions, and experience reports about combining tests and proofs; etc. Deadline for submissions: March 1, 2021 (abstracts), March 8, 2021 (full papers).

June 28 - July 02
15th ACM **International Conference on Distributed Event-Based Systems** (DEBS'2021), Milan, Italy. Topics include: systems dealing with collecting, detecting, processing and responding to events through distributed middleware and applications; models, architectures and paradigms (trustworthy event-based systems, real-time analytics, ...); systems and software (distributed programming, security, reliability and resilience, ...); applications (Internet-of-Things, cyber-physical systems, healthcare and logistics, ...); etc. Deadline for submissions: February 26, 2021 (abstracts), March 5, 2021 (papers).

July 07-09 (v)
33rd **Euromicro Conference on Real-Time Systems** (ECRTS'2021), Modena, Italy. Topics include: all aspects of timing requirements in computer systems; elements of time-sensitive computer systems, such as operating systems, hypervisors, middlewares and frameworks, programming languages and compilers, runtime environments, ...; classic worst-case execution time (WCET) analysis; formal methods for the verification and validation of real-time systems; the interplay of timing predictability and other non-functional qualities such as reliability, security, quality of control, scalability, ...; foundational scheduling and predictability questions, such as schedulability analysis, locking and

non-blocking synchronization protocols, computational complexity, ...; etc. Deadline for submissions: March 3, 2021.

☺ July 12-16 — 35th **European Conference on Object-Oriented Programming** (ECOOP'2021), Aarhus, Denmark. Topics include: design, implementation, optimization, analysis, testing, verification, and theory of programs, programming languages, and programming environments. Deadline for submissions: January 11, 2021 (papers), January 31, 2021 (workshops), February 28, 2021 (Artifact Evaluation Committee nominations).

July 12-16 (v) — 45th **Annual IEEE Conference on Computers, Software and Applications** (COMPSAC'2021), Madrid, Spain. Deadline for submissions: January 15, 2021 (main conference papers), April 1, 2021 (student competition), April 21, 2021 (worskhop papers).

July 12-16 — 1st IEEE **International Workshop on Software Engineering for Industrial Cyber-Physical Systems** (SE4ICPS'2021). Topics include: middleware design for industrial IoT/CPS; software design theory for IoT/CPS; formal Methods for IoT/CPS; safety-critical cyber-physical software systems; software quality attributes of IoT/CPS; fault-tolerant IoT/CPS; testing, validation, verification, simulation, and visualization of IoT/CPS; IoT/CPS engineering Methods and Tools; etc. Deadline for submissions: April 21, 2021 (papers).

☺ August 23-27 (v) — 25th **International Conference on Formal Methods for Industrial Critical Systems** (FMICS'2021), Paris, France. Co-located with CONCUR'2021 and FORMATS'2021. Topics include: case studies and experience reports on industrial applications of formal methods, focusing on lessons learned or identification of new research directions; methods, techniques and tools to support automated analysis, certification, debugging, descriptions, learning, optimisation and transformation of complex, distributed, real-time, embedded, mobile and autonomous systems; verification and validation methods that address shortcomings of existing methods with respect to their industrial applicability (e.g., scalability and usability issues); impact of the adoption of formal methods on the development process and associated costs; application of formal methods in standardisation and industrial forums. Deadline for submissions: May 7, 2021 (abstracts), May 14, 2021 (papers).

☺ Aug 30 - Sep 03 (v) — 27th **International European Conference on Parallel and Distributed Computing** (Euro-Par'2021), Lisbon, Portugal. Topics include: all flavors of parallel and distributed processing, such as compilers, tools and environments, scheduling and load balancing, theory and algorithms for parallel and distributed processing, parallel and distributed programming, interfaces, and languages, multicore and manycore parallelism, etc. Deadline for submissions: February 5, 2021 (abstracts), February 12, 2021 (papers).

September 07-10 (h) — 40th **International Conference on Computer Safety, Reliability and Security** (SafeComp'2021), York, UK. Deadline for submissions: January 25, 2021 (workshops, tutorials), February 1, 2021 (abstracts), February 15, 2021 (full papers).

September 08-11 — 14th **International Conference on the Quality of Information and Communications Technology** (QUATIC'2021), Faro, Portugal. Topics include: all quality aspects in ICT systems engineering and management; quality in ICT process, product and applications domains; practical studies; etc. Tracks on ICT verification and validation, safety, security and privacy, model-driven engineering, quality in cyber-physical systems, software evolution, evidence-based software quality engineering, software quality education and training, etc. Deadline for submissions: March 30, 2021 (full papers), May 25, 2021 (short papers).

September 21-23 — 20th **International Conference on Intelligent Software Methodologies, Tools and Techniques** (SOMET'2021), Cancun, Mexico. Topics include: state-of-art and new trends on software methodologies, tools and techniques; software methodologies, and tools for robust, reliable, non-fragile software design; software developments techniques and legacy systems; software evolution techniques; agile software and lean methods; formal methods for software design; software maintenance; software security tools and techniques; formal techniques for software representation, software testing and validation; software reliability; Model Driven Development (DVD), code centric to model centric software engineering; etc. Deadline for submissions: March 21, 2021 (papers).

October 10-15 (v) — **Embedded Systems Week 2021** (ESWEEK'2021). Shanghai, China. The venues for ESWEEK 2020 and 2021 were swapped. ESWEEK 2020 was to be held in Hamburg, Germany from September 20-25, 2020, and ESWEEK 2021 would be held in Shanghai, China from October 10-15, 2021. Includes

CASES'2021 (International Conference on Compilers, Architectures, and Synthesis for Embedded Systems), CODES+ISSS'2021 (International Conference on Hardware/Software Codesign and System Synthesis), EMSOFT'2021 (International Conference on Embedded Software). Deadline for submissions: April 2, 2021 (journal track abstracts), April 9, 2021 (journal track full papers), April 16, 2021 (workshops), April 30, 2021 (tutorials, special sessions), June 4, 2021 (Work-in-Progress papers).

October 10-15 (v) — ACM SIGBED **International Conference on Embedded Software** (EMSOFT'2021). Topics include: the science, engineering, and technology of embedded software development; research in the design and analysis of software that interacts with physical processes; results on cyber-physical systems, which integrate computation, networking, and physical dynamics. Deadline for submissions: April 2, 2021 (Journal-Track abstracts), April 9, 2021 (Journal-Track full papers), June 4, 2021 (Work-in-Progress submissions).

October 10-15 (v) — **International Conference on Hardware/Software Codesign and System Synthesis** (CODES+ISSS'2021). Topics include: system-level design, hardware/software co-design, modeling, analysis, and implementation of modern Embedded Systems, Cyber-Physical Systems, and Internet-of-Things, from system-level specification and optimization to system synthesis of multi-processor hardware/software implementations. Deadline for submissions: April 2, 2021 (Journal-Track abstracts), April 9, 2021 (Journal-Track full papers), June 4, 2021 (Work-in-Progress submissions).

October 10-15 (v) — **International Conference on Compilers, Architecture, and Synthesis for Embedded Systems** (CASES'2021). Topics include: latest advances in compilers and architectures for high-performance, low-power, and domain-specific embedded systems; compilers for embedded systems: multi- and many-core processors, GPU architectures, reconfigurable computing including FPGAs and CGRAs, security, reliability, and predictability (secure architectures, hardware security, and compilation for software security; architecture and compiler techniques for reliability and aging; modeling, design, analysis, and optimization for timing and predictability; validation, verification, testing & debugging of embedded software); etc. Deadline for submissions: April 2, 2021 (Journal-Track abstracts), April 9, 2021 (Journal-Track full papers), June 4, 2021 (Work-in-Progress submissions).

October 18-22 — 19th **International Symposium on Automated Technology for Verification and Analysis** (ATVA'2021), Gold Coast, Australia. Topics include: theoretical and practical aspects of automated analysis, synthesis, and verification of hardware, software, and machine learning (ML) systems; program analysis and software verification; analytical techniques for safety, security, and dependability; testing and runtime analysis based on verification technology; analysis and verification of parallel and concurrent systems; verification in industrial practice; applications and case studies; automated tool support; etc. Deadline for submissions: April 9, 2021 (full papers).

November 15-19 — 36th IEEE/ACM **International Conference on Automated Software Engineering** (ASE'2021), Melbourne, Australia. Topics include: foundations, techniques, and tools for automating the analysis, design, implementation, testing, and maintenance of large software systems; testing, verification, and validation; software analysis; empirical software engineering; maintenance and evolution; software security and trust; program comprehension; software architecture and design; reverse engineering and re-engineering; model-driven development; specification languages; software product line engineering; etc. Deadline for submissions: March 22, 2021 (workshops), April 16, 2021 (research track abstracts), April 23, 2021 (research papers), June 11, 2021 (tutorials, New Ideas and Emerging Results (NIER) track, Late Breaking Results track, tool demos).

November 20-26 — 24th **International Symposium on Formal Methods** (FM'2021), Beijing, China. Topics include: formal methods in a wide range of domains including software, computer-based systems, systems-of-systems, cyber-physical systems, security, human-computer interaction, manufacturing, sustainability, energy, transport, smart cities, and healthcare; formal methods in practice (industrial applications of formal methods, experience with formal methods in industry, tool usage reports, experiments with challenge problems); tools for formal methods (advances in automated verification, model checking, and testing with formal methods, tools integration, environments for formal methods, and experimental

validation of tools); formal methods in software and systems engineering (development processes with formal methods, usage guidelines for formal methods, and method integration); etc. Deadline for submissions: February 15, 2021 (workshops, tutorials), April 30, 2021 (abstracts), May 6, 2021 (full papers).

☺ Dec 07-10 42nd IEEE **Real-Time Systems Symposium** (RTSS'2021), Taipei, Taiwan. Topics include: addressing some form of real-time requirements such as deadlines, response times or delays/latency; real-time system track (middleware, compilers, tools, scheduling, QoS support, testing and debugging, design and verification, modeling, WCET analysis, performance analysis, fault tolerance, security, system experimentation and deployment experiences, ...); design and application track (cyber-physical systems design methods, tools chains, security and privacy, performance analysis, robustness and safety, analysis techniques and tools, ...; architecture description languages and tools; Internet of Things (IoT) aspects of scalability, interoperability, reliability, security, middleware and programming abstractions, protocols, modelling, analysis and performance evaluation, ...); etc. Deadline for submissions: February 28, 2021 (TCRTS award nominations), May 27, 2021 (papers).

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

Conference Chair
Michael González Harbour
Universidad de Cantabria, Spain
mgh@unican.es

Program Chairs
Mario Aldea Rivas
Universidad de Cantabria, Spain
aldeam@unican.es
J. Javier Gutiérrez García
Universidad de Cantabria, Spain
gutierjj@unican.es

Work-in-Progress Chair
Kristoffer Nyborg Gregertsen
SINTEF Digital, Norway
kristoffer.gregertsen@sintef.no

Tutorial & Workshop Chair
Jorge Garrido Balaguer
Universidad Politécnica de Madrid, Spain
jorge.garrido@upm.es

Industrial Chair
Patricia Balbastre Betoret
Universitat Politècnica de València, Spain
patricia@ai2.upv.es

Exhibition & Sponsorship Chair
Ahlan Marriott
White Elephant GmbH, Switzerland
software@white-elephant.ch

Publicity Chair
Dirk Craeynest
Ada-Belgium & KU Leuven, Belgium
dirk.craeynest@cs.kuleuven.be

General Information

The **25th Ada-Europe International Conference on Reliable Software Technologies (AEiC 2021 aka Ada-Europe 2021)**, initially scheduled to take place in Santander, Spain, will be held online from the 7th to the 11th of June, 2021. The conference schedule includes a technical program, vendor exhibition and parallel tutorials and workshops.

The organizing committee estimates that the conditions for a safe in-person conference will not be met in June 2021. Consequently, the AEiC 2021 Conference will be a **virtual-only event.**

Schedule

14 January 2021	Submission of journal-track papers and workshop proposals
19 March 2021	Notification of journal-track paper presentations and workshops
31 March 2021	Submission of Work-in-Progress (WiP) papers, industrial presentation outlines, and tutorial and invited presentation proposals
30 April 2021	Notification of acceptance for WiP papers, industrial presentation outlines, and tutorial and invited presentations proposals

Topics

The conference is a leading international forum for providers, practitioners and researchers in reliable software technologies. The conference presentations will illustrate current work in the theory and practice of the design, development and maintenance of long-lived, high-quality software systems for a challenging variety of application domains. The program will have keynotes, Q&A sessions and discussions, and virtual social events. Participants include practitioners and researchers from industry, academia and government organizations active in the promotion and development of reliable software technologies.

The topics of interest for the conference include but are not limited to:

- **Design and Implementation of Real-Time and Embedded Systems,**
- **Design and Implementation of Mixed-Criticality Systems,**
- **Theory and Practice of High-Integrity Systems,**
- **Software Architectures for Reliable Systems,**
- **Methods and Techniques for Quality Software Development and Maintenance,**
- **Ada Language and Technologies,**
- **Mainstream and Emerging Applications with Reliability Requirements,**
- **Achieving and Assuring Safety in Machine Learning Systems,**
- **Experience Reports on Reliable System Development,**
- **Experiences with Ada.**

Refer to the conference website for the full list of topics.

Call for Journal-Track Papers

The **journal-track papers** submitted to the conference are full-length papers that must describe mature research work on the conference topics. They must be original and shall undergo anonymous peer review.

Accepted journal-track papers will get a presentation slot within a technical session of the conference and they will be published in an open-access special issue of the **Journal of Systems Architecture** (Q2 in the JCR and SJR ranks) with no additional costs to authors. The corresponding authors shall submit their work by 14 January 2021 via the Special Issue web page: *https://www.journals.elsevier.com/journal-of-systems-architecture/call-for-papers/special-issue-on-reliable-software-technologies-aeic2021*.

Submitted papers must follow the guidelines provided in the "Guide-for-Authors" of the JSA (*https://www.elsevier.com/journals/journal-of-systems-architecture/1383-7621/guide-for-authors*). In particular, JSA does not impose any restriction on the format or extension of the submissions.

Call for WiP-Track Papers

The **Work-in-Progress papers (WiP-track)** are short (4-page) papers describing evolving and early-stage ideas or new research directions. They must be original and shall undergo anonymous peer review. The corresponding authors shall submit their work by 31 March 2021, via *https://easychair.org/conferences/?conf=aeic2021*, strictly in PDF and following the Ada User Journal style (*http://www.ada-europe.org/auj/*).

Authors of accepted WiP-track papers will get a presentation slot within a regular technical session of the conference and will also be requested to present a poster. The papers will be published in the Ada User Journal as part of the proceedings of the Conference.

The conference is listed in the principal citation databases, including DBLP, Scopus, Web of Science, and Google Scholar. The Ada User Journal is indexed by Scopus and by EBSCOhost in the Academic Search Ultimate database.

Call for Industrial Presentations

The conference seeks **industrial presentations** that deliver insightful information value but may not sustain the strictness of the review process required for regular papers. The authors of industrial presentations shall submit their proposals, in the form of a short (one or two pages) abstract, by 31 March 2021, via *https://easychair.org/conferences/?conf=aeic2021*, strictly in PDF and following the Ada User Journal style (*http://www.ada-europe.org/auj/*).

The Industrial Committee will review the submissions anonymously and make recommendations for acceptance. The abstract of the accepted contributions will be included in the conference booklet, and authors will get a presentation slot within a regular technical session of the conference.

These authors will also be invited to expand their contributions into articles for publication in the Ada User Journal, as part of the proceedings of the Industrial Program of the Conference.

Awards

Ada-Europe will offer an honorary **award for the best presentation**. All journal-track and industrial presentations are eligible.

Call for Invited Presentations

The **invited presentations** are intended to allow researchers to present paramount research results that are relevant to the conference attendees. There will be no publication associated to these presentations, which may include previously published works, relevant new tools, methods or techniques.

The invited presentations will be allocated a presentation slot.

The Program Committee will select invited presentation proposals that may be submitted by e-mail to one of the Program Chairs as a one-page summary of the proposed presentation, along with the information and/or links required to show the relevance of the covered topic.

Call for Educational Tutorials

The conference is seeking tutorials in the form of educational seminars including hands-on or practical demonstrations. Proposed tutorials can be from any part of the reliable software domain, they may be purely academic or from an industrial base making use of tools used in current software development environments. We are also interested in contemporary software topics, such as IoT and artificial intelligence and their application to reliability and safety.

Tutorial proposals shall include a title, an abstract, a description of the topic, an outline of the presentation, the proposed duration (half day or full day), and the intended level of the tutorial (introductory, intermediate, or advanced). All proposals should be submitted by e-mail to the Educational Tutorial Chair.

The Ada User Journal will offer space for the publication of summaries of the accepted tutorials.

Call for Workshops

Workshops on themes that fall within the conference scope may be proposed. Proposals may be submitted for half- or full-day events, to be scheduled at either end of the conference days. Workshop proposals should be submitted by e-mail to the Workshop Chair. The workshop organizer shall also commit to producing the proceedings of the event, for publication in the Ada User Journal.

Call for Exhibitors

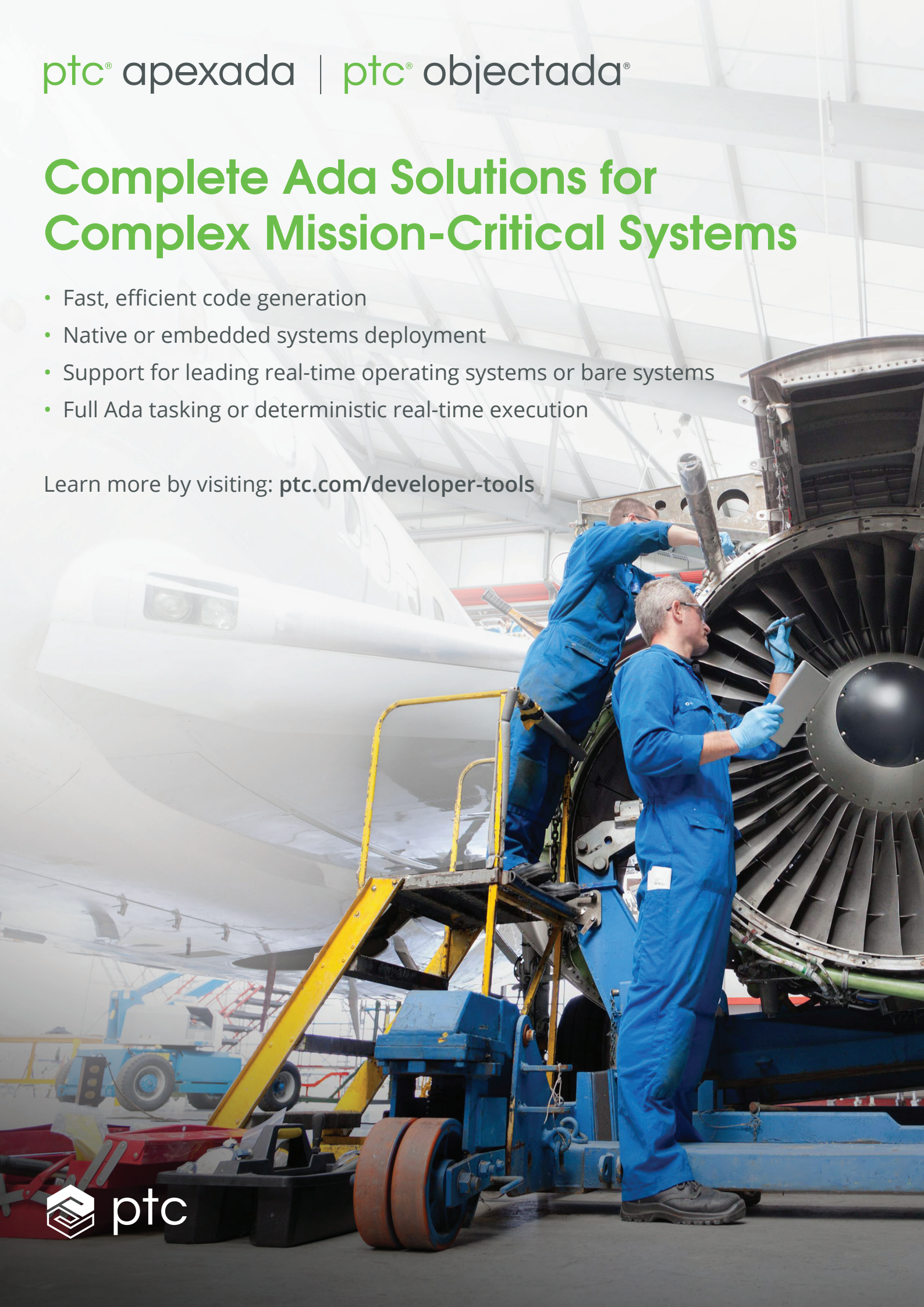
The commercial exhibition will span the core days of the main conference. As an alternative to the traditional physical exhibition, a virtual room will be provided for exhibition activities. Vendors and providers of software products and services should contact the Exhibition Chair for information and for allowing suitable planning of the exhibition space and time.

ptc® apexada | ptc® objectada®

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**



ptc

From Ada to Platinum SPARK: A Case Study

Patrick Rogers, PhD
AdaCore; email: rogers@adacore.com

1 Introduction

An effective approach to learning a new programming language is to implement data structures common to computer programming. The approach is effective because the problem to be solved is well understood, allowing one to focus on the language details. Moreover, several different forms of a given data structure are often possible: bounded versus unbounded, sequential versus thread-safe, and so on. These multiple forms likely require a wide range of language features.

Fortunately, for learning SPARK one need not start from scratch. We can begin with an existing, production-ready Ada implementation for a common data structure and make the changes necessary to conform to SPARK. This approach is possible because SPARK is based directly on Ada, and the component architecture, based on the principles of software engineering, is the same in both languages. In both cases we would have a generic package exporting a private type, with primitive operations manipulating that type. The type might be limited, and might be tagged, using the same criteria to decide. As a result, the changes need not be extensive, although they are important and, in some cases, subtle.

Therefore, we will start with a fundamental reusable component used in real-world applications: a sequential, bounded stack abstract data type (ADT). By "sequential" we mean that the code is not thread-safe. By "bounded" we mean that it is backed by an array, which as usual entails a discriminant on the private type to set the upper bound of the internal array component. Client misuse of the Push and Pop routines, e.g., pushing onto a full stack, raises exceptions. As Ada has evolved new features have been applied to make the code more robust, for example the Push and Pop routines use preconditions to detect clients misusing the abstraction, raising exceptions from within the preconditions in response.

The result will be a completely proven SPARK implementation that relies on static verification of the abstraction's semantics instead of run-time enforcement. We will prove that there are no reads of unassigned variables, no run-time errors (array indexing errors, range errors, numeric overflow errors, etc.), and that subprogram bodies implement their functional requirements, including some requirements not previously identified. For compliant client code, analysis will be able to prove there are no attempts to push onto a full stack, no attempts to pop from an empty stack, and so forth. As a result, we get a maximally robust implementation of a full, complete Stack ADT that allows provably compliant client usage.

We assume familiarity with Ada, including preconditions and postconditions. Basic language details can be obtained from the online learning facilities available at https://learn.adacore.com/, an interactive site allowing one to enter, compile, and execute Ada programs in a web browser. We also assume a minimal degree of familiarity with SPARK. That same web site provides a similar interactive environment and materials for learning SPARK, including formal proof.

2 SPARK Adoption Levels

In 2016, AdaCore collaborated with Thales in a series of experiments on the application of SPARK to existing software projects written in Ada. The resulting document presents a set of guidelines for adopting formal verification in existing projects. These guidelines are arranged in terms of five levels of software assurance, in increasing order of benefits and costs. The levels are named Stone, Bronze, Silver, Gold and Platinum. Successfully reaching a given level requires successfully achieving the goals of the previous levels as well.

The guidelines were developed jointly by AdaCore and Thales for the adoption of the SPARK language technology at Thales but are applicable across a wide range of application domains. The document is available online: http://www.adacore.com/knowledge/technical-papers/implementation-guidance-spark

2.1 Stone Level

The goal at the Stone level is to identify as much code as possible that belongs to the SPARK subset. That subset provides a strong semantic coding standard that enforces safer use of Ada language features and forbids those features precluding analysis (e.g., exception handlers). The result is potentially more understandable, maintainable code.

2.2 Bronze Level

The goal at the Bronze level is to verify initialization and correct data flow, as indicated by the absence of GNATprove messages during SPARK flow analysis. Flow analysis detects programming errors such as reading uninitialized data, problematic aliasing between formal parameters, and data races between concurrent tasks. In addition, GNATprove checks unit specifications for the actual data read or written, and the flow of information from inputs to outputs. As one can see, this level provides significant benefits, and can be reached with comparatively low cost. There are no proofs attempted at this level, only data and flow analyses.

2.3 Silver Level

The goal at the Silver level is to statically prove absence of run-time errors (AoRTE), i.e., that there are no exceptions raised. Proof at this level detects programming errors such as divide by zero, array indexes that are out of bounds, and numeric overflow (integer, fixed-point and floating-point), among others. These errors are detected via the implicit language-defined checks that raise language-defined exceptions. The checks themselves preclude a number of significant situations, including, for example, buffer overflow, which is often exploited to inject malicious executable code.

Preconditions, among other additions, may be required to prove these checks. To illustrate the benefit and part of the cost of achieving the Silver level, consider the way the Ada version of the stack ADT uses preconditions for this purpose. (The complete Ada implementation is explored in section 4.1.) First, here is the full declaration for type Stack in the package private part:

```
type Content is array (Positive range <>) of Element;
type Stack (Capacity : Positive) is record
  Values : Content (1 .. Capacity);
  Top    : Natural := 0;
 end record;
```

The type Element represents the kind of individual values contained by stack objects. Top is used as the index into the array Values and can be zero. The Values array uses 1 for the lower index bound so when Top is zero the enclosing stack object is logically empty. The following function checks for that condition:

```
function Empty (This : Stack) return Boolean is
  (This.Top = 0);
```

Consider, then, a function using Empty as a precondition. The function takes a stack parameter as input and returns the Element value at the logical top of the stack:

```
19 function Top_Element (This : Stack) return Element
     with
20   Pre => not Empty (This);
```

Given the precondition on line 20, within the function implementation we know that Top has a value that is a potentially valid array index. (We'll also have to be more precise about Top's upper bound, as explained later in section 4.4.) There is no need for defensive code so the body is simply as follows:

```
57 function Top_Element (This : Stack) return Element
     is
58   (This.Values (This.Top));
```

If we did not have the precondition specified, GNATprove would issue a message:

```
58:24: medium: array index check might fail, (e.g. when
This = (…, Top => 0) and …)
```

The message shows an example situation in which the check could fail: Top is zero, i.e., the stack is empty. (We have elided some of the message content to highlight the part mentioning Top.)

GNATprove will attempt to prove, statically, that the preconditions hold at every call site, flagging those calls, if any, in which the preconditions might not hold. Those failures must be addressed at the Silver level because the preconditions are necessary to the proof of absence of run-time errors.

As you can see, the Silver level provides highly significant benefits, but does require more contracts and potentially complex changes to the code. The effort required to achieve this level can be high. Arguably, however, this level should be the minimum target level, especially if the application executable is to be deployed with run-time checks disabled.

2.4 Gold Level

The goal at the Gold level is proof of key integrity properties. These properties are typically derived from software requirements but also include maintaining critical data invariants throughout execution. Working at this level assumes prior completion at the Silver level to ensure program integrity, such that control flow cannot be circumvented through run-time errors and data cannot be corrupted. Verification at this level is also expected to pass without any violations.

Key integrity properties are expressed as additional preconditions and postconditions beyond those used for defensive purposes. In addition, the application may explicitly raise application-defined exceptions to signal violations of integrity properties. GNATprove will attempt to prove that the code raising an exception is never reached, and thus, that the property violation never occurs. This approach may also require further proof-oriented code.

The Gold level provides extremely significant benefits. In particular, it can be less expensive to prove at this level than to test to the same degree of confidence. However, the analysis may take a long time, may require adding more precise types (ranges), and may require adding more preconditions and postconditions. Even if a property is provable, automatic provers may fail to prove it due to limitations of the provers, requiring either manual proof or, alternatively, testing.

2.5 Platinum Level

The goal at the Platinum level is nothing less than full functional proof of the requirements, including the functional unit level requirements, but also any abstract requirements such as, for example, safety and security.

As with the Gold level, the application code must pass SPARK analysis without any violations. Furthermore, at the Platinum level GNATprove must verify complete user specifications for type invariants, preconditions, postconditions, type predicates, loop variants, and loop termination.

The effort to achieve Platinum level is high, so high that this level is not recommended during initial adoption of SPARK.

3 Development Environment and Configuration

When we say we use SPARK, we mean that we develop the sources in the SPARK language, but also that we use the SPARK analysis tool to examine and verify those sources. We developed our sources in GNAT Studio (formerly GPS), a multi-lingual IDE supporting both Ada and SPARK, among others. The SPARK analysis tool is named GNATprove, a command-line tool integrated with GNAT Studio. GNAT Studio facilitates invocation of GNATprove with control over switches and source files, providing traversable results and even, if need be, interactive proof.

3.1 The Provers

A critical concept for using GNATprove is that it transparently invokes third-party "provers" to analyze the given source files. These provers are somewhat specialized in their ability to analyze specific semantics expressed by the source code. As a result, invocation of a series of provers may be required before some source code is successfully proven. In addition, we may need to ask the provers to "try harder" when attempting to analyze difficult situations. GNATprove can do both for us via the "level=n" switch, where "n" is a number from 0 to 4 indicating increasing strength of analysis and additional provers invoked. In proving our stack implementation we use level 4.

3.2 Language-Defined Run-time Checks

GNATprove is also integrated with the GNAT Ada compiler, including the analysis of language-defined run-time checks produced by the compiler. GNATprove attempts to verify that no exceptions are raised due to these checks. It will do so even if we suppress the checks with compiler switches or pragma Suppress, so we can interpret lack of corresponding messages as successful verification of those checks.

Integer overflow checks are a special case, and as a result have a dedicated GNAT switch that affects whether that specific check is generated by the compiler. They are a special case because, in addition to the functional code, they may appear in the logical assertions about the functional code, including subprogram preconditions and postconditions. In these contexts, we might expect them to behave mathematically, without implementation bounds. For example, consider the following declaration for a procedure that enters a log entry into a file:

```
5  Entry_Num : Natural := 0;
6
7  procedure Log (This : String) with
8    Pre  => Entry_Num + 1 <= Integer'Last,
9    Global => (In_Out => Entry_Num);
```

The procedure body increments Entry_Num by one and then prepends the result to the string passed as the log entry. This addition in the body might overflow, but the issue under consideration is the addition in the precondition on line 8. If Entry_Num is Integer'Last at the point of the call, the addition on line 8 will overflow, as GNATprove indicates:

```
8:26: medium: overflow check might fail (e.g. when
Entry_Num = Natural'Last)
```

We could revise the code so that the expression cannot overflow:

```
Pre => Entry_Num <= Integer'Last - 1,
```

although that is slightly less readable. Other alternatives within the code are possible as well. However, with regard to switches pertinent for check generation, GNAT provides the "-gnato" switch that allows us to control how integer overflow is treated. (There is a pragma as well, with the same effects.) We can use that switch to have the compiler implement integer arithmetic mathematically, without bounds, the way we might conceptually expect it to work within logical, non-functional assertions. As a result, there will be no integer overflow checks generated. The default effect for the switch, and the default if the switch is not present, is to enable overflow checks in both functional and assertion code so we just need to be aware of non-default usage when we want to determine whether integer overflow checks have been verified. (See the SPARK User Guide, section 5.7 "Overflow Modes" for the switch parameters.) In our GNAT project file, the switch is explicitly set to enable overflow checks in both the functional code and the assertion code.

3.3 Source Code File Organization

Logically, there are four source files in the application: two (declaration and body) for the generic package, one for the instantiation of that generic package, and one containing the demonstration main subprogram. Operationally, however, there are multiple source files for the generic package. Rather than have one implementation that we alter as we progress through the SPARK adoption levels, we have chosen to have a distinct generic package for each level. Each generic package implements a common stack ADT in a manner consistent with an adoption level. The differences among them reflect the changes required for the different levels. This approach makes it easier to keep the differences straight when examining the code. Furthermore, we can apply the proof analyses to a conceptually common abstraction at arbitrary adoption levels without having to alter the code.

In addition to the content differences required by the adoption levels, each generic package name reflects the corresponding level. We have generic package Bounded_Stacks_Stone for the Stone level, Bounded_Stacks_Gold for the Gold level, and so on. Therefore, although the instantiation is always named Character_Stacks, we have multiple generic packages available to declare the one instantiation used.

There are also multiple files for the instantiations. Each instantiation is located within a dedicated source file corresponding to a given adoption level (lines 2 and 3 below). For example, here is the content of the file providing the instance for the Stone level:

```
1  pragma Spark_Mode (On);
2  with Bounded_Stacks_Stone;
```

```
3  package Character_Stacks is new
     Bounded_Stacks_Stone
4    (…);
```

The file names for these instances must be unique but are otherwise arbitrary. For the above, the file name is "character_stacks-stone.ads" because it is the instance of the Stone level generic.

Only one of these instances can be used when GNATprove analyzes the code (or when building the executable). To select among them we use a "scenario variable" defined in the GNAT project file that has scenario values matching the adoption level names. In the IDE this scenario variable is presented with a pull-down menu so all we must do to work at a given level is select the adoption level name in the pull-down list. The project file then selects the instantiation file corresponding to the level, e.g., "character_stacks-silver.ads" when the Silver level is selected.

There are also multiple source files for the main program. Rather than have one file that must be edited as we prove the higher levels, we have two: one for all levels up to and including the Silver level, and one for all levels above that. The scenario variable also determines which of these two source files is active. The main procedures exist only to act as clients so that we prove certain properties about the ``Stack`` type. Therefore, they declare objects of that type and make a series of assertions and calls to ``Stack`` operations. They have no functional purpose whatsoever.

3.4 Verifying Generic Units

One of the current limitations of GNATprove is that it cannot verify generic units on their own. GNATprove must instead be provided an instantiation to verify. Therefore, whenever we say that we are verifying the generic package defining the stack ADT, we mean we are invoking GNATprove on an instantiation of that generic. As noted earlier in section 3.3, there are multiple source files containing these instantiations so we must select the file corresponding to the desired level when we want to verify the generic package alone.

However, because there are only four total files required at any one time, we usually invoke the IDE action that has GNATprove analyze all the files in the closure of the application. The instantiation file corresponding to the scenario variable's current selection will be analyzed; other instantiation files are ignored. This approach also verifies the main program's calls to the stack routines, which is vital to the higher adoption levels.

4 Implementations Per Adoption Level

Our first main procedure (Listing 1), used for all adoption levels up through Silver, declares two stack objects (line 6) and manipulates them via the abstraction's interface:

This is the "demo_aorte.adb" file. The purpose of the code is to illustrate issues found at the initial levels, including proof in a caller context. It has no functional purpose whatsoever. As we progress through the levels, we will add more assertions to highlight more issues, as will be seen in the other main procedure in the "demo_gold.adb" file.

4.1 Initial Ada Implementation

The initial version defines a canonical representation of a sequential, bounded stack. As an abstract data type, the Stack type is declared as a private type with routines manipulating objects of the type. The type is declared within a generic package that has one generic formal parameter, a type representing the kind of elements

```
1  with Ada.Text_IO; use Ada.Text_IO;
2  with Character_Stacks; use Character_Stacks;
3
4  procedure Demo_AoRTE with SPARK_Mode is
5
6    S1, S2 : Stack (Capacity => 10);  -- arbitrary
7
8    X, Y : Character;
9
10 begin
11   pragma Assert (Empty (S1) and Empty (S2));
12   pragma Assert (S1 = S2);
13   Push (S1, 'a');
14   Push (S1, 'b');
15   Put_Line ("Top of S1 is '" & Top_Element (S1) & "'");
16
17   Pop (S1, X);
18   Put_Line ("Top of S1 is '" & Top_Element (S1) & "'");
19   Pop (S1, Y);
20   pragma Assert (Empty (S1) and Empty (S2));
21   Put_Line (X & Y);
22
23   Reset (S1);
24   Put_Line ("Extent of S1 is" & Extent (S1)'Image);
25
26   Put_Line ("Done");
27 end Demo_AoRTE;
```

Listing 1. Procedure Demo_AoRTE

```
1  generic
2    type Element is private;
3  package Bounded_Stacks_Magma is
4
5    type Stack (Capacity : Positive) is private;
6
7    procedure Push (This : in out Stack; Item : Element) with
8      Pre => not Full (This) or else raise Overflow;
9
10   procedure Pop (This : in out Stack; Item : out Element) with
11     Pre => not Empty (This) or else raise Underflow;
12
13   function Top_Element (This : Stack) return Element with
14     Pre => not Empty (This) or else raise Underflow;
15     -- Returns the value of the Element at the "top" of This
16     -- stack, i.e., the most recent Element pushed. Does not
17     -- remove that Element or alter the state of This stack
18     -- in any way.
19
20   overriding function "=" (Left, Right : Stack) return Boolean;
21
22   procedure Copy (Destination : out Stack; Source : Stack) with
23     Pre => Destination.Capacity >= Extent (Source)
24       or else raise Overflow;
25     -- An alternative to predefined assignment that does not
26     -- copy all the values unless necessary. It only copies
27     -- the part "logically" contained, so is more efficient
28     -- when Source is not full.
29
30   function Extent (This : Stack) return Natural;
31     -- Returns the number of Element values currently
32     -- contained within This stack.
33
34   function Empty (This : Stack) return Boolean;
35
36   function Full (This : Stack) return Boolean;
37
38   procedure Reset (This : out Stack);
39
40   Overflow  : exception;
41   Underflow : exception;
42
43 private
44
45   type Content is array (Positive range <>) of Element;
46
47   type Stack (Capacity : Positive) is record
48     Values : Content (1 .. Capacity);
49     Top    : Natural := 0;
50   end record;
51
52 end Bounded_Stacks_Magma;
```

Listing 2. Generic package Bounded_Stacks_Magma

contained by Stack objects. This approach is used in all the implementations.

Some routines have “defensive” preconditions to ensure correct functionality. They raise exceptions, declared within the package, when the preconditions do not hold.

The generic package in Ada is declared in Listing 2.

This version is below the Stone level because it is not within the SPARK subset, due to the raise expressions on lines 8, 11, 14, and 24. We will address those constructs in the Stone version.

The generic package body is shown in Listing 3.

Note that both procedure Copy and function “=” are defined for the sake of increased efficiency when the objects in question are not full. The procedure only copies the slice of Source.Values that represents the Element values logically contained at the time of the call. The language-defined assignment operation, in contrast, would copy the entire contents. Similarly, the overridden equality operator only compares the array slices, rather than the entire arrays, after first ensuring the stacks are the same logical size.

However, in addition to efficiency, the "=" function is also required for proper semantics. The comparison should not compare array elements that are not, and perhaps never have been, currently contained in the stack objects. The

```
1  package body Bounded_Stacks_Magma is
2
3    procedure Reset (This : out Stack) is
4    begin
5      This.Top := 0;
6    end Reset;
7
8    function Extent (This : Stack) return Natural is
9      (This.Top);
10
11   function Empty (This : Stack) return Boolean is
12     (This.Top = 0);
13
14   function Full (This : Stack) return Boolean is
15     (This.Top = This.Capacity);
16
17   procedure Push (This : in out Stack; Item : Element) is
18   begin
19     This.Top := This.Top + 1;
20     This.Values (This.Top) := Item;
21   end Push;
22
23   procedure Pop (This : in out Stack; Item : out Element) is
24   begin
25     Item := This.Values (This.Top);
26     This.Top := This.Top - 1;
27   end Pop;
28
29   function Top_Element (This : Stack) return Element is
30     (This.Values (This.Top));
31
32   function "=" (Left, Right : Stack) return Boolean is
33     (Left.Top = Right.Top and then
34      Left.Values (1 .. Left.Top) = Right.Values (1 .. Right.Top));
35
36   procedure Copy (Destination : out Stack; Source : Stack) is
37     subtype Contained is Integer range 1 .. Source.Top;
38   begin
39     Destination.Top := Source.Top;
40     Destination.Values (Contained) := Source.Values (Contained);
41   end Copy;
42
43 end Bounded_Stacks_Magma;
```

Listing 3. Generic package body.

predefined equality would do so and must, therefore, be replaced.

The changes to the body made for the sake of SPARK will amount to moving certain bodies to the package declaration so we will not show the package body again. The full Platinum implementation, both declaration and body, is provided at the end of this article.

4.2 Stone Implementation

The Stone level version of the package cannot have the "raise expressions" in the preconditions because they are not in the SPARK subset. The rest of the preconditions are unchanged. Here are the updated declarations for Push and Pop, for example:

```
procedure Push (This : in out Stack; Item : Element)
with Pre => not Full (This);

procedure Pop (This : in out Stack; Item : out Element)
with  Pre => not Empty (This);
```

When we get to the adoption levels involving proof, GNATprove will attempt to verify statically that the preconditions will hold at each call site. Either that verification will succeed, or we will know that we must change the calling code accordingly. Therefore, the prohibited “raise expressions” are not needed.

The exception declarations, although within the subset, are also removed because they are no longer needed.

The remaining code is wholly within the SPARK subset so we have reached the Stone level.

4.3 Bronze Implementation

The Bronze level is about initialization and data flow. When we apply GNATprove to the Stone version in flow analysis mode, GNATprove issues messages on the declarations of procedures Copy and Reset in the generic package declaration:

```
medium: "Destination.Values" might not be initialized
in "Copy"

high: "This.Values" is not initialized in "Reset"
```

The procedure declarations are repeated below for reference:

```
procedure Copy (Destination : out Stack;
                Source : Stack)
  with Pre => Destination.Capacity >= Extent (Source);

procedure Reset (This : out Stack);
```

Both messages result from the fact that the updated formal stack parameters have mode "out" specified. That mode, in SPARK, means more than it does in Ada. It indicates that the actual parameters are fully assigned by the procedures, but these two procedure bodies do not do so. Procedure Reset simply sets the Top to zero because that is all that a stack requires, at run-time, to be fully reset. It does nothing at all to the Values array component. Likewise, procedure Copy may only assign part of the array, i.e., just those array components that are logically part of the Source object. (Of course, if Source is full, the entire array is copied.) In both subprograms our notion of being fully assigned is less than SPARK requires. Therefore, we have two choices. Either we assign values to all components of the record, or we change the modes to "in out." These two procedures exist for the sake of efficiency, i.e., not writing any more data than logically necessary. Having Reset assign anything to the array component would defeat the purpose. For the same reason, having Copy assign more than the partial slice (when the stack is not full) is clearly inappropriate. Therefore, we change the mode to "in out" for these two subprograms. In other cases we might change the implementations to fully assign the objects.

The other change required for initialization concerns the type Stack itself. In the main subprogram, GNATprove complains that the two objects of type Stack have not been initialized:

```
warning: "S1" may be referenced before it has a value

high: private part of "S1" is not initialized

warning: "S2" may be referenced before it has a value

high: private part of "S2" is not initialized

high: private part of "S1" is not initialized
```

Our full definition of the Stack type in the private part is such that default initialization (i.e., elaboration of object declarations without an explicit initial value) will assign the record components so that a stack will behave as if initially empty. Specifically, default initialization assigns zero to Top (line 5 below), and since function Empty examines only the Top component, such objects are empty.

```
1  type Content is array (Positive range <>) of Element;
2
3  type Stack (Capacity : Positive) is record
4    Values : Content (1 .. Capacity);
5    Top : Natural := 0;
6  end record;
```

Proper run-time functionality of the Stack ADT does not require the Values array component to be assigned by default initialization. But just as with Reset and Copy, although this approach is sufficient at run-time, the resulting objects will not be fully initialized in SPARK, which analyzes the code prior to run-time. As a result, we need to assign an array aggregate to the Values component as well. Expressing the array aggregate is problematic because the array component type is the generic formal private type Element, with a private view within the package. Inside the generic package we don't know how to construct a value of type Element so we cannot construct an aggregate containing such values. Therefore, we add the Default_Value generic formal object parameter and use it to initialize the array components.

This new generic formal parameter, shown below on line 5, is added from the Bronze version onward:

```
1  generic
2    type Element is private;
3    -- The type of values contained by objects
     -- of type Stack
4
5    Default_Value : Element;
6    -- The default value used for stack contents. Never
7    -- acquired as a value from the API, but required for
8    -- initialization in SPARK.
9  package Bounded_Stacks_Bronze is
```

The full definition for type Stack then uses that parameter to initialize Values (line 2):

```
1  type Stack (Capacity : Positive) is record
2    Values : Content (1 .. Capacity) :=
                                  (others => Default_Value);
3    Top : Natural := 0;
4  end record;
```

With those changes in place flow analysis completes without further complaint. The implementation has reached the Bronze level.

The need for that additional generic formal parameter is unfortunate because it becomes part of the user's interface without any functional use. None of the API routines ever return it as a value as such, and the actual value chosen is immaterial.

Note that SPARK will not allow the aggregate to contain default components (line 2):

```
1  type Stack (Capacity : Positive) is record
2    Values : Content (1 .. Capacity) := (others => <>);
3    Top : Natural := 0;
4  end record;
```

as per SPARK RM 4.3(1).

Alternatively, we could omit this generic formal object parameter if we use an aspect to promise that the objects are initially empty, and then manually justify any resulting messages. We will in fact add that aspect for other reasons, but we prefer to have proof as automated as possible, for convenience and to avoid human error.

Finally, although the data dependency contracts, i.e., the "Global" aspects, would be generated automatically, we add them explicitly, indicating that there are no intended accesses to any global objects. For example, on line 3 in the following:

```
1  procedure Push (This : in out Stack; Item : Element)
2    with Pre => not Full (This),
3         Global => null;
```

We do so because mismatches between reality and the generated contracts are not reported by GNATprove, but we prefer positive confirmation for our understanding of the dependencies.

The flow dependency contracts (the "Depends" aspects) also can be generated automatically. Unlike the data dependency contracts, however, usually these can be omitted from the code even though mismatches with the corresponding bodies are not reported. That lack of notification is not a problem because the generated contracts are safe: they express at least the dependencies that the code actually exhibits. Therefore, all actual dependencies are covered. For example, a generated flow dependency will state that all outputs depend on all inputs, which is possible but not necessarily the case.

However, overly conservative contracts can lead to otherwise-avoidable issues with proof, leading the developer to add precise contracts explicitly when necessary. The other reason to express them explicitly is when we want to prove data flow dependencies as part of the abstract properties, for example data flowing only between units at appropriate security levels. We are not doing so in this case.

4.4 Silver Implementation

If we try to prove the Bronze level version of the generic package, GNATprove will complain about various run-time checks that cannot be proved in the generic package body. The Silver level requires these checks to be proven not to fail, i.e., not to raise exceptions.

The check messages are as follows, preceded by the code fragments they reference, with some message content elided in order to emphasize parts that lead us to the solution:

```
37 procedure Push (This : in out Stack; Item : Element)
   is
38 begin
39  This.Top := This.Top + 1;
40  This.Values (This.Top) := Item;
41 end Push;
```

bounded_stacks_silver.adb:39:28: medium: overflow check might fail, … (e.g. when This = (…, Top => Natural'Last) …

bounded_stacks_silver.adb:40:24: medium: array index check might fail, … (e.g. when This = (…, Top => 2) and This.Values'First = 1 and This.Values'Last = 1)

```
47 procedure Pop (This : in out Stack; Item : out
   Element) is
48 begin
49  Item := This.Values (This.Top);
50  This.Top := This.Top - 1;
51 end Pop;
```

bounded_stacks_silver.adb:49:32: medium: array index check might fail, … (e.g. when This = (…, Top => 2) and This.Values'First = 1 and This.Values'Last = 1)

```
57 function Top_Element (This : Stack) return Element
   is
58  (This.Values (This.Top));
```

bounded_stacks_silver.adb:58:24: medium: array index check might fail, … (e.g. when This = (…, Top => 2) and This.Values'First = 1 and This.Values'Last = 1)

```
64 function "=" (Left, Right : Stack) return Boolean is
65  (Left.Top = Right.Top and then
66   Left.Values (1 .. Left.Top) = Right.Values (1 ..
     Right.Top));
```

bounded_stacks_silver.adb:66:12: medium: range check might fail, … (e.g. when Left = (Capacity => 1, …, Top => 2) …

bounded_stacks_silver.adb:66:43: medium: range check might fail, … (e.g. when Right = (Capacity => 1, …, Top => 2) …

```
72 procedure Copy(Destination : in out Stack; Source :
   Stack) is
73  subtype Contained is Integer range 1 .. Source.Top;
74 begin
75  Destination.Top := Source.Top;
76  Destination.Values(Contained) :=
     Source.Values(Contained);
77 end Copy;
```

bounded_stacks_silver.adb:76:47: medium: range check might fail, … (e.g. when Destination = (Capacity => 1, …) and Source = (Capacity => 1, …), Top => 2)

All of these messages indicate that the provers do not know that the Top component is always in the range 0 .. Capacity. The code has not said so, and indeed, there is no way to use a discriminant in a scalar record component declaration to constrain the component's range. This is what we would write for the record type implementing type Stack in the full view, if we could (line 3):

```
1 type Stack (Capacity : Positive) is record
2  Values : Content (1 .. Capacity) := (others =>
    Default_Value);
3  Top : Natural range 0 .. Capacity := 0;
4 end record;
```

but that range constraint on Top is not legal. The reason it is illegal is that the application can change the value of a discriminant at run-time, under controlled circumstances, but there is no way at run-time to change the range checks in the object code generated by the compiler. With Ada and SPARK there is now a way to express the constraint on Top, and the provers will recognize the meaning during analysis. Specifically, we apply a "subtype predicate" to the record type declaration (line 5):

```
1 type Stack (Capacity : Positive) is record
2  Values : Content (1 .. Capacity) := (others =>
    Default_Value);
3  Top : Natural := 0;
4 end record with
5  Predicate => Top in 0 .. Capacity;
```

This aspect informs the provers that the Top component for any object of type Stack is always in the range 0 .. Capacity. That addition successfully addresses all the messages about the generic package body. Note that the provers will verify the predicate too.

However, GNATprove also complains about the main program. Consider that the first two assertions in the main procedure are not verified:

```
10 begin
11    pragma Assert (Empty (S1) and Empty (S2));
12    pragma Assert (S1 = S2);
```

GNATprove emits:

> 11:19: medium: assertion might fail, cannot prove Empty (S1)
>
> 12:19: medium: assertion might fail, cannot prove S1 = S2

We can address the issue for function Empty, partly, by adding another aspect to the declaration of type Stack, this time to the visible declaration:

```
type Stack (Capacity : Positive) is private
  with Default_Initial_Condition => Empty (Stack);
```

The new aspect indicates that default initialization results in stack objects that are empty, making explicit, and especially, verifiable, the intended initial object state. We will be notified if GNATprove determines that the aspect does not hold.

That new aspect will handle the first assertion in the main program on line 11 but GNATprove complains throughout the main procedure that the preconditions involving Empty and Full cannot be proven. For example:

```
13 Push (S1, 'a');
14 Push (S1, 'b');
15 Put_Line ("Top of S1 is '" & Top_Element (S1) & "'");
```

GNATprove emits:

> 13:06: medium: precondition might fail, cannot prove not Full (This)
>
> 14:06: medium: precondition might fail, cannot prove not Full (This) [possible explanation: call at line 13 should mention This (for argument S1) in a postcondition]
>
> 15:35: medium: precondition might fail, cannot prove not Empty (This) [possible explanation: call at line 14 should mention This (for argument S1) in a postcondition]

Note the “possible explanations” that GNATprove gives us. These are clear indications that we are not specifying sufficient postconditions. Remember that when analyzing code that includes a call to some procedure, the provers’ knowledge of the call’s effect is provided entirely by the procedure’s postcondition. That postcondition might be insufficient, especially if it is absent!

Therefore, we must tell the provers about the effects of calling Push and Pop, as well as the other routines that change state. We add a new postcondition on Push (line 3):

```
1 procedure Push (This : in out Stack; Item : Element)
with
2    Pre => not Full (This),
3    Post => Extent (This) = Extent (This)'Old + 1,
4    Global => null;
```

The new postcondition expresses the fact that the Stack contains one more Element value after the call. This is sufficient because the provers know that function Extent is simply the value of Top:

```
function Extent (This : Stack) return Natural is
  (This.Top);
```

Hence the provers know that Top is incremented by Push.

The same approach addresses the messages for Pop (line 3):

```
1 procedure Pop (This : in out Stack; Item : out
Element) with
2    Pre => not Empty (This),
3    Post => Extent (This) = Extent (This)'Old - 1,
4    Global => null;
```

In the above we say that the provers know what the function Extent means. For that to be the case when verifying client calls, we must move the function completion from the generic package body to the generic package declaration. In addition, the function must be implemented as an “expression function,” which Extent already is (see above). As expression functions in the package spec, the provers will know the semantics of those functions automatically, as if each is given a postcondition restating the corresponding expression explicitly. We also need functions Full and Empty to be known in this manner. Therefore, we move the Extent, Empty, and Full function completions, already expression functions, from the generic package body to the package declaration. We put them in the private part because these implementation details should not be exported to clients.

However, we have a potential overflow in the postcondition for Push, i.e., the increment of the number of elements contained after Push returns (line 3 below). The postcondition for procedure Pop, of course, does not have that problem.

```
1  procedure Push (This : in out Stack; Item : Element)
with
2     Pre => not Full (This),
3     Post => Extent (This) = Extent (This)'Old + 1,
4     Global => null;
```

The increment might overflow because Extent returns a value of subtype Natural, which could be the value Integer'Last. Hence the increment could raise Constraint_Error and the check cannot be verified. We must either apply the “-gnato” switch so that assertions can never overflow, or alternatively, declare a safe subrange so that the result of the addition cannot be greater than Integer'Last.

Our choice is to declare a safe subrange because the effects are explicit in the code, as opposed to an external switch. Therefore, here are the added subtype declarations:

```
subtype Element_Count is
  Integer range 0 .. Integer'Last - 1;
-- The number of Element values currently contained
-- within any given stack. The lower bound is zero
-- because a stack can be empty. We limit the upper
-- bound (minimally) to preclude overflow issues.
```

```
   subtype Physical_Capacity is
     Element_Count range 1 .. Element_Count'Last;
   -- The range of values that any given stack object can
   -- specify (via the discriminant) for the number of
   -- Element values the object can physically contain.
   -- Must be at least one.
```

We use the second subtype for the discriminant in the partial view for Stack (line 1):

```
1  type Stack (Capacity : Physical_Capacity) is private
2    with Default_Initial_Condition => Empty (Stack);
```

and both subtypes in the full declaration in the private part (lines 1, 3, and 5):

```
1 type Content is array (Physical_Capacity range <>)
                                          of Element;
2
3 type Stack (Capacity : Physical_Capacity) is record
4   Values : Content (1 .. Capacity) :=
                                  (others =>
Default_Value);
5   Top : Element_Count := 0;
6 end record with
7   Predicate => Top in 0 .. Capacity;
```

The function Extent is changed to return a value of the subtype Element_Count so adding one in the postcondition cannot go past Integer'Last. Overflow is precluded but note that there will now be range checks for GNATprove to verify.

With these changes in place we have achieved the Silver level. There are no run-time check verification failures and the defensive preconditions are proven at their call sites.

4.5 Gold Implementation

We will now address the remaining changes needed to reach the Gold level. The process involves iteratively attempting to prove the main program that calls the stack routines and makes assertions about the conditions that follow. This process will result in changes to the generic package, especially postconditions, so it will require verification along with the main procedure. Those additional postconditions may require additional preconditions as well.

In general, a good way to identify postcondition candidates is to ask ourselves what conditions we, as the developers, know to be true after a call to the routine in question. Then we can add assertions after the calls to see if the provers can verify those conditions. If not, we extend the postcondition on the routine.

```
1  with Ada.Text_IO; use Ada.Text_IO;
2  with Character_Stacks; use Character_Stacks;
3
4  procedure Demo_Gold with SPARK_Mode is
5
6    S1, S2 : Stack (Capacity => 10);  -- arbitrary
7
8    X, Y : Character;
9
10 begin
11   pragma Assert (Empty (S1) and Empty (S2));
12   pragma Assert (S1 = S2);
13   Push (S1, 'a');
14   pragma Assert (not Empty (S1));
15   pragma Assert (Top_Element (S1) = 'a');
16   Push (S1, 'b');
17   pragma Assert (S1 /= S2);
18
19   Put_Line ("Top of S1 is '" & Top_Element (S1) & "'");
20
21   Pop (S1, X);
22   Put_Line ("Top of S1 is '" & Top_Element (S1) & "'");
23   Pop (S1, Y);
24   pragma Assert (X = 'b');
25   pragma Assert (Y = 'a');
26   pragma Assert (S1 = S2);
27   Put_Line (X & Y);
28
29   Push (S1, 'a');
30   Copy (Source => S1, Destination => S2);
31   pragma Assert (S1 = S2);
32   pragma Assert (Top_Element (S1) = Top_Element (S2));
33   pragma Assert (Extent (S1) = Extent (S2));
34
35   Reset (S1);
36   pragma Assert (Empty (S1));
37   pragma Assert (S1 /= S2);
38
39   Put_Line ("Done");
40 end Demo_Gold;
```

Listing 4. Demo_Gold procedure.

For example, we can say that after a call to Push, the corresponding stack cannot be empty. Likewise, after a call to Pop, the stack cannot be full. These additions are not required for the sake of assertions or other preconditions because the Extent function already tells the provers what they need to know in this regard. However, they are good documentation and may be required to prove additional conditions added later. (That is the case, in fact, as will be shown.)

To see what other postconditions are required, we now switch to the other main procedure, in the "demo_gold.adb" file. This version of the demo program includes a number of additional assertions (Listing 4).

For example, we have added assertions after the calls to Reset and Copy, on lines 31 through 33 and 36 through 37, respectively. GNATprove now emits the following (elided) messages for those assertions:

> demo_gold.adb:31:19: medium: assertion might fail, cannot prove S1 = S2 (e.g. when S1 = (…, Top => 0) and S2 = (…, Top => 0)) [possible explanation: call at line 30 should mention Destination (for argument S2) in a postcondition]
>
> demo_gold.adb:36:19: medium: assertion might fail, cannot prove Empty (S1) … [possible explanation: call at line 35 should mention This (for argument S1) in a postcondition]

Note again the "possible explanation" hints. For the first message we need to add a postcondition on Copy specifying that the value of the argument passed to Destination will be equal to that of the Source parameter (line 3):

```
1  procedure Copy (Destination : in out Stack;
                                        Source : Stack)
with
2     Pre => Destination.Capacity >= Extent (Source),
3     Post => Destination = Source,
4     Global => null;
```

We must move the "=" function implementation to the package spec so that the provers will know the meaning. The function was already completed as an expression function so moving it to the spec is all that is required.

For the second message, regarding the failure to prove that a stack is Empty after Reset, we add a postcondition to that effect (line 2):

```
1  procedure Reset (This : in out Stack) with
2     Post   => Empty (This),
3     Global => null;
```

The completion for function Empty was already moved to the package spec, earlier.

The implementations of procedure Copy and function "=" might have required explicit loops, likely requiring loop invariants, but using array slicing we can express the loop implicitly. Here is function "=" again, for example:

```
1 function "=" (Left, Right : Stack) return Boolean is
2  (Left.Top = Right.Top and then
3   Left.Values (1 .. Left.Top) = Right.Values (1 ..
   Right.Top));
```

The slice comparison on line 3 expresses an implicit loop for us, as does the slice assignment in procedure Copy.

The function could have been implemented as follows, with an explicit loop:

```
1  function "=" (Left, Right : Stack) return Boolean is
2  begin
3     if Left.Top /= Right.Top then
4        -- They hold a different number of element values
   so
5        -- cannot be equal.
6        return False;
7     end if;
8     -- The two Top values are the same, and the arrays
9     -- are 1-based, so the bounds are the same. Hence
   the
10    -- choice of Left.Top or Right.Top is arbitrary and
11    -- there is no need for index offsets.
12    for K in 1 .. Left.Top loop
13       if Left.Values (K) /= Right.Values (K) then
14          return False;
15       end if;
16       pragma Loop_Invariant
17          (Left.Values (1 .. K) = Right.Values (1 .. K));
18    end loop;
19    -- We didn't find a difference
20    return True;
21 end "=";
```

Note the loop invariant on lines 16 and 17. In some circumstances GNATprove will handle the invariants for us but often it cannot. In practice, writing sufficient loop invariants is one of the more difficult facets of SPARK development so the chance to avoid them is welcome.

Continuing, we know that after the body of Push executes, the top element contained in the stack will be the value passed to Push as an argument. But the provers cannot verify an assertion to that effect (line 15 below):

```
13 Push (S1, 'a');
14 pragma Assert (not Empty (S1));
15 pragma Assert (Top_Element (S1) = 'a');
```

GNATprove emits this message:

> demo_gold.adb:15:19: medium: assertion might fail, cannot prove Top_Element (S1) = 'a'

We must extend the postcondition for Push to state that Top_Element would return the value just pushed, as shown on line 4 below:

```
1  procedure Push (This : in out Stack; Item : Element)
with
2     Pre => not Full (This),
3     Post => not Empty (This)
4        and then Top_Element (This) = Item
5        and then Extent (This) = Extent (This)'Old + 1,
6     Global => null;
```

Now the assertion on line 15 is verified successfully.

Recall that the precondition for function Top_Element is that the stack is not empty. We already have that assertion in the postcondition (line 3) so the precondition for Top_Element is satisfied. We must use the short circuit

form for the conjunction, though, to control the order of evaluation so that "not Empty" is verified before Top_Element.

The short-circuit form on line 4 necessitates the same form on line 5, per Ada rules. That triggers a subtle issue flagged by GNATprove. The short-circuit form, by definition, means that the evaluation of line 5 might not occur. If it is not evaluated, we've told the compiler to call Extent and make a copy of the result (via 'Old, on the right-hand side of "=") that will not be needed. Moreover, the execution of Extent might raise an exception. Therefore, the language disallows applying 'Old in any potentially unevaluated expression that might raise exceptions. As a consequence, in line 5 we cannot apply 'Old to the result of calling Extent. GNATprove issues this error message:

> prefix of attribute "Old" that is potentially unevaluated must denote an entity

We could address the error by changing line 5 to use Extent(This'Old) instead, but there is a potential performance difference between Extent(This)'Old and Extent(This'Old). With the former, only the result of the function call is copied, whereas with the latter, the value of the parameter is copied. Copying the parameter could take significant time and space if This is a large object. Of course, if the function returns a large value the copy will be large too, but in this case Extent only returns an integer.

In SPARK, unlike Ada, preconditions, postconditions, and assertions in general are verified statically, prior to execution, so there is no performance issue. Ultimately, though, the application will be executed. Having statically proven the preconditions and postconditions successfully, we can safely deploy the final executable without them enabled, but not all projects follow that approach (at least, not on that basis). Therefore, for the sake of emphasizing the idiom with typically better performance, we prefer applying 'Old to the function in our implementation.

We can tell GNATprove that this is a benign case, using a pragma in the package spec:

```
pragma Unevaluated_Use_of_Old (Allow);
```

GNATprove will then allow use of 'Old on the call to function Extent and will ensure that no exceptions will be raised by the function.

As with procedure Push, we can also use Top_Element to strengthen the postcondition for procedure Pop (line 4 below):

```
1  procedure Pop (This : in out Stack; Item : out
Element)
2    with Pre => not Empty (This),
3    Post => not Full (This)
4      and Item = Top_Element (This)'Old
5      and Extent (This) = Extent (This)'Old – 1,
6    Global => null;
```

Line 4 states that the Item returned in the parameter to Pop is the value that would be returned by Top_Element prior to the call to Pop.

One last significant enhancement now remains to be made. Consider the assertions in the main procedure about the effects of Pop on lines 24 and 25, repeated below:

```
21 Pop (S1, X);
22 Put_Line ("Top of S1 is '" & Top_Element (S1) & "'");
23 Pop (S1, Y);
24 pragma Assert (X = 'b');
25 pragma Assert (Y = 'a');
```

Previous lines had pushed 'a' and then 'b' in that order onto S1. GNATprove emits this one message:

> 25:19: medium: assertion might fail, cannot prove Y = 'a' (e.g. when Y = 'b')

The message is about the assertion on line 25, alone. The assertion on line 24 was verified. Also, the message indicates that Y could be some arbitrary character. We can conclude that the provers do not know enough about the state of the stack after a call to Pop. The postcondition requires strengthening.

The necessary postcondition extension reflects a unit-level functional requirement for both Push and Pop. If one considers that postconditions correspond to the low-level unit functional requirements (if not more), one can see why the postconditions must be complete. Identifying and expressing complete functional requirements is difficult in itself, and indeed the need for this additional postcondition content is not obvious at first.

The unit-level requirement for both operations is that the prior array components within the stack are not altered, other than the one added or removed. We need to state that Push and Pop have not reordered them, for example, nor changed their values. Specifically, for Push we need to say that the new stack state has exactly the same prior array slice contents, ignoring the newly pushed value. For Pop, we need to say that the new state has exactly the prior array slice contents without the old value at the top.

A new function can be used to express these requirements for both Push and Pop:

```
function Unchanged (Invariant_Part, Within : Stack)
  return Boolean;
```

The Within parameter is a stack whose internal state will be compared against that of the Invariant_Part parameter. The name "Invariant_Part" is chosen to indicate the stack state that has not changed. The name "Within" is chosen for readability in named parameter associations on the calls. For example:

```
Unchanged (X, Within => Y)
```

means that the Element values of X should be equal to precisely the corresponding values within Y.

However, this function is not one that users would call directly. We only need it for proof. Therefore, we mark the Unchanged function as a "ghost" function so that the compiler will neither generate code for it nor allow the application code to call it. The function is declared with that aspect (on line 2) as follows:

```
1  function Unchanged (Invariant_Part, Within : Stack)
                                          return Boolean
2   with Ghost;
```

Key to the usage is the fact that by passing This'Old and This to the two parameters we can compare the before/after states of a single object. Viewing the function's implementation will help understand its use in the postconditions:

```
1  function Unchanged (Invariant_Part, Within : Stack)
                                          return Boolean is
2   (Invariant_Part.Top <= Within.Top and then
3     (for all K in 1 .. Invariant_Part.Top =>
4       Within.Values (K) = Invariant_Part.Values (K)));
```

This approach is based directly on a very clever one by Rod Chapman, as seen in some similar code.

The function states that the array components logically contained in Invariant_Part must have the same values as those corresponding array components in Within. Note how we allow Invariant_Part to contain fewer values than the other stack (line 2 above). That is necessary because we use this function in the postconditions for both the Push and Pop operations, in which one more or one less Element value will be present, respectively.

For Push, we add a call to the function in the postcondition as line 6, below:

```
1  procedure Push (This : in out Stack; Item : Element)
   with
2    Pre => not Full (This),
3    Post => not Empty (This)
4      and then Top_Element (This) = Item
5      and then Extent (This) = Extent (This)'Old + 1
6      and then Unchanged (This'Old, Within => This),
7    Global => null;
```

This'Old provides the value of the stack prior to the call of Push, without the new value included, whereas This represents the stack state after Push returns, with the new value in place. Thus, the prior values are compared to the corresponding values in the new state, with the newly included value ignored.

Likewise, we add the function call to the postcondition for Pop, also line 6, below:

```
1  procedure Pop (This : in out Stack; Item : out
     Element)
2    with Pre => not Empty (This),
3    Post => not Full (This)
4      and Item = Top_Element (This)'Old
5      and Extent (This) = Extent (This)'Old - 1
6      and Unchanged (This, Within => This'Old),
7    Global => null;
```

In contrast with procedure Push, on line 6 This and This'Old are passed to the opposite parameters. In this case the new state of the stack, with one less array component logically present, is used as the invariant to compare against. Line 6 expresses the requirement that the new state's content is the same as the old state's content except for the one array component no longer present. Because the function only compares the number of array components within the Invariant_Part, the additional top element value within This'Old is ignored.

Note that we must apply 'Old to This in the calls to Unchanged in both procedures, rather than to some function result. That is unavoidable because we must refer to the prior state of the one stack object being compared.

With those additions to the postconditions we get no further messages from GNATprove from the main procedure, including assertions about the states resulting from a series of calls. We have achieved the Gold level.

Some additional postconditions are possible, however, for completeness. We can also use function Unchanged in a new postcondition for the "=" function:

```
1  function "=" (Left, Right : Stack) return Boolean with
2    Post => "="'Result = (Extent (Left) = Extent (Right)
3      and then Unchanged (Left, Right));
```

This postcondition expresses an implication: whenever the "=" function comparing the two stacks returns True, the Extent (i.e., Top) values will be the same and Unchanged will hold. In other words, they will have the same logical size and content. Whenever "=" returns False, the conjunction will not hold. Note that on line 3, neither argument to function Unchanged has 'Old applied because we are comparing two distinct stack objects, rather than different states for one object. The sizes will be the same (from line 2) so Unchanged will compare the entire slices logically contained by Left and Right.

We can use the same implication approach in a new postcondition for function Empty:

```
function Empty (This : Stack) return Boolean with
  Post => Empty'Result = (Extent (This) = 0);
```

Whenever Empty returns True, Top (i.e., Extent) will be zero, otherwise Top will not be zero.

4.6 Platinum Implementation

Our Gold level implementation also achieved the Platinum level because our postconditions fully covered the functional requirements and there were no abstract properties to be proven. Achieving the Platinum level is rare in itself, all the more so using the Gold level implementation. Doing so is possible in no small part because stacks are simple abstractions.

5 Concluding Remarks

We have shown how to transition an Ada implementation of a sequential, bounded stack abstract data type into a SPARK implementation supporting formal proof of the abstraction's semantics.

Overall, the changes were relatively simple and brief. The truly difficult part of the effort, of course, was determining what changes to make in order to satisfy the provers. That difficulty is somewhat understated in the text because we go directly from specific problems to their solutions, without indicating the time and effort required to identify those solutions. Similarly, we elided parts of the GNATprove messages to highlight the parts indicating the actual problem. Knowing how to interpret the messages, the counterexamples, and possible explanations is a skill that comes with experience.

In addition, we must point out that stacks are simple, especially bounded stacks based on arrays. The relative ease in reaching the Gold or Platinum levels would likely not be possible for other data structures. In particular, a "model" of the abstraction's state will often be required, resulting in complexity well beyond the Unchanged function that was sufficient for bounded stacks. See, for example, the formal containers shipped with GNAT.

Thanks are due to Yannick Moy and the entire SPARK team at AdaCore for their essential help. The source code and full GNAT project are available on GitHub here.

6 Gold/Platinum Implementation Listing

The following is the generic package declaration and body for the Platinum level implementation. As described earlier, the Platinum level implementation is the same as the Gold level implementation. We have kept the two versions in separate packages and files. The Platinum version, like the Gold version, did not include the Depends contracts. Listings 5 and 6 show a version with those contracts, for completeness.

```
generic
  type Element is private;
  -- The type of values contained by objects of type Stack
  Default_Value : Element;
  -- The default value used for stack contents. Never
  -- acquired as a value from the API, but required for
  -- initialization in SPARK.
package Bounded_Stacks_Platinum is
  pragma Unevaluated_Use_of_Old (Allow);
  subtype Element_Count is Integer range 0 .. Integer'Last - 1;
  -- The number of Element values currently contained
  -- within any given stack. The lower bound is zero
  -- because a stack can be empty. We limit the upper
  -- bound (minimally) to preclude overflow issues.
  subtype Physical_Capacity is
    Element_Count range 1 .. Element_Count'Last;
  -- The range of values that any given stack object can
  -- specify (via the discriminant) for the number of
  -- Element values the object can physically contain.
  -- Must be at least one.
  type Stack (Capacity : Physical_Capacity) is private
    with Default_Initial_Condition => Empty (Stack);
  procedure Push (This : in out Stack; Item : Element) with
    Pre => not Full (This),
    Post => not Empty (This)
      and then Top_Element (This) = Item
      and then Extent (This) = Extent (This)'Old + 1
      and then Unchanged (This'Old, Within => This),
    Global => null;
  procedure Pop (This : in out Stack; Item : out Element) with
    Pre => not Empty (This),
    Post => not Full (This)
      and Item = Top_Element (This)'Old
      and Extent (This) = Extent (This)'Old - 1
      and Unchanged (This, Within => This'Old),
    Global => null;
  function Top_Element (This : Stack) return Element with
    Pre => not Empty (This),
    Global => null;
  -- Returns the value of the Element at the "top" of This
  -- stack, i.e., the most recent Element pushed. Does not
  -- remove that Element or alter the state of This stack
  -- in any way.
  overriding function "=" (Left, Right : Stack) return Boolean with
    Post => "="'Result = (Extent (Left) = Extent (Right)
      and then Unchanged (Left, Right)),
    Global => null;
  procedure Copy (Destination : in out Stack; Source : Stack) with
    Pre => Destination.Capacity >= Extent (Source),
    Post => Destination = Source,
    Global => null;
  -- An alternative to predefined assignment that does not
  -- copy all the values unless necessary. It only copies
  -- the part "logically" contained, so is more efficient
  -- when Source is not full.
```

```
  function Extent (This : Stack) return Element_Count with
    Global => null;
  --  Returns the number of Element values currently
  --  contained within This stack.
  function Empty (This : Stack) return Boolean with
    Post => Empty'Result = (Extent (This) = 0),
    Global => null;
  function Full (This : Stack) return Boolean with
    Post => Full'Result = (Extent (This) = This.Capacity),
    Global => null;
  procedure Reset (This : in out Stack) with
    Post => Empty (This),
    Global => null;
  function Unchanged (Invariant_Part, Within : Stack) return Boolean
    with Ghost;
  --  Returns whether the Element values of Invariant_Part
  --  are unchanged in the stack Within, e.g., that inserting
  --  or removing an Element value does not change the other
  --  Element values held.
private
  type Content is array (Physical_Capacity range <>) of Element;
  type Stack (Capacity : Physical_Capacity) is record
    Values : Content (1 .. Capacity) := (others => Default_Value);
    Top : Element_Count := 0;
  end record with
    Predicate => Top in 0 .. Capacity;
  --------------
  -- Extent --
  --------------
  function Extent (This : Stack) return Element_Count is
    (This.Top);
  --------------
  -- Empty --
  --------------
  function Empty (This : Stack) return Boolean is
    (This.Top = 0);
  -----------
  -- Full --
  -----------
  function Full (This : Stack) return Boolean is
    (This.Top = This.Capacity);
  ------------------------
  -- Top_Element --
  ------------------------
  function Top_Element (This : Stack) return Element is
    (This.Values (This.Top));
  ----------
  -- "=" --
  ----------
  function "=" (Left, Right : Stack) return Boolean is
    (Left.Top = Right.Top and then
     Left.Values (1 .. Left.Top) = Right.Values (1 .. Right.Top));
  ---------------------
  -- Unchanged --
  ---------------------
  function Unchanged (Invariant_Part, Within : Stack) return Boolean is
    (Invariant_Part.Top <= Within.Top and then
      (for all K in 1 .. Invariant_Part.Top =>
        Within.Values (K) = Invariant_Part.Values (K)));
end Bounded_Stacks_Platinum;
```

Listing 5. Platinum Version.

```
package body Bounded_Stacks_Platinum is
  --------------
  -- Reset --
  --------------
  procedure Reset (This : in out Stack) is
  begin
     This.Top := 0;
  end Reset;
  -------------
  -- Push --
  -------------
  procedure Push (This : in out Stack; Item : Element) is
  begin
    This.Top := This.Top + 1;
    This.Values (This.Top) := Item;
  end Push;
  ------------
  -- Pop --
  ------------
  procedure Pop (This : in out Stack; Item : out Element) is
  begin
    Item := This.Values (This.Top);
    This.Top := This.Top - 1;
  end Pop;
  -------------
  -- Copy --
  -------------
  procedure Copy (Destination : in out Stack; Source : Stack) is
    subtype Contained is Element_Count range 1 .. Source.Top;
  begin
    Destination.Top := Source.Top;
    Destination.Values (Contained) := Source.Values (Contained);
  end Copy;
end Bounded_Stacks_Platinum;
```

Listing 6. Platinum Version Body.

A Layered Mapping of Ada 202X to OpenMP

S. Tucker Taft
AdaCore, Lexington, MA

1 Introduction

The OpenMP specification defines a set of compiler directives, library routines, and environment variables that together represent the OpenMP Application Programming Interface, and is currently defined for C, C++, and Fortran. The forthcoming version of Ada, currently dubbed Ada 202X, includes lightweight parallelism features, in particular parallel blocks and parallel loops. All versions of Ada, since its inception in 1983, have included “tasking,” which corresponds to what are traditionally considered “heavyweight” parallelism features, or simply “concurrency” features. Ada “tasks” typically map to what are called “kernel threads,” in that the operating system manages them and schedules them. However, one of the goals of lightweight parallelism is to reduce overhead by doing more of the management outside the kernel of the operating system, using a light-weight-thread (LWT) scheduler. The OpenMP library routines support both levels of threading, but for Ada 202X, the main interest is in making use of OpenMP for its lightweight thread scheduling capabilities.

For C, C++, and Fortran, the programmer is fully aware when they are making use of OpenMP for any lightweight parallelism features, in that they use OpenMP-specific compiler directives and in some cases explicit calls on the OpenMP library, to implement their program. By contrast, for Ada 202X, the language defines the lightweight parallelism features, and our goal is to enable the implementation of those features on top of OpenMP, or on top of some alternative LWT scheduler, or as a fallback, with effectively sequential semantics. We anticipate the desire to pass options to the underlying LWT scheduler, be it OpenMP or some other infrastructure, but such options would be intended to only affect performance, with no effect on the fundamental dynamic semantics.

Given the above goals, we are recommending a layered mapping to OpenMP (or other LWT scheduler), where upon seeing the syntax for a parallel construct, the compiler generates calls on a top layer (dubbed "System.Parallelism" for now). This layer is independent of the particular LWT scheduler that will be controlling the light-weight threads that are spawned as a result of the parallel constructs. Below System.Parallelism is a package dubbed "System.LWT", which provides the LWT-scheduler-independent API, and implements it using a "plug-in" architecture. Specific LWT schedulers would be children of this package, for example "System.LWT.OpenMP", and one of them could be "plugged in" to System.LWT and handle the various calls through the API. In the absence of any plugin, System.LWT would fall back to a purely sequential implementation.

The user will determine which particular LWT scheduler, if any, gets linked into the program by mentioning in a “with” clause a package (e.g. Interfaces.OpenMP) and declaring a "control" object of a type that was declared in that package (e.g. Interfaces.OpenMP.OMP_Parallel), in the task body for the Ada tasks (or the main subprogram for the environment task) where it is desired to have multiple light weight threads of control. Data within the control object can be used to control the level of parallelism desired (e.g. "Control : OMP_Parallel (Num_Threads => 5);"), as well as potentially other options that should apply by default across all parallel constructs in the given Ada task. This approach is modeled on the "#pragma omp parallel" of OpenMP which creates a "parallel region" in which work-sharing or other forms of parallelism can be used. The Interfaces.OpenMP package might have other subprograms intended to be called directly by the user, in particular those that are part of the "official" OpenMP API, such as "omp_get_thread_num" or "omp_get_team_size,” though there would be no requirement to call such subprograms in normal operation of a parallel program.

2 Interaction with Ada tasks

As mentioned above, Ada has always had heavier weight “tasks” to provide basic concurrency, where tasks are defined by creating an object of a given task type. The Ada program as a whole always represents another task, which is called the environment task. We propose that each such Ada task defines its own parallelism region, if any, recognizing that many Ada tasks will need no internal parallelism, and might continue to serve special roles in a real-time environment, such as managing particular hardware devices, or specific jobs of the real-time system, at a particular real-time priority.

For each Ada task that does want internal parallelism, the expectation is that the underlying LWT scheduler (e.g. OpenMP) will start up (or re-use) additional (heavy-weight) "kernel threads" to act as servers for the LWTs that will be spawned somewhere within the Ada task. Each of these servers will run at the same priority as the enclosing Ada task, and will share the Ada "identity" and "attributes" of that Ada task from the point of view of the Ada semantics. The LWTs served by these server threads will in turn get their Ada task "identity" and "attributes" from these servers.

3 Light-weight thread groups

Each light-weight thread is run in the context of an "LWT group," which maps quite directly to an OpenMP "taskgroup." All of the LWTs spawned during the scope of an LWT group are automatically awaited at the end of the LWT group. This ensures that the scope where an LWT is defined isn't exited while such LWTs are still running and potentially making up-level references to objects from the enclosing scope. When the Ada 202X compiler sees a parallel construct, it will emit code to create an LWT group, and then call the appropriate sequence of System.Parallelism and System.LWT routines at appropriate points, and then emit code to wait for the group to complete before proceeding past the construct.

4 OpenMP options

One challenge is how the Ada programmer, and the Ada compiler, can pass options through the LWT layer down to the underlying thread scheduler. The general idea is to allow for a variant of Ada's standard "aspect" syntax (similar to what are called "annotations" in other languages) any place where the parallel reserved word is used to initiate a parallel construct, and allow the syntax to be generalized by specifying a value of any type that is an extension of a special Ada.Aspects.Root_Aspect tagged type. This value can be passed through the various System.Parallelism and System.LWT APIs, allowing the underlying LWT scheduler to receive the options and use them as it sees fit, with a default value of Null_Aspect. The advantage of this approach is that a user-provided LWT scheduler might be substituted for one provided by the compiler vendor, and it could also take advantage of the generalized aspect syntax without any need to add special handling into the compiler. This also allows us to reach additional options for some future OpenMP standard without further additions to the compiler.

5 Conclusion

This layered approach, along with the parallelism control object and the generalized aspects defined by types from the Interface.OpenMP package, should allow Ada 202X users to take advantage of the OpenMP features of interest, and to accommodate evolution of the OpenMP standard as well as the ability to use other LWT schedulers which might come from, say, an RTOS vendor. If the user chooses to use no LWT scheduler, a sequential fall back will be part of System.LWT whenever there is no LWT scheduler "plugged in." We believe this layered approach may be a model for other languages that want to provide a binding to OpenMP capabilities, while not requiring heavy use of compiler directives, which can hurt readability and do not tend to be as composable as syntax.

Appendix

Handling secondary stack, exceptions, and transfers of control out of the code for a light-weight thread

Independent of which particular LWT scheduler is present (if any), the code for a particular light-weight thread is defined by a function pointer and a data object. For Ada, the data object will typically be a tagged data object, and the function will be a simple wrapper that merely invokes a special "LWT_Body" dispatching operation on the object, and handles all exceptions propagated by the body (similar to the way a wrapper around an Ada task body handles all exceptions). Normal secondary stack and exception raising and handling can be performed inside the LWT_Body, because light-weight threads run on a given server until completion or cancelation. They aren't swapped back and forth, so there is no added complexity in stack or exception management. Effectively, exceptions are being raised and handled on the stack of the server, in the usual way.

When an exception is propagated out of an LWT_Body, or if the code for an LWT_Body has an explicit transfer of control out of the code for the light-weight thread, an attempt is made to cancel the other threads in the LWT group. The first LWT to attempt cancelation receives an indication of "success" from this attempt. Later LWTs of the same group making such an attempt will not receive the "success" indicator. Cancelation in OpenMP, and in Ada 202X, allows cancelation to be implemented using a polling approach, where there are various well-defined "cancelation points." When code within LWT_Body detects that the enclosing LWT group has been canceled, it generally just returns from the LWT_Body. The LWT that successfully initiated the cancelation records in a variable visible at the point where the LWT group ends, what action should be taken. The code at the end of the LWT group propagates the exception, or continues any other transfer of control, after waiting for all of the LWTs within the group to complete their execution.

Expansions for proposed Ada 202X features

A. Expansion for Ada 202X parallel block

The OpenMP recommended approach to supporting a sequence of blocks to be (potentially) run in parallel is to create a loop around a switch/case statement. The "GOMP" implementation indicates the same approach in:

https://gcc.gnu.org/onlinedocs/libgomp/Implementing-SECTIONS-construct.html

We suggest the same approach for Ada. Here is an example expansion:

```
procedure Walk (Tree : Tree_Ptr) is
  -- Walk nodes of Tree in parallel
begin
  if Tree = null then
    return;
  elsif Tree.Kind = Leaf then
    Process (Tree);
  else
    parallel
    do
      Walk (Tree.Left);
    and
      Walk (Tree.Right);
    end do;
  end if;
end Walk;
```

expands into:

```
procedure Walk (Tree : Tree_Ptr) is
  -- Walk nodes of Tree in parallel
begin
  if Tree = null then
    return;
  elsif Tree.Kind = Leaf then
    Process (Tree);
  else
    parallel
    for _I in 1 .. 2 loop
      case _I is
      when 1 =>
        Walk (Tree.Left);
      when 2 =>
        Walk (Tree.Right);
      end case;
    end loop;
  end if;
end Walk;
```

which then expands further as a parallel loop, as described below. This approach makes it easy to turn parallelism on and off, and requires the creation of only one out-of-line procedure independent of the number of arms in the parallel block statement.

B. Expansions for Ada 202X parallel loop

In this description, we show an optional intermediate step where the compiler might use a pragma Par_Loop so that parallel loops could be specified while remaining compilable by older Ada compilers, analogous to the way the "Pre" aspect expands into a “pragma Precondition” in the GNAT compiler.

Ada 202X defines the notion of a “chunk specification” which can give a user-specified name to the index used to identify a chunk. When using a pragma instead of syntax, there would be no way to specify the chunk-index name, so the value of the chunk index can be referenced when inside the body of a parallel loop by calling the intrinsic parameterless function System.Parallelism.Chunk_Index, which will always return 1 in a version of the System.Parallelism package for use with sequential-only implementations. If there is an explicit "chunk parameter" in the chunk specification, references to the chunk parameter could be replaced by a computation based on the result of a call to this intrinsic Chunk_Index function.

```
parallel (Num_Chunks)
for ... loop
  <loop body>
end loop;
```

expands into:

```
pragma Par_Loop(Num_Chunks);
for ... loop
  <loop body>
end loop;
```

which expands further according to the kind of for-loop immediately following the pragma Par_Loop:

(1) Parallel loop over a range of values:

```
pragma Par_Loop(Num_Chunks);
for I in S range A..B loop
  <loop body>
end loop;
```

expands into:

```
declare
  procedure I__Loop_Body
    (I__Low, I__High : Longest_Integer;
     I__Chunk_Index : Positive) is
  begin
    for I in S'Val (I__Low) .. S'Val (I__High) loop
      <loop body>
    end loop;
  end I__Loop_Body;
begin
    System.Parallelism.Par_Range_Loop
      (S'Pos(A), S'Pos(B), Num_Chunks,
       Loop_Body => I__Loop_Body'Access);
end;
```

(2) Parallel loop over an array:

```
pragma Par_Loop(Num_Chunks);
for C of Arr loop
  <loop body>
end loop;
```

expands into:

```
pragma Par_Loop(Num_Chunks);
for C__Index in Arr'Range loop
  declare
    C renames Arr(C__Index);
  begin
    <loop body>
  end;
end loop;
```

which then expands according to expansion (1) above for a loop over a range. Note that a loop over a multidimensional array would be transformed effectively into a loop over a conceptually flattened array, as is done in the sequential loop case.

(3) Parallel loop over a generalized iterator:

```
pragma Par_Loop(Num_Chunks);
for C of Iterator loop
  <loop body>
end loop;
```

expands into:

```
declare
  package Inst renames <some instantiation of
                        Ada.Iterator_Interfaces>;
  package Par_Iterator_Inst is new
                        Inst.Par_Iterator_Loop;
  procedure C__Loop_Body
    (C__Iterator : Inst.Parallel_Iterator'Class;
     C__Chunk_Index : Positive) is
      C : Inst.Cursor :=
            C__Iterator.First (C__Chunk_Index);
  begin
    while Inst.Has_Element(C) loop
      <loop body>
      C := C__Iterator.Next (C, C__Chunk_Index);
    end loop;
  end C__Loop_Body;
begin
  Par_Iterator_Inst
    (Iterator, Num_Chunks, Loop_Body =>
                        C__Loop_Body'Access);
end;
```

(4) Parallel loop over a container:

```
pragma Par_Loop(Num_Chunks);
for E of Container loop
   <loop body>
end loop;
```

expands into:

```
pragma Par_Loop(Num_Chunks);
for E__Cursor of
          Container'Default_Iterator(Container) loop
 declare
  E renames Container(E__Cursor);
 begin
  <loop body>
 end;
end loop;
```

which then expands according to (3) above for a loop over an iterator.

C. System.Parallelism package and Ada.Iterator_Interfaces.Par_Iterator_Loop child

The System.Parallelism package spec might contain (at least) the following:

```
package System.Parallelism is
 type Longest_Integer is range
                    System.Min_Int .. System.Max_Int;
  -- Not worrying about unsigned ranges with
  -- upper bound > System.Max_Int for now.
  -- Could be handled by having a version of
  -- Par_Range_Loop that operates on
  -- unsigned integers.
 procedure Par_Range_Loop
  (Low, High : Longest_Integer;
   Num_Chunks : Positive;
   Aspects : access Ada.Aspects.Root_Aspect'Class :=
                                              null;
   Loop_Body : access procedure
                    (Low, High : Longest_Integer;
                     Chunk_Index : Positive));
 function Chunk_Index return Positive
  with Convention => Intrinsic;
end System.Parallelism;
```

A child unit of Ada.Iterator_Interfaces, Par_Iterator_Loop, could be provided as follows. Note that the Ada 202X compiler might want to instantiate this just once for each instantiation of Ada.Iterator_Interfaces, rather than at each parallel loop over an iterator.

```
generic
procedure Ada.Iterator_Interfaces.Par_Iterator_Loop is
 (Iterator : Parallel_Iterator'Class;
  Num_Chunks : Positive;
  Aspects : access Ada.Aspects.Root_Aspect'Class :=
                                              null;
  Loop_Body : access procedure
                 (Iterator : Parallel_Iterator'Class;
                  Chunk_Index : Positive));
```

Parallel Software to Offset the Cost of Higher Precision*

Jan Verschelde

University of Illinois at Chicago, Department of Mathematics, Statistics, and Computer Science, 851 S. Morgan St. (m/c 249), Chicago, IL 60607-7045; email: janv@uic.edu; URL: http://www.math.uic.edu/∼jan

Abstract

Hardware double precision is often insufficient to solve large scientific problems accurately. Computing in higher precision defined by software causes significant computational overhead. The application of parallel algorithms compensates for this overhead. Newton's method to develop power series expansions of algebraic space curves is the use case for this application.

1 Problem Statement and Overview

While parallel computers are fast and can solve large problems, the propagation of roundoff errors increases as problems grow larger and the hardware supports only double precision. If we can afford the same time as on a sequential run, then we ask for *quality up*: by how much can we improve the quality of the results with a parallel run? To us, quality means accuracy. The goal is to compensate for the overhead of multiple double arithmetic with parallel computations.

The focus of this paper is on recently developed code for new algorithms described in [3], [12, 13], added to PHCpack [14]. PHCpack is a free and open source package to apply Polynomial Homotopy Continuation to solve systems of many polynomials in several variables. Continuation methods are classic algorithms in applied mathematics, see e.g. [9]. Ada is the main language in which the algorithms in PHCpack have been developed during the past thirty years. Strong typing and standardization make that the same code runs on different platforms (Linux, Windows, Mac OS X) and that the same code continues to run, even after decades, without the need to update for upgrades of the language. Ada tasking provides an effective high level tool to develop algorithms for parallel shared memory computers; see [2] or [8] for introductions.

Using QDlib [6] and the software CAMPARY [7], we extend the range of precision offered by hardware doubles [10], as a step towards rigorous verification. In our numerical study of algebraic curves [15], we apply algorithmic differentiation [5], numerical linear algebra [4], and rational approximation techniques [1].

The first three sections in this paper motivate the need for higher precision and describe the computational cost overhead. This overhead then motivates the application of multitasking. All computational experiments for this paper were done on a CentOS Linux workstation with 256 GB RAM and two 22-core 2.2 GHz Intel Xeon E5-2699 processors.

*Supported by the National Science Foundation under grant DMS 1854513.

2 Multiple Double Numbers

A double double is an unevaluated sum of two hardware doubles. With the application of basic arithmetical operations in IEEE double format, we obtain more accurate results, up to twice the accuracy of the hardware double precision. In [11], double double arithmetic is described in the context of error-free transformations; see also [10, Chapter 14]. Double double and quad double arithmetic is provided by QDlib [6]. Code generators for general multiple double and multiple float arithmetical operations are available in the software CAMPARY [7].

As an illustration of multiple double arithmetic, consider the computation of the 2-norm of a vectors of dimension 64 of complex numbers generated as $\cos(\theta) + \sin(\theta)\sqrt{-1}$, for random angles θ. The 2-norm equals 8. Observe in Table 1 the second double of the multiple double 2-norm.

The format of the result shown in Table 1 is for this experiment preferable over the decimal expansion which may appear as $7.999\ldots9$. The Ada code for multiple double precision is available in the free and open source software PHCpack [14], under version control at github.

Table 2 shows the cost of the basic operations in multiple double precision, expressed in the number of hardware double arithmetical operations. A tenfold increase in precision from double to deca double leads to a more than thousandfold increase in the count of the arithmetical operations.

The operation counts in Table 2 then motivate the need for parallel computations as follows. What takes a millisecond to compute in double precision will take several seconds in deca double precision. A program that finishes in a second in double precision will take more than an hour in deca double precision. A computation in double precision that takes a hour will in deca double precision take more than a month to finish.

3 Polynomials as Truncated Power Series

Writing a polynomial backwards (starting at the constant term and then listing the monomials in the increasing degree order), leads to the interpretation of a polynomial as the sum of the leading terms in a power series. Unlike polynomials, every power series with a leading nonzero constant term has an

```
double double : 8.00000000000000E+00 - 6.47112461314111E-32
triple double : 8.00000000000000E+00 + 1.78941597340672E-48
  quad double : 8.00000000000000E+00 + 3.20475411419393E-65
 penta double : 8.00000000000000E+00 + 2.24021706293649E-81
  octo double : 8.00000000000000E+00 - 9.72609915198313E-129
  deca double : 8.00000000000000E+00 + 3.05130075600701E-161
```

Table 1: Illustration of multiple double arithmetic.

	double double				triple double				quad double			
	+	−	∗	/	+	−	∗	/	+	−	∗	/
add	8	12			13	22			35	54		
mul	5	9	9		83	84	42		99	164	73	
div	33	18	16	3	113	214	63	4	266	510	112	5
	penta double				octo double				deca double			
	+	−	∗	/	+	−	∗	/	+	−	∗	/
add	44	78			95	174			139	258		
mul	162	283	109		529	954	259		952	1743	394	
div	474	898	175	6	1599	3070	448	9	2899	5598	700	11

Table 2: Number of double operations for addition (add), multiplication (mul), division (div), required for a 2-fold, 3-fold, 4-fold, 5-fold, 8-fold, and 10-fold increase in precision.

inverse. One can divide power series by another and calculate with power series similar as to number arithmetic [15]. In this section, we consider Newton's method where the arithmetic happens with truncated power series instead of with ordinary numbers.

One common parameter representation for points on the circle with radius one is $(\cos(t), \sin(t))$, for $t \in [0, 2\pi[$. With truncated power series arithmetic we can approximate this representation. Consider a system of two polynomials in three variables:

$$\begin{cases} t - 1/6t^3 + 1/120t^5 - 1/5040t^7 - y = 0 \\ x^2 + y^2 - 1 = 0. \end{cases}$$

The first polynomial represents the equation $y = t - 1/6t^3 + 1/120t^5 - 1/5040t^7$. The right hand side of this equation contains the first four leading terms of the Taylor expansion of $\sin(t)$.

Given the leading terms of $\sin(t)$, running Newton's method, with 8 as the truncation degree of the power series, starting at $x = 1$, $y = 0$, and $t = 0$, the leading terms of $\cos(t)$ will appear as the solution series for x. Indeed, the numerical output contains

```
2.48015873015868E-05*t^8
- 1.38888888888889E-03*t^6
+ 4.16666666666667E-02*t^4
- 5.00000000000000E-01*t^2 + 1.
```

The second polynomial has floating-point coefficients which approximate the Taylor series of the $\cos(t)$, in particular $x = 1 - 1/2t^2 + 1/24t^4 - 1/720t^6 + 1/40320t^8$. Although many programmers will experience the temptation to display `5.00000000000000E-01` as `1/2`, the `7` in the number `4.16666666666667E-02` gives an indication about the size of the roundoff error. This information would be lost if one would display the result by the nearest rational number `1/24`.

Looking at polynomials as truncated power series has the benefit that the solver can handle larger classes of nonlinear systems, as the first equation of the above polynomial system can be viewed as an approximation for $\sin(t) - y = 0$. With truncated power series as coefficients, the solutions of systems where the number of variables is one more than the number of equations are also power series. Although the convergence radius of power series can be limited, power series serve as input to compute highly accurate rational approximations for functions [1].

Even as the above calculation was performed in double precision, Newton's method did not run on vectors of numbers, but on vectors of truncated power series, represented as power series with vector coefficients. Working with truncated power series causes an extra cost overhead and provides an additional motivation for parallel computations. In particular, the multiplication of two power series truncated to degree d requires $(d + 2)(d + 1)/2$ multiplications and $(d + 1)d/2$ additions. For a modest degree $d = 8$, the formulas in the previous sentence evaluate to 45 and 36. For $d = 32$ the corresponding numbers are 561 and 528. These numbers predict the cost overhead factors in working with truncated power series arithmetic.

Working with power series of increasing degrees of truncation leads to more roundoff and requires therefore arithmetic in higher precision, as will be made explicit in the next section.

4 Newton's Method on Truncated Power Series

In this section we make our problem statement more precise. In particular, running a lower triangular block Toeplitz solver results in a loss of accuracy.

One step of Newton's method requires evaluation and differentiation of the system, followed by the solution of a linear system. Consider $\mathbf{f}(\mathbf{x}) = \mathbf{0}$ as a system of poly-

nomials in several variables, with coefficients as truncated power series in the variable t, where $\mathbf{f} = (f_1, f_2, \ldots, f_N)$, $\mathbf{x} = (x_1, x_2, \ldots, x_n)$, and $N \geq n$. For $N > n$, the linear systems are solved in the least squares sense, either with QR or SVD; for $N = n$, an LU factorization can be applied; see [4] for an introduction to matrix factorizations.

Then we compute $\mathbf{x}(t)$ a power series solution to $\mathbf{f}(\mathbf{x}) = \mathbf{0}$, starting at a point $\mathbf{x}(0) = \mathbf{z}$, $\mathbf{x}(t) = \mathbf{z} + \mathbf{x}_1 t + \mathbf{x}_2 t^2 + \cdots$. With linearization [3], instead of vectors and matrices of power series, we consider power series with vectors and matrices as coefficients. A matrix is denoted with a capitalized letter, e.g.: A; vectors are denoted in bold, e.g.: $\mathbf{x}$, $\mathbf{b}$. To compute the update $\Delta\mathbf{x}$ to the solution in Newton's method, a linear system is solved. With truncated power series arithmetic, this linear system is written in short as $\mathbf{A}(t)\Delta\mathbf{x}(t) = \mathbf{b}(t)$. The given coefficients in this equation $\mathbf{A}$ and $\mathbf{b}$, where $\mathbf{A}$ is a vector of matrices $\mathbf{A} = (A_0, A_1, \ldots, A_d)$ and $\mathbf{b}$ is a vector of power series.

In linearized format, for truncation degree d, $\mathbf{A}(t)\Delta\mathbf{x}(t) = \mathbf{b}(t)$ represents

$$\begin{aligned} & \left(A_0 + A_1 t + A_2 t^2 + \cdots + A_d t^d\right) \\ & \left(\Delta\mathbf{x}_0 + \Delta\mathbf{x}_1 t + \Delta\mathbf{x}_2 t^2 + \cdots + \Delta\mathbf{x}_d t^d\right) \\ = \; & \mathbf{b}_0 + \mathbf{b}_1 t + \mathbf{b}_2 t^2 + \cdots + \mathbf{b}_d t^d. \end{aligned}$$

The A_0 is the matrix of all partial derivatives of the polynomials in $\mathbf{f}$ at the leading constant coefficient of the power series expansion of the solution vector. Methods of algorithmic differentiation [5] lead to an efficient calculation of $\mathbf{A}(t)$. In particular, computing all n partial derivatives of a function f in n variables requires about 3 (and not n) times the cost to evaluate f.

Expanding the multiplication and rearranging the terms according to the powers of t leads to a lower triangular block system:

$$\begin{bmatrix} A_0 & & & & \\ A_1 & A_0 & & & \\ A_2 & A_1 & A_0 & & \\ \vdots & \vdots & \vdots & \ddots & \\ A_d & A_{d-1} & A_{d-2} & \cdots & A_0 \end{bmatrix} \begin{bmatrix} \Delta\mathbf{x}_0 \\ \Delta\mathbf{x}_1 \\ \Delta\mathbf{x}_2 \\ \vdots \\ \Delta\mathbf{x}_d \end{bmatrix} = \begin{bmatrix} \mathbf{b}_0 \\ \mathbf{b}_1 \\ \mathbf{b}_2 \\ \vdots \\ \mathbf{b}_d \end{bmatrix}.$$

Forward substitution is applied as follows. First solve $A_0\Delta\mathbf{x}_0 = \mathbf{b}_0$. Once $\Delta\mathbf{x}_0$ is known, The second equation $A_1\Delta\mathbf{x}_0 + A_0\Delta\mathbf{x}_1 = \mathbf{b}_1$ then becomes $A_0\Delta\mathbf{x}_1 = \mathbf{b}_1 - A_1\Delta\mathbf{x}_0$. After the computation of $\Delta\mathbf{x}_1$, the third equation $A_2\Delta\mathbf{x}_0 + A_1\Delta\mathbf{x}_1 + A_0\Delta\mathbf{x}_2 = \mathbf{b}_2$ turns into the equation $A_0\Delta\mathbf{x}_2 = \mathbf{b}_2 - A_2\Delta\mathbf{x}_0 - A_1\Delta\mathbf{x}_1$, etc.

The exploitation of the block structure reduces the computation of d linear systems with the same N-by-n coefficient matrix A_0. Suppose that in each step up to two decimal places of accuracy would be lost, then the accuracy loss of the last $\Delta\mathbf{x}_d$ could be as large as $2d$. Even with a modest degree of 8, in double precision, this would imply that all 16 decimal places of accuracy are lost. A loss of 16 decimal places of accuracy in double double precision still leads to sufficiently accurate results. This argument is expressed formally in [13].

5 Application of Multitasking

In multiple double arithmetic, programs become compute bound, which is beneficial on computers with faster processor speed than memory speed. On a parallel shared memory computer, multiple threads run within one process. In the type of multitasking applied in this paper, each task is mapped to one kernel thread. Typically the total number of tasks in each parallel run should then not exceed (twice) the number of available cores on the processor.

Current processors run at the speed of a couple of GHz and have thus a theoretical peak performance of one billion floating-point operations per second. One billion is 10^9 or $1,000 \times 1,000 \times 1,000$. The first two thousands in this billion represent roughly the overhead caused by multiple double and power series arithmetic, with the last thousand the original computational cost in double precision. This rough estimate explains that one job will typically take several seconds and is relatively much larger than the cost of launching several threads.

All data is allocated and defined before the threads are launched. Figure 1 illustrates the design of a job queue with 8 jobs and 4 finished jobs, as the counter counts the number of finished jobs. The arrows in the picture point to the read only input data and the work space for each job. While the input may point to shared data, each job has distinct, non overlapping memory locations for the work space needed for each job. When a task needs to work on the next job, it requests entry to the binary semaphore that guards the counter for the next job. Once entry is granted, the tasks increments the value of the counter and releases the lock. The time spent inside a critical section is thus minimal.

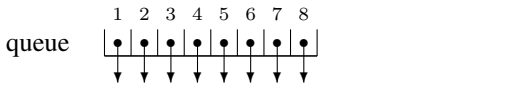


Figure 1: Schematic of a job queue with a counter guarded by a binary semaphore.

The organization of all computational work into a job queue determines the granularity of the parallelism. In the application of running Newton's method, medium grained parallelism is applied. For the evaluation and differentiation of a system of polynomials, one job is concerned with one polynomial. For the solving of the block triangular linear system, one job is the update of one right hand side vector after the computation of one update vector. The solution of the linear system can happen only after all polynomials in the system are evaluated and differentiated. The synchronization between those two stages is performed by terminating all tasks and launching a new set of tasks for the next stage. Details are described in [13].

The main executable in PHCpack is `phc`. The user can specify the number of tasks at the command line, e.g., call the solver with eight tasks as `phc -b -t8`. If no number follows the `-t`, then the number of tasks equals the number

of available kernel threads. Below is the output of `time phc -b` and `time phc -b2 -t`, respectively in double and double double precision, on the cyclic 7-roots benchmark, using 88 threads.

```
real    0m10.310s    real    0m1.661s
user    0m10.188s    user    1m12.226s
sys     0m0.008s     sys     0m0.083s
```

The numbers after `real` are the elapsed wall clock time. With multitasking, the speedup in double double over double precision is $10.310/1.661 \approx 6.2$. We have speedup and quality up.

6 A Numerical Experiment using Multiple Doubles

At the end of [13], we reported an instance where quad double precision was insufficient for Newton's method to converge and compute the coefficients of the series past degree 15.

Details for the experiment can be found in [13], a short summary follows. The series development start at a generic point on a 7-dimensional surface of cyclic 128-roots, defined by a polynomial system of 128 polynomials in 128 variables, augmented with seven linear equations. To every equation in the system, a parameter t is added. At $t = 0$, the generic point on the 7-dimensional surface is then the leading coefficient vector of the power series expansion of the solution curve in t.

For this problem, the inverse of the condition number of the matrix A_0 is estimated at $4.6\text{E}{-6}$, which implies that up to six decimal places of accuracy may be lost in the computation of the next term of the power series. The accuracy of the power series is measured by $\|\Delta\mathbf{x}\|$, the modulus of the update to the last coefficient in the power series. The tolerance on $\|\Delta\mathbf{x}\|$ for all runs is set to $1.0\text{E}{-32}$. Newton's method stops when $\|\Delta\mathbf{x}\| \leq 1.0\text{E}{-32}$ or when the number of steps has exceeded the maximum number of iterations. The maximum number of iterations with Newton's method is as many as as 8, 8, 12, and 16, for the respective degrees 8, 16, 24, and 32 of the power series.

Table 3 summarizes the data of the numerical experiment with `phc -u -t`. Once $\|\Delta\mathbf{x}\|$ is too large for one degree, computations for the next degree are not done.

For degree 8, the computations with penta doubles finish in 10 seconds sooner than the computations with quad doubles, because 5 iterations suffice. For degree 16, the results in deca double precision are much more accurate than in octo double precision, with the same number of iterations. Adding up all seconds in Table 3 gives 18,717 seconds, or 5 hours, 11 minutes, and 57 seconds. Without parallel software, this experiment would have taken more than 100 hours, more than 4 days. The multiplication factor of 20 is derived from the efficiency study in the next section. Obviously, parallel software saves time when running numerical experiments.

7 Computational Results

The runs are done on a CentOS Linux workstation with 256 GB RAM and two 22-core 2.2 GHz Intel Xeon E5-2699 processors. If one is mainly interested in the fastest throughput, then with hyperthreading, runs could be done with 88 threads. However, the effect of hyperthreading is not equivalent to doubling the number of cores. In the practical evaluation of the parallel implementation, the runs therefore stop at 40 worker threads.

Random polynomial systems are generated, 64 polynomials with 64 monomials per polynomial. Power series are truncated to degrees 8, 16, and 32. Efficiencies are reported for 2, 4, 8, 16, 32, and 40 worker threads. Efficiency is speedup divided by the number of worker threads.

The plots in Figure 2 show efficiencies for degrees 8 and 16 of the truncated power series. The efficiencies decrease from close to 100% (a near perfect speedup for 2 threads) to below 60% when 40 worker threads are used. As efficiency equals speedup divided by the number of worker threads, the speedup corresponding to 60% efficiency for 40 worker threads equals $0.6 \times 40 = 24$.

The plots in Figure 3 compare the efficiencies for degrees 16 and 32. For truncation degree 32, we observe that 60% efficiency is reached already at triple double precision. More extensive numerical experiments would increase the number of polynomials and the number of monomials per polynomial to investigate the notion of isoefficiency. In particular, by how much should the size of the problem increase to obtain the same efficiency as the number of threads increases?

The computational results of this section (the 60% efficiency or the 24 speedup) justify the multiplication factor of 20 used in the last paragraph of section 6.

8 Conclusions

This paper presents a use case of multiple double precision in the application of Newton's method to develop power series expansions for solution curves of polynomial systems. The experiments described in this paper are performed by recent additions to the free and open source software PHCpack, available via github.

PHCpack contains an Ada version of the code in QDlib [6], for double double and quad double precision, and of code generated by the software CAMPARY [7], for triple, penta, octo, and deca double precision. The cost overhead factors of multiple double precision are multiplied with the cost overhead factors of truncated power series arithmetic. This cost overhead justifies the application of multitasking to write parallel software. Using all kernel threads on a 44-core computer, numerical experiments that took about 5 hours are estimated to take more than four days without multitasking.

The efficiency of the current implementation is limited by the medium grained parallelism and may not scale well on shared memory computers with over one hundred cores. In refining the granularity of the current implementation, the Ada 202X parallel features look promising.

Acknowledgments. The author thanks Clyde Roby, Tucker Taft, and Richard Wai, the organization committee of the HILT 2020 Workshop on Safe Languages and Technologies for Structured and Efficient Parallel and Distributed/Cloud Computing. Earlier versions of the software were presented in the Ada devroom at FOSDEM 2020. The author is grateful

precision	output	degree 8	degree 16	degree 24	degree 32
quad	$\|\Delta \mathbf{x}\|$	2.2E−30	1.6E+3		
	#iterations	8	8		
	seconds	56	168		
penta	$\|\Delta \mathbf{x}\|$	1.1E−47	1.1E−14	4.1E+19	
	#iterations	5	8	12	
	seconds	46	231	722	
octo	$\|\Delta \mathbf{x}\|$	1.4E−69	9.5E−63	3.8E−30	3.4E+3
	#iterations	5	6	12	16
	seconds	128	472	1,934	4,400
deca	$\|\Delta \mathbf{x}\|$	1.4E−69	2.4E−95	1.2E−62	1.1E−29
	#iterations	5	6	7	16
	seconds	222	807	1,952	7,579

Table 3: Newton's method for the power series expansion of a generic point on a surface of cyclic 128-roots, for truncation degrees 8, 16, 24, and 32, for quad, penta, octo, and deca double precision. The seconds record the wall clock time with 88 threads.

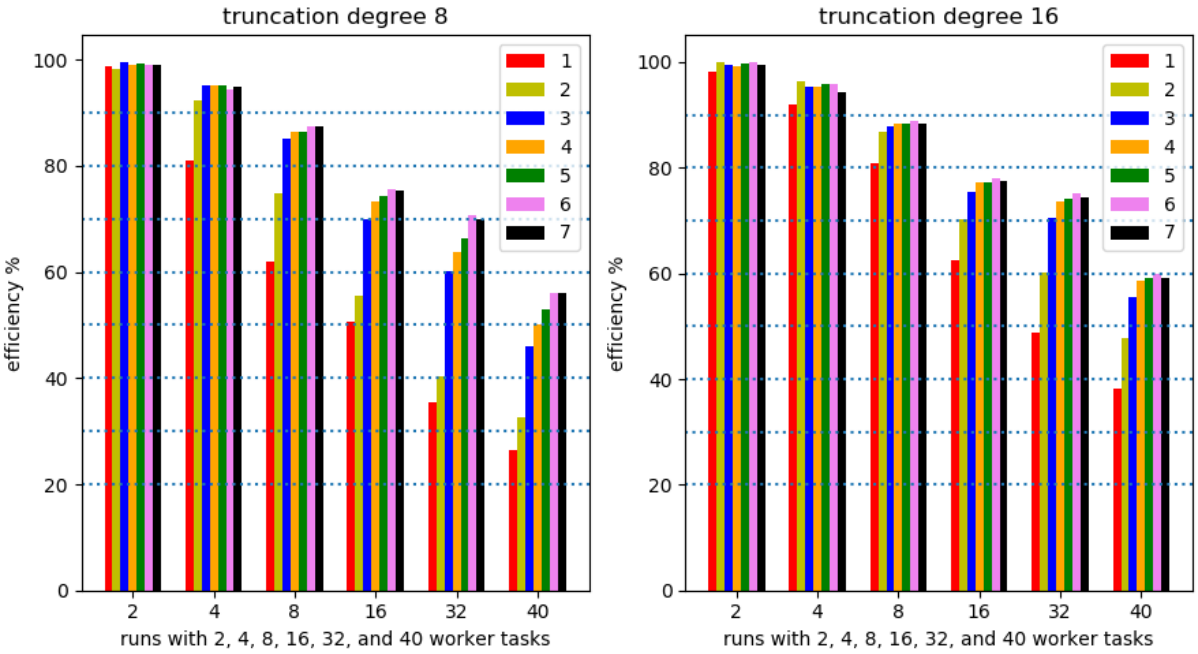


Figure 2: Efficiency plots for power series truncated to degrees 8 and 16, for 2, 4, 8, 16, 32, and 40 worker tasks, for seven precisions: double (d), 2-d, 3-d, 4-d, 5-d, 8-d, and 10-d, in the plots labeled respectively by bars 1, 2, 3, 4, 5, 6, and 7.

to Dirk Craeynest and Jean-Pierre Rosen for the organization of the FOSDEM 2020 Ada devroom.

References

[1] G. A. Baker and P. Graves-Morris. *Padé Approximants*, volume 59 of *Encyclopedia of Mathematics and its Applications*. Second edition, Cambridge University Press, 1996.

[2] A. Burns and A. Wellings. *Concurrent and Real-Time Programming in Ada.* Cambridge University Press, 2007.

[3] N. Bliss and J. Verschelde. The method of Gauss-Newton to compute power series solutions of polynomial homotopies. *Linear Algebra and Its Applications* 542:569–588, 2018.

[4] G. H. Golub and C. F. Van Loan. *Matrix Computations.* The Johns Hopkins University Press, 1983.

[5] A. Griewank and A. Walther. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. SIAM, 2008.

[6] Y. Hida, X. S. Li, and D. H. Bailey. Algorithms for quad-double precision floating point arithmetic. In the *Proceedings of the 15th IEEE Symposium on Computer Arithmetic (Arith-15 2001)*, pages 155–162. IEEE Computer Society, 2001.

[7] M. Joldes, J.-M. Muller, V. Popescu, W. Tucker. CAMPARY: Cuda Multiple Precision Arithmetic Library and Applications. In *Mathematical Software – ICMS 2016, the 5th International Conference on Mathematical Software*, pages 232–240, Springer-Verlag, 2016.

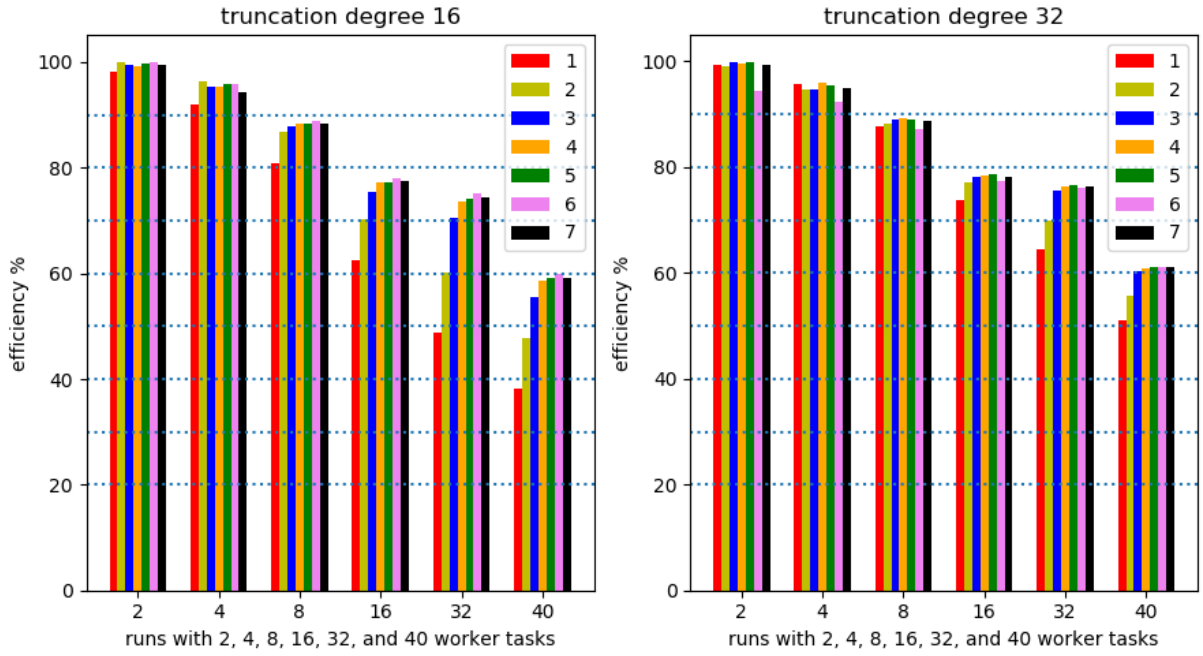


Figure 3: Efficiency plots for power series truncated to degrees 16 and 32, for 2, 4, 8, 16, 32, and 40 worker tasks, for seven precisions: double (d), 2-d, 3-d, 4-d, 5-d, 8-d, and 10-d, in the plots labeled respectively by bars 1, 2, 3, 4, 5, 6, and 7.

[8] J. W. McCormick, F. Singhoff, and J. Hugues. *Building Parallel, Embedded, and Real-Time Applications with Ada.* Cambridge University Press, 2011.

[9] A. Morgan. *Solving Polynomial Systems using Continuation for Engineering and Scientific Problems*, volume 57 of *Classics in Applied Mathematics*. SIAM, 2009.

[10] J.-M. Muller, N. Brunie, F. de Dinechin, C.-P. Jeannerod, M. Joldes, V. Lefevre, G. Melquiond, N. Revol, S. Torres. *Handbook of Floating-Point Arithmetic.* Second Edition, Springer-Verlag, 2018.

[11] S. M. Rump. Verification methods: Rigorous results using floating-point arithmetic. *Acta Numerica* 19:287–449, 2010.

[12] S. Telen, M. Van Barel, and J. Verschelde. A robust numerical path tracking algorithm for polynomial homotopy continuation. *SIAM Journal on Scientific Computing* 42(6):A3610–A3637, 2020.

[13] S. Telen, M. Van Barel, and J. Verschelde. Robust numerical tracking of one path of a polynomial homotopy on parallel shared memory computers. In the *Proceedings of the 22nd International Workshop on Computer Algebra in Scientific Computing (CASC 2020)*, volume 12291 of *Lecture Notes in Computer Science*, pages 563–582. Springer-Verlag, 2020.

[14] J. Verschelde. Algorithm 795: PHCpack: A general-purpose solver for polynomial systems by homotopy continuation. *ACM Transactions on Mathematical Software* 25(2):251–276, 1999. Runs online at `www.phcpack.org`.

[15] R. J. Walker. *Algebraic Curves.* Princeton University Press, 1950.

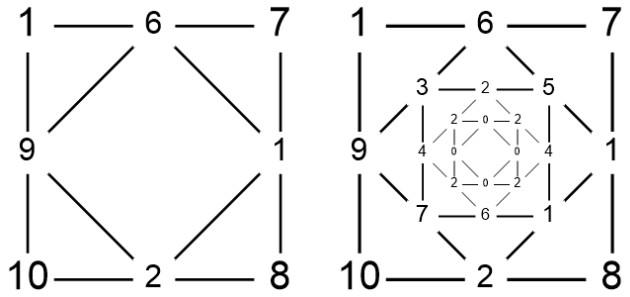
Shrinking Squares and Colourful Cubes

John Barnes

11 Albert Road, Caversham, Reading, RG4 7AN, UK; Tel: +44 118 9474125; email: john@jbinformatics.co.uk

Hello readers

The new puzzle this time is an easy one about cubes. But first we have to look at the nasty nested squares from last time. The numbers at the corner of an inner square are the differences between the numbers at the corners of the outer square. The sample shown was as below.



In that example it took five iterations to get all zeroes. The question is what is the maximum number of iterations required? The overall brief answer is that it is unlimited. But it depends upon the ordering of the original numbers. There are three possibilities:

1) Largest and smallest are opposite. 2) Largest and smallest adjacent, second next to largest. 3) Largest and smallest adjacent, second next to smallest. The example above is of type 3.

In type 1, it always converges in at most six iterations. In type 2, it always converges in at most four iterations. These are easy to prove. For example, here is the proof for type 2.

Suppose the numbers around the square are *a*, *b*, *c*, and *d* with *a* being the minimum and *d* the maximum. So we have

$a < c < b < d$

also we set $x = |(d - b) - (c - a)|$, that is the absolute value of the difference between *d* minus *b* and *c* minus *a*. Then the sequence of numbers per iteration goes as follows

0:	*a*	*b*	*c*	*d*
1:	*b–a*	*b–c*	*d–c*	*d–a*
2:	*d–b*	*c–a*	*d–b*	*c–a*
3:	*x*	*x*	*x*	*x*
4:	0	0	0	0

and hence it converges to zeroes on the fourth iteration.

Note that if *d–b* and *c–a* happen to be equal as in (1, 4, 3, 6), then *x* itself is zero so it converges in just three iterations.

Type 1 where the largest and smallest are opposite can be proved to converge in six iterations in a similar way.

However, in type 3, there is no limit at all. It is fairly easy to write a nice Ada program to explore various samples. The smallest to require seven iterations is (0, 1, 2, 4) thus

0		1		2		4	
	1		1		2		4
3		0		1		2	
	3		1		1		1
2		2		0		0	
	0		2		0		2
2		2		2		2	
	0		0		0		0

Amazingly, no matter what numbers we start with, in the final step every number is a power of 2. For example, the smallest sequence that takes eight iterations starts with 0, 1, 4, 9 and has penultimate sequence 4, 4, 4, 4 thus

0		1		4		9	
	1		3		5		9
8		2		2		4	
	6		0		2		4
2		6		2		2	
	4		4		0		0
4		0		4		0	
	4		4		4		4
0		0		0		0	

These smallest sequences form groups of three according to the power of 2 for the penultimate stage. It is convenient to assume that the smallest number is zero.

We find that the smallest sequences ending with $4 = 2^2$ are the group, which we can call G(2)

8:	1	4	9
9:	2	5	11
10:	2	6	13

where the first number is the sequence length and the other three together with zero are the smallest initial sequence. Those ending with $8 = 2^3$ are the group G(3) thus

11:	5	14	31
12:	6	17	37
13:	7	20	44

The next two groups are G(4) ending with $16 = 2^4$

14:	17	48	105
15:	20	57	125
16:	24	68	149

and G(5) ending with $32 = 2^5$ thus

17:	57	162	355
18:	68	193	423
19:	81	230	504

Surprisingly, by just looking, we see that these groups are related by the following recurrence relation

$G(n) = 3G(n–1) + G(n–2) + G(n–3)$

For example, taking $n = 5$ and considering the top row in each group we find

$57 = 3×17+5+1;\ \ 162 = 3×48+14+4;\ \ 355 = 3×105+31+9$

so it works! Readers might recall that such a recurrence relation can be solved by considering the cubic equation

$x^3 – 3x^2 – 3x – 1 = 0$

The real root is about 3.382975768. This means that the ratio between groups G(n–1) and G(n) should be close to this for large n. The largest numbers in groups 4 and 5 are 149 and 504 and their ratio is 3.38255....which is quite close.

We can easily calculate higher groups by simply using the recurrence relation. For example the bottom rows of G(16) and G(17) are

52:	53798080	152748176	334745777
55:	181997601	516743378	1132436852

The ratio of the bottom right numbers 334745777 and 1132436852 (using my c1985 HP 15C pocket calculator) is 3.382975768 which is overwhelmingly convincing.

We have seen that the sequences appear to go on for ever. And that they are grouped in threes. Moreover, that the groups satisfy a curious recurrence relation. But we haven't actually proved anything yet.

Suppose we start with $(0, x, y, z)$ where $0 < x < y < z$. Then the next sequence down is $(x, y–x, z–y, z)$ and after the first term is reduced to zero is $(0, a, b, c)$ where

$a = y – 2x$
$b = z – x – y$
$c = z – x$

However, given a, b, and c, these three equations can also be treated as equations for the three unknowns x, y, z in terms of a, b, c. This gives

$x = (c – a – b)/2$
$y = c – b$
$z = (3c – a – b)/2$

We should check that if $0 < a < b < c$ then the same property applies to x, y, z. This is easily done and is left to the reader. Thus we have shown that the process goes on for ever. If a, b, c take n iterations to reach all zeroes, then x, y, z will take n+1 iterations.

Better just check that it works. Consider group G(3). In the case of 5, 14, 31 we get

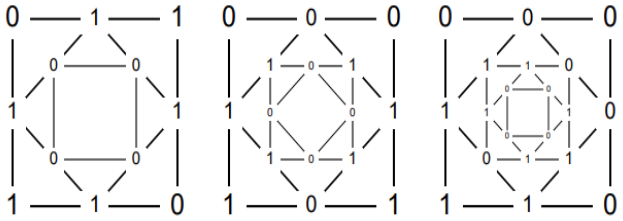
$x = (31 – 5 – 14)/2 = 12/2 = 6$
$y = 31 – 14 = 17$
$z = (3×31 – 5 – 14)/2 = 74/2 = 37$

So that is correct. Similarly (6, 17, 37) leads to (7, 20, 44). But if we try it again we get

$x = (44 – 7 – 20)/2 = 17/2 = 8½$
$y = 44 –20 = 24$
$z = (3×44 – 7 – 20)/2 = 105/2 = 52½$

which is not as expected because of the halves. So the sequence has to be doubled to give (17, 48, 105). That explains why the groups of three occur. And why it then goes on for ever as integers. And also why the sequences end with powers of 2.

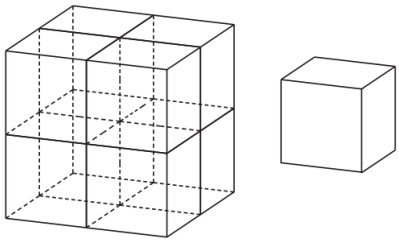
Just for fun here are the short sequences of lengths 2, 3, and 4. Note that they end with 1, 1, 1, 1. Remember that 2^0 is 1.



Well that's quite enough about the squares. One might ask what happens if we try doing nested triangles in the same way. It's a bit boring and often gets stuck in a rut with values oscillating at 1, 0, 1. One problem is that the smallest and largest numbers are always adjacent and the third number is in between. I suppose one could try doing it with a pentagon.

And now for something quite different and much easier. How many ways can one colour a cube with six different colours, one colour on each face? Hint: it's more than 10.

And suppose we have such a set of different coloured cubes. Pick one out and from the remainder find eight that can be put together 2 by 2 by 2 to make a large cube that matches the one removed in such a way that not only do the faces match externally but the faces touching internally also match. Thus



Another example of cubes with different faces is provided by dice. In this case there are six numbers as opposed to six colours. Moreover, real dice always have the 6 opposite the 1, the 5 opposite 2, and 4 opposite 3. However, the pips for two and three can slope one way or the other and the six can be done as 2 by 3 or 3 by 2. So there are quite a few kinds of dice.

The UK delegation at Portland in 2014 (Jeff Cousins, head of delegation, and I) drove up the Columbia river and stopped for refreshment at a hotel on Mt Hood. We encountered some ladies playing Craps. Their dice were an amazing jumble! How many different kinds can you find?

In Memoriam: William (Bill) Bail

William (Bill) Bail, well known to many Ada-Europe conference participants, passed away suddenly at his home on Monday 7 December 2020.

His daughter Evin wrote: *"His final day was not expected to be his last, but he did what he loved... his work. His passion for computer science and his integrity fueled him in his 30 years at MITRE. He earned his PhD in computer science in the mid-80s when the field was ever changing. He was active in Ada circles and Ada-Europe."*

Informing us of Bill's passing, his wife Julie wrote: *"You may share this information with anyone at Ada-Europe. He loved the conferences and would come back with wonderful reports on whom he saw and how much he enjoyed seeing his friends. Thanks for helping me to get the word out to the people who were so important to him."*

Bill was active in the Ada community, including almost two decades of Ada-Europe conferences (2001-2017) as well as ACM SIGAda events. He was a regular participant and frequent contributor to the Ada-Europe conferences, and taught numerous tutorials over the years. Bill was an experienced presenter, his tutorials were very well prepared and appreciated by the attendees. He always made various submissions, in order to let the organizers choose the most appropriate ones, matching their preferences and the schedule of the event. Being more software engineering related, his proposals were of a different "nature" than many of the others, which provided a good mix of topics to participants.

In his (in hindsight) last e-mails, after we announced our plans to hold the 2021 conference in the originally planned location for 2020 (Santander, Spain), he reminisced about past Ada-Europe conferences: *"If you remember my first was Leuven, a wonderful event, and almost every one since then until Lisbon. Wonderful people, wonderful discussions…"* and *"I still regret every day missing Lisbon and Warsaw since these meetings are such wonderful events – I have enjoyed every one I have been to, made all the more rewarding because of the people attending with whom I feel closely connected after all these years (attended 14 conferences since 2001 in Leuven, only missed 3 since then until Lisbon)."*

His last sentence was: *"Here's hoping that we can meet again at the next conference in Spain."*

Alas...

Dirk Craeynest, Ada-Europe Board, December 2020

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

Ada-Europe Sponsors

27 Rue Rasson
B-1030 Brussels, Belgium
Contact:Ludovic Brenta
ludovic@ludovic-brenta.org

46 Rue d'Amsterdam
F-75009 Paris, France
Contact: Jamie Ayre
sales@adacore.com
www.adacore.com

2 Rue Docteur Lombard
92441 Issy-les-Moulineaux Cedex
France
Contact: Jean-Pierre Rosen
rosen@adalog.fr
www.adalog.fr/en/

22 St. Lawrence Street
Southgate
Bath BA1 1AN, United Kingdom
Contact: Stuart Matthews
sparkinfo@altran.com
www.altran.co.uk

Jacob Bontiusplaats 9
1018 LL Amsterdam
The Netherlands
Contact: Wido te Brake
wido.tebrake@deepbluecap.com
www.deepbluecap.com

24 Quai de la Douane
29200 Brest, Brittany
France
Contact: Pierre Dissaux
pierre.dissaux@ellidiss.com
www.ellidiss.com

In der Reiss 5
D-79232 March-Buchheim
Germany
Contact: Frank Piron
info@konad.de
www.konad.de

3271 Valley Centre Drive,
Suite 300
San Diego, CA 92069, USA
Contact: Shawn Fanning
sfanning@ptc.com
www.ptc.com/developer-tools

Signal Business Centre
2 Innotec Drive, Bangor
North Down BT19 7PD
Northern Ireland, UK
enquiries@sysada.co.uk
www.sysada.co.uk

1090 Rue René Descartes
13100 Aix en Provence, France
Contact: Patricia Langle
patricia.langle@systerel.fr
www.systerel.fr/en/

Tiirasaarentie 32
FI 00200 Helsinki, Finland
Contact: Niklas Holsti
niklas.holsti@tidorum.fi
www.tidorum.fi

Millennium Tower, floor 41
Handelskai 94-96
A-1200 Austria
Contact: Massimo Bombino
sales@at.vector.com
www.vector.com

Beckengässchen 1
8200 Schaffhausen
Switzerland
Contact: Ahlan Marriott
admin@white-elephant.ch

United Kingdom
Contact: Chris Nettleton
nettelton@xgc.com
www.xgc.com


Ada
europe